DevOps · Level MAX

[Sổ tay toàn diện cho DevOps Engineer]

Do Cao Hieu

DevOps Engineer Handbook

Level MAX - Theo Roadmap.sh

Đỗ Cao Hiếu

Tháng 3, 2026

Contents

Chapter 1: Programming Languages cho DevOps

DevOps engineer cần thành thạo ít nhất một scripting language (Python hoặc Bash), hiểu Go để đọc/viết tooling. Mỗi ngôn ngữ có vị trí riêng trong ecosystem.

---

1. Python cho Automation

Python là ngôn ngữ số 1 cho DevOps automation nhờ ecosystem phong phú, readable syntax và cross-platform support.

1.1 Thư viện cốt lõi

```
import os
import subprocess
import sys
from pathlib import Path
```

os module - tương tác với OS:

```
# Environment variables
db_host = os.environ.get("DB_HOST", "localhost")
os.environ["APP_ENV"] = "production"

# File/directory operations
os.makedirs("/opt/app/logs", exist_ok=True)
os.path.exists("/etc/nginx/nginx.conf")

# Walk directory tree
for root, dirs, files in os.walk("/var/log"):
    for file in files:
        if file.endswith(".log"):
            print(os.path.join(root, file))
```

subprocess module - chạy shell commands:

```
import subprocess

# Capture output
result = subprocess.run(
    ["docker", "ps", "--format", "{{.Names}}"],
    capture_output=True,
    text=True,
    check=True  # raise CalledProcessError nếu exit code != 0
)
containers = result.stdout.strip().split("\n")

# Shell pipeline
```

```
result = subprocess.run(
    "ps aux | grep nginx | wc -l",
    shell=True,
    capture_output=True,
    text=True
)

# Streaming output (cho long-running commands)
process = subprocess.Popen(
    ["tail", "-f", "/var/log/app.log"],
    stdout=subprocess.PIPE,
    text=True
)
for line in process.stdout:
    print(line, end="")
```

1.2 HTTP và API

requests - HTTP client:

```
import requests

# Health check endpoint
def check_service_health(url: str, timeout: int = 5) -> bool:
    try:
        resp = requests.get(url, timeout=timeout)
        return resp.status_code == 200
    except requests.exceptions.ConnectionError:
        return False

# POST với JSON body
def trigger_deployment(api_url: str, token: str, payload: dict):
    headers = {"Authorization": f"Bearer {token}", "Content-Type": "application/json"}
    resp = requests.post(f"{api_url}/deploy", json=payload, headers=headers)
    resp.raise_for_status()
    return resp.json()
```

1.3 AWS với boto3

```
import boto3

# EC2 - list running instances
ec2 = boto3.client("ec2", region_name="ap-southeast-1")
response = ec2.describe_instances(
    Filters=[{"Name": "instance-state-name", "Values": ["running"]}]
)
```

```python
for reservation in response["Reservations"]:
    for instance in reservation["Instances"]:
        name = next(
            (t["Value"] for t in instance.get("Tags", []) if t["Key"] == "Name"),
            "unnamed"
        )
        print(f"{name}: {instance['InstanceId']} ({instance['PrivateIpAddress']})")

# S3 - upload file
s3 = boto3.client("s3")
s3.upload_file("build.tar.gz", "my-artifacts-bucket", "releases/v1.2.3/build.tar.gz")

# SSM Parameter Store - đọc secrets
ssm = boto3.client("ssm")
param = ssm.get_parameter(Name="/prod/db/password", WithDecryption=True)
db_password = param["Parameter"]["Value"]
```

1.4 Fabric - Remote Execution

```python
from fabric import Connection

def deploy_to_server(host: str, user: str, key_path: str, version: str):
    conn = Connection(
        host=host,
        user=user,
        connect_kwargs={"key_filename": key_path}
    )
    with conn:
        # Pull latest code
        conn.run(f"cd /opt/app && git pull && git checkout {version}")
        # Restart service
        conn.sudo("systemctl restart app.service")
        # Verify
        result = conn.run("systemctl is-active app.service", warn=True)
        if result.stdout.strip() != "active":
            raise Exception(f"Service not active on {host}")
```

1.5 Ví dụ thực tế: Log Parser

```python
#!/usr/bin/env python3
"""Parse nginx access logs và tìm top slow requests."""

import re
import sys
from collections import defaultdict
from pathlib import Path
```

```
LOG_PATTERN = re.compile(
    r'(?P<ip>\S+) \S+ \S+ \[.*?\] "(?P<method>\S+) (?P<path>\S+).*?" '
    r'(?P<status>\d+) \d+ ".*?" ".*?" (?P<response_time>[\d.]+)'
)

def parse_log(log_file: str, threshold_ms: float = 1000):
    slow_requests = []
    error_count = defaultdict(int)

    for line in Path(log_file).read_text().splitlines():
        match = LOG_PATTERN.match(line)
        if not match:
            continue
        data = match.groupdict()
        rt = float(data["response_time"]) * 1000  # convert to ms
        status = int(data["status"])

        if status >= 500:
            error_count[data["path"]] += 1
        if rt > threshold_ms:
            slow_requests.append((rt, data["path"], data["method"]))

    slow_requests.sort(reverse=True)
    print(f"\nTop 10 slow requests (>{threshold_ms}ms):")
    for rt, path, method in slow_requests[:10]:
        print(f"  {method} {path}: {rt:.0f}ms")

    print(f"\nTop 5 error endpoints:")
    for path, count in sorted(error_count.items(), key=lambda x: -x[1])[:5]:
        print(f"  {path}: {count} errors")

if __name__ == "__main__":
    parse_log(sys.argv[1] if len(sys.argv) > 1 else "/var/log/nginx/access.log")
```

---

2. Go cho Tooling

Go lý tưởng cho CLI tools, daemons, và distributed systems. Binary compilation, fast startup, built-in concurrency.

2.1 CLI Tool với cobra

```
// cmd/healthcheck/main.go
package main
```

```
import (
    "fmt"
    "net/http"
    "os"
    "time"

    "github.com/spf13/cobra"
)

var rootCmd = &cobra.Command{
    Use:   "healthcheck [url]",
    Short: "Check HTTP endpoint health",
    Args:  cobra.ExactArgs(1),
    Run: func(cmd *cobra.Command, args []string) {
        timeout, _ := cmd.Flags().GetInt("timeout")
        client := &http.Client{Timeout: time.Duration(timeout) * time.Second}

        resp, err := client.Get(args[0])
        if err != nil {
            fmt.Fprintf(os.Stderr, "ERROR: %v\n", err)
            os.Exit(1)
        }
        defer resp.Body.Close()

        if resp.StatusCode >= 200 && resp.StatusCode < 300 {
            fmt.Printf("OK: %s returned %d\n", args[0], resp.StatusCode)
        } else {
            fmt.Printf("FAIL: %s returned %d\n", args[0], resp.StatusCode)
            os.Exit(1)
        }
    },
}

func main() {
    rootCmd.Flags().Int("timeout", 10, "timeout in seconds")
    if err := rootCmd.Execute(); err != nil {
        os.Exit(1)
    }
}
```

2.2 Concurrent Health Check nhiều endpoints

```
package main

import (
    "fmt"
    "net/http"
```

```
    "sync"
    "time"
)

type CheckResult struct {
    URL    string
    Status int
    Err    error
    RTms   float64
}

func checkURL(url string) CheckResult {
    start := time.Now()
    resp, err := http.Get(url)
    rt := float64(time.Since(start).Milliseconds())
    if err != nil {
        return CheckResult{URL: url, Err: err, RTms: rt}
    }
    defer resp.Body.Close()
    return CheckResult{URL: url, Status: resp.StatusCode, RTms: rt}
}

func checkAll(urls []string) []CheckResult {
    results := make([]CheckResult, len(urls))
    var wg sync.WaitGroup
    for i, url := range urls {
        wg.Add(1)
        go func(idx int, u string) {
            defer wg.Done()
            results[idx] = checkURL(u)
        }(i, url)
    }
    wg.Wait()
    return results
}
```

2.3 Cross-compilation

```
# Build cho nhiều platform từ Linux
GOOS=linux   GOARCH=amd64  go build -o dist/tool-linux-amd64   .
GOOS=linux   GOARCH=arm64  go build -o dist/tool-linux-arm64   .
GOOS=darwin  GOARCH=arm64  go build -o dist/tool-darwin-arm64  .
GOOS=windows GOARCH=amd64  go build -o dist/tool-windows.exe   .
```

---

3. Bash Scripting

Bash là glue language của Linux. Dùng cho system automation, CI/CD scripts, bootstrap scripts.

3.1 Cấu trúc script chuẩn

```
#!/usr/bin/env bash
set -euo pipefail  # e=exit on error, u=unset var error, o pipefail

readonly SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
readonly LOG_FILE="/var/log/deploy.log"

log() { echo "[$(date '+%Y-%m-%d %H:%M:%S')] $*" | tee -a "$LOG_FILE"; }
error() { log "ERROR: $*" >&2; exit 1; }

# Cleanup on exit
cleanup() {
    local exit_code=$?
    log "Script exiting with code: $exit_code"
    rm -f /tmp/deploy.lock
}
trap cleanup EXIT
trap 'error "Interrupted"' INT TERM
```

3.2 Variables, Arrays, Functions

```
# Variables và string operations
APP_VERSION="${APP_VERSION:-1.0.0}"           # default value
APP_NAME="${APP_NAME:?'APP_NAME is required'}" # required variable

# Substring: ${var:offset:length}
short_hash="${GIT_COMMIT:0:8}"

# Arrays
servers=("web-01" "web-02" "web-03")
for server in "${servers[@]}"; do
    echo "Deploying to $server"
done
echo "Total servers: ${#servers[@]}"

# Associative arrays (Bash 4+)
declare -A service_ports
service_ports["nginx"]=80
service_ports["postgres"]=5432
service_ports["redis"]=6379
```

```
for svc in "${!service_ports[@]}"; do
    echo "$svc runs on port ${service_ports[$svc]}"
done
```

3.3 Error Handling và Exit Codes

```
# Explicit exit codes
readonly EXIT_OK=0
readonly EXIT_ERROR=1
readonly EXIT_USAGE=2

check_dependency() {
    local cmd="$1"
    if ! command -v "$cmd" &>/dev/null; then
        error "Required command not found: $cmd"
    fi
}

retry() {
    local max_attempts="$1"
    local delay="$2"
    shift 2
    local attempt=1
    until "$@"; do
        if (( attempt >= max_attempts )); then
            error "Command failed after $max_attempts attempts: $*"
        fi
        log "Attempt $attempt failed. Retrying in ${delay}s..."
        sleep "$delay"
        (( attempt++ ))
    done
}

# Usage: retry 3 5 curl -f http://localhost/health
```

3.4 Ví dụ thực tế: Deploy Script

```
#!/usr/bin/env bash
set -euo pipefail

APP="myapp"
DEPLOY_DIR="/opt/$APP"
BACKUP_DIR="/opt/backups/$APP"
VERSION="${1:?'Usage: $0 <version>'}"

log() { echo "[$(date '+%H:%M:%S')] $*"; }
```

```
pre_deploy_checks() {
    log "Running pre-deploy checks..."
    check_dependency docker
    check_dependency docker-compose
    # Verify image exists
    docker pull "registry.example.com/$APP:$VERSION" || error "Image not found"
}

backup_current() {
    local ts; ts=$(date '+%Y%m%d_%H%M%S')
    mkdir -p "$BACKUP_DIR"
    cp "$DEPLOY_DIR/.env" "$BACKUP_DIR/.env.$ts" 2>/dev/null || true
    log "Backup created at $BACKUP_DIR/.env.$ts"
}

deploy() {
    log "Deploying $APP version $VERSION..."
    export APP_VERSION="$VERSION"
    docker-compose -f "$DEPLOY_DIR/docker-compose.yml" up -d --no-deps app
}

health_check() {
    log "Waiting for health check..."
    retry 10 3 curl -sf http://localhost:8080/health
    log "Health check passed"
}

pre_deploy_checks
backup_current
deploy
health_check
log "Deployment of $VERSION completed successfully"
```

---

4. So sánh: Khi nào dùng ngôn ngữ nào?

Tình huống	Ngôn ngữ	Lý do
Quick one-off script	Bash	Zero dependencies, available everywhere
Automation phức tạp (>100 lines)	Python	Readability, error handling tốt hơn
AWS/Cloud API integration	Python (boto3)	Ecosystem phong phú
CLI tool distributed cho team	Go	Single binary, fast startup

Tình huống	Ngôn ngữ	Lý do
System daemon / agent	Go	Low memory, concurrency, no runtime
Data processing / ML pipeline	Python	Pandas, NumPy, rich ecosystem
Bootstrap / provisioning scripts	Bash	Chạy được trên bare metal

Quy tắc thực tế: - Script < 50 lines, không có logic phức tạp → Bash - Cần HTTP, JSON parsing, retry logic → Python - Cần distribute binary, concurrency cao → Go - CI/CD pipeline steps → Bash (gọi Python/Go tools khi cần)

---

5. Best Practices

Python

```python
# Dùng type hints cho readability
def deploy_service(name: str, version: str, dry_run: bool = False) -> bool: ...

# Dùng dataclasses thay dict cho structured data
from dataclasses import dataclass

@dataclass
class DeployConfig:
    app_name: str
    version: str
    replicas: int = 1
    timeout: int = 300
```

Bash

```bash
# Luôn dùng shellcheck để lint
shellcheck deploy.sh

# Quote mọi variables
cp "$src" "$dst"  # không phải cp $src $dst

# Dùng [[ ]] thay [ ] cho conditionals
[[ -f "$config" ]] && source "$config"

# Avoid using ls in scripts, dùng glob
for f in /var/log/*.log; do ...
```

```
# Prefer printf over echo
printf "Version: %s\n" "$VERSION"
```

Go

```
# Run linter
golangci-lint run ./...

# Test với race detector
go test -race ./...

# Build với version info
go build -ldflags="-X main.version=$(git describe --tags)" .
```

Testing Scripts

```
# Unit test Bash scripts với bats-core
# File: test/deploy.bats
@test "deploy fails without version argument" {
    run ./deploy.sh
    [ "$status" -eq 2 ]
}

# Test Python scripts
pytest tests/ -v --tb=short
```

Chapter 2: Operating System Concepts cho DevOps

Hiểu OS internals giúp DevOps engineer debug production issues, tune performance, và thiết kế hệ thống resilient. Chapter này tập trung vào Linux - nền tảng của hầu hết server infrastructure.

---

1. Process Management

1.1 Vòng đời của Process

Mỗi process có một **PID** (Process ID) và **PPID** (Parent PID). Kernel quản lý process thông qua **process table**.

```
fork() → copy parent process (copy-on-write)
exec() → replace process image với program mới
wait() → parent chờ child exit
```

Zombie process: Child đã exit nhưng parent chưa gọi `wait()` → entry vẫn trong process table.

```
ps aux | awk '$8 == "Z"'  # liệt kê zombie processes
```

Orphan process: Parent đã die trước child → kernel giao lại cho init (PID 1).

1.2 Signals

Signal	Number	Mô tả	Có thể bắt?
SIGHUP	1	Hangup / reload config	Có
SIGINT	2	Ctrl+C	Có
SIGKILL	9	Buộc kill, không thể block	Không
SIGTERM	15	Graceful shutdown	Có
SIGSTOP	19	Pause process	Không
SIGUSR1/2	10/12	User-defined	Có

```
# Gửi signals
kill -TERM 1234          # graceful shutdown
kill -HUP $(pidof nginx) # reload nginx config
kill -9 1234             # force kill (last resort)
pkill -f "python worker" # kill by name pattern
killall -TERM gunicorn

# Trap signals trong Bash
trap 'echo "Received SIGTERM, shutting down..."; cleanup; exit 0' SIGTERM
trap 'echo "Received SIGINT"; exit 130' SIGINT
```

1.3 Process Priority: nice / renice

Kernel scheduler dùng **nice value** (-20 đến +19, thấp hơn = ưu tiên cao hơn).

```
# Start process với low priority (backup job)
nice -n 19 rsync -av /data /backup

# Thay đổi priority của process đang chạy
renice -n 10 -p 1234
renice -n -5 -p $(pidof nginx)  # cần root

# Real-time priority (cho latency-sensitive workloads)
chrt -f 99 ./critical-process   # SCHED_FIFO, priority 99
```

1.4 Monitoring Commands

```
# ps - snapshot processes
ps aux --sort=-%cpu | head -20   # top CPU consumers
```

```
ps -eo pid,ppid,cmd,%cpu,%mem --sort=-%mem | head -10
ps -ef | grep "[n]ginx"          # dùng bracket trick để không match chính grep

# top / htop - interactive monitor
top -b -n 1 -o %CPU              # batch mode, sort by CPU
htop -p 1234,5678                # monitor specific PIDs

# pgrep / pstree
pgrep -a nginx                   # show PID + command
pstree -p 1                      # show process tree từ init
```

---

2. Threads vs Processes và Concurrency Models

2.1 Threads vs Processes

	Process	Thread
Memory space	Riêng	Shared với parent
Creation cost	Cao (fork)	Thấp (clone)
Isolation	Tốt	Kém (shared state)
Communication	IPC (pipe, socket, shm)	Shared memory trực tiếp
Crash impact	Isolated	Crash toàn process

Linux thực tế: Cả process và thread đều dùng `clone()` syscall, khác nhau ở flags chia sẻ memory space.

2.2 Concurrency Models trong Production

```
Apache (prefork)     → Multi-process, 1 request/process - stable nhưng heavy
Nginx / Node.js      → Event loop (epoll) - single thread, non-blocking I/O
Gunicorn             → Multi-process workers + async (gevent/uvicorn)
Go programs          → Goroutines - M:N threading trên OS threads
```

```
# Xem thread count của process
cat /proc/1234/status | grep Threads
ls /proc/1234/task | wc -l   # mỗi entry là một thread
```

---

3. Memory Management

3.1 Virtual Memory

Mỗi process có virtual address space riêng. Kernel map virtual → physical qua **page table**.

```
# Xem memory map của process
cat /proc/1234/maps
pmap -x 1234

# Memory stats tổng quan
free -h
              total        used        free      shared  buff/cache   available
Mem:           15G        8.2G        1.1G        512M        5.7G        6.1G
Swap:          4.0G       256M        3.7G
```

buff/cache: Kernel dùng free memory để cache disk I/O → bình thường. "available" quan trọng hơn "free".

3.2 Swap

```
# Xem swap usage
swapon --show
vmstat -s | grep -i swap

# swappiness - kernel tendency to use swap (0-100)
cat /proc/sys/vm/swappiness    # default: 60
sysctl vm.swappiness=10        # tuning cho database servers
echo "vm.swappiness=10" >> /etc/sysctl.conf  # persistent

# Flush swap (khi cần reclaim)
swapoff -a && swapon -a
```

3.3 OOM Killer

Khi memory exhausted, kernel OOM killer chọn và kill process dựa trên **oom_score**.

```
# Xem OOM score
cat /proc/1234/oom_score         # điểm cao hơn = dễ bị kill hơn
cat /proc/1234/oom_score_adj     # -1000 đến +1000

# Protect critical processes
echo -1000 > /proc/$(pidof sshd)/oom_score_adj    # không bao giờ kill sshd
echo 1000  > /proc/$(pidof chrome)/oom_score_adj  # kill chrome trước

# Xem OOM events trong kernel log
dmesg | grep -i "oom\|killed process"
journalctl -k | grep -i oom
```

3.4 cgroups (Control Groups)

cgroups giới hạn tài nguyên cho group processes. Docker, systemd, Kubernetes đều dùng cgroups.

```
# cgroups v2 (modern)
cat /sys/fs/cgroup/memory/docker/CONTAINER_ID/memory.current
cat /sys/fs/cgroup/memory/docker/CONTAINER_ID/memory.max

# Giới hạn memory cho systemd service
systemctl set-property app.service MemoryMax=512M CPUQuota=50%

# Xem resource limits của container
docker stats --no-stream
```

---

4. I/O Model

4.1 Blocking vs Non-blocking I/O

```
Blocking I/O:     Process gọi read() → block cho đến khi data available
Non-blocking I/O: Gọi read() → return ngay với EAGAIN nếu không có data
Async I/O:        Gọi aio_read() → kernel notify khi done qua signal/callback
```

4.2 epoll - Multiplexed I/O

Nginx, Redis, Node.js dùng epoll để handle hàng nghìn connections với single thread.

```
// Conceptual flow
int epfd = epoll_create1(0);
struct epoll_event ev = {.events = EPOLLIN, .data.fd = server_fd};
epoll_ctl(epfd, EPOLL_CTL_ADD, server_fd, &ev);

// Event loop
while (1) {
    int nfds = epoll_wait(epfd, events, MAX_EVENTS, -1);
    for (int i = 0; i < nfds; i++) {
        handle_event(events[i].data.fd);
    }
}
```

```
# Xem I/O wait
iostat -x 1 5           # extended I/O stats
iotop -o                # processes đang I/O
ioping -c 10 /dev/sda   # disk latency test
```

4.3 I/O Scheduler

```
# Xem scheduler hiện tại
cat /sys/block/sda/queue/scheduler
# Ví dụ: none mq-deadline [kyber] bfq
```

```
# Đổi scheduler (cho SSD nên dùng 'none' hoặc 'mq-deadline')
echo mq-deadline > /sys/block/sda/queue/scheduler
```

---

5. Filesystem

5.1 Filesystem Types

FS	Use case	Đặc điểm
ext4	General purpose	Stable, journaling, tốt cho nhỏ đến vừa
xfs	Large files, databases	Scale tốt, parallel I/O, default trên RHEL
btrfs	Snapshots, RAID	CoW, subvolumes, checksums
tmpfs	In-memory (/tmp, /run)	RAM-backed, mất khi reboot

5.2 Inodes và File Descriptors

```
# Inode usage (có thể hết inode dù còn disk space)
df -i
# Tìm directory dùng nhiều inode nhất
find / -xdev -printf '%h\n' | sort | uniq -c | sort -rn | head -10

# File descriptors
ulimit -n                       # per-process fd limit (thường 1024)
cat /proc/sys/fs/file-max       # system-wide fd limit
lsof -p 1234 | wc -l            # fd count của process
lsof -i :8080                   # process đang dùng port 8080
lsof +D /var/log                # files đang mở trong directory

# Tăng fd limit
echo "* soft nofile 65536" >> /etc/security/limits.conf
echo "* hard nofile 65536" >> /etc/security/limits.conf
sysctl -w fs.file-max=2097152
```

5.3 /proc và /sys

```
# /proc - virtual filesystem, kernel exposes process/system info
cat /proc/cpuinfo               # CPU info
cat /proc/meminfo               # memory details
cat /proc/net/tcp               # network connections
```

```
cat /proc/loadavg                  # load average + task count
cat /proc/1234/cmdline             # command line của process

# /sys - sysfs, hardware/driver configuration
ls /sys/block/                     # block devices
cat /sys/block/sda/size            # disk size in 512-byte sectors
echo 1 > /sys/block/sda/device/rescan  # rescan disk
```

---

6. Systemd

6.1 Unit Files

```
# /etc/systemd/system/app.service
[Unit]
Description=My Application
After=network.target postgresql.service
Requires=postgresql.service

[Service]
Type=notify
User=app
WorkingDirectory=/opt/app
ExecStart=/opt/app/bin/server --config /etc/app/config.yaml
ExecReload=/bin/kill -HUP $MAINPID
Restart=on-failure
RestartSec=5s
StandardOutput=journal
StandardError=journal

# Resource limits
MemoryMax=1G
CPUQuota=200%        # 2 CPU cores
LimitNOFILE=65536

# Security hardening
NoNewPrivileges=true
PrivateTmp=true
ReadOnlyPaths=/etc

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

```
# Manage services
systemctl start|stop|restart|reload app.service
systemctl enable app.service       # start on boot
```

```
systemctl status app.service        # xem trạng thái chi tiết
systemctl list-units --failed       # liệt kê failed units

# Reload after editing unit files
systemctl daemon-reload

# Check dependencies
systemctl list-dependencies app.service
```

6.2 journalctl - Log Management

```
journalctl -u app.service          # logs của service
journalctl -u app.service -f       # follow (như tail -f)
journalctl -u app.service --since "1 hour ago"
journalctl -u app.service -p err   # chỉ errors (emerg, alert, crit, err)
journalctl --disk-usage            # xem log size
journalctl --vacuum-size=500M      # cleanup logs
journalctl -k                      # kernel messages (như dmesg)
journalctl -b                      # logs từ lần boot hiện tại
journalctl -b -1                   # logs từ lần boot trước
```

---

7. Performance Monitoring Commands

```
# vmstat - virtual memory stats
vmstat 1 10   # interval=1s, count=10
# r: run queue, b: blocked, si/so: swap in/out, us/sy/id/wa: CPU %

# iostat - I/O stats
iostat -xz 1  # extended, skip zero, every 1s
# %util: disk utilization, await: avg wait time, r/s+w/s: IOPS

# sar - System Activity Reporter (từ sysstat package)
sar -u 1 10   # CPU utilization
sar -r 1 10   # memory
sar -b 1 10   # I/O
sar -n DEV 1  # network
sar -A        # tất cả metrics

# strace - trace system calls (debug tool)
strace -p 1234              # attach to running process
strace -e trace=open,read,write ./app  # filter specific syscalls
strace -c ./app             # count syscalls - summary

# Tổng hợp nhanh: USE Method (Utilization, Saturation, Errors)
```

```
# CPU: sar -u, Saturation: vmstat r column, Errors: dmesg
# Memory: free -h, Saturation: vmstat si/so, Errors: dmesg OOM
# Disk: iostat %util, Saturation: iostat await, Errors: dmesg
# Network: sar -n DEV, Saturation: ss -s, Errors: ip -s link
```

Chapter 3: Linux Terminal Mastery

Terminal là công cụ chính của DevOps engineer. Thành thạo terminal giúp bạn làm việc nhanh hơn 10x so với GUI tools, và đây là kỹ năng bắt buộc khi làm việc với remote servers.

---

1. Text Processing Tools

1.1 grep - Search Text

```
# Cơ bản
grep "ERROR" /var/log/app.log
grep -i "error" /var/log/app.log        # case insensitive
grep -r "TODO" /opt/app/src/            # recursive
grep -l "pattern" *.conf                # chỉ show filename
grep -v "DEBUG" app.log                 # invert match (exclude)
grep -c "ERROR" app.log                 # count matches
grep -n "pattern" file.txt              # show line numbers

# Extended regex
grep -E "ERROR|WARN|CRITICAL" app.log
grep -E "^[0-9]{4}-[0-9]{2}" app.log  # lines starting with date
grep -P '\d{3}-\d{4}' file.txt        # Perl regex

# Context
grep -A 5 "Exception" app.log          # 5 lines After
grep -B 3 "Exception" app.log          # 3 lines Before
grep -C 2 "Exception" app.log          # 2 lines Context (both)

# Multiple patterns
grep -E "OOM|killed" /var/log/kern.log
grep -f patterns.txt app.log           # patterns từ file
```

1.2 awk - Column Processing

awk là mini programming language, tốt nhất cho structured text (columns).

```
# Syntax: awk 'pattern { action }' file
# Built-in: $0=whole line, $1/$2/...=fields, NF=num fields, NR=line number
```

```
# Print specific columns
awk '{print $1, $4}' access.log                    # print col 1 và 4
awk -F: '{print $1}' /etc/passwd                   # custom delimiter
awk -F',' '{print $2}' data.csv                    # CSV

# Filtering + processing
awk '$9 >= 500' access.log                         # HTTP 5xx errors
awk '/ERROR/ {print NR": "$0}' app.log             # errors với line number
awk 'NR>=10 && NR<=20' file.txt                    # print lines 10-20

# Aggregation
awk '{sum += $NF} END {print "Total:", sum}' numbers.txt
awk '{count[$1]++} END {for (ip in count) print ip, count[ip]}' access.log

# Real example: tổng response time từ nginx log
awk '{sum += $NF; count++} END {printf "Avg: %.2fms\n", sum/count*1000}' access.log

# Disk usage report
df -h | awk 'NR>1 && $5+0 > 80 {print "WARN: "$6, $5, "full"}'

# Xử lý multi-file với FILENAME
awk 'FNR==1 {print "=== " FILENAME " ==="}' *.log
```

1.3 sed - Stream Editor

sed tốt cho text transformation, substitution trong pipelines.

```
# Substitution: s/pattern/replacement/flags
sed 's/foo/bar/' file.txt              # replace first occurrence per line
sed 's/foo/bar/g' file.txt             # replace all occurrences
sed 's/foo/bar/gi' file.txt            # case insensitive
sed -i 's/old/new/g' file.txt          # in-place edit
sed -i.bak 's/old/new/g' file.txt      # in-place với backup

# Delete lines
sed '/^#/d' config.conf                # xóa comment lines
sed '/^$/d' file.txt                   # xóa blank lines
sed '5,10d' file.txt                   # xóa lines 5-10

# Print specific lines
sed -n '10,20p' file.txt               # print lines 10-20
sed -n '/START/,/END/p' file.txt       # print between markers

# Insert/append
sed '5a\New line after line 5' file.txt
sed '5i\New line before line 5' file.txt
```

```
# Real examples
# Update version trong config
sed -i "s/version: .*/version: ${NEW_VERSION}/" config.yaml

# Extract value từ config file
sed -n 's/^DB_HOST=//p' .env

# Remove trailing whitespace
sed -i 's/[[:space:]]*$//' file.txt
```

1.4 find - File Search

```
# Tìm theo name/type
find /var/log -name "*.log" -type f
find /opt -name "*.conf" -type f -not -path "*/\.git/*"
find . -name "*.py" -newer requirements.txt    # modified sau requirements.txt

# Tìm theo time
find /tmp -mtime +7 -type f                    # modified >7 days ago
find /var/log -mmin -60 -type f                # modified trong 60 phút
find . -newer /tmp/marker -type f              # newer than reference file

# Tìm theo size
find / -size +100M -type f 2>/dev/null
find . -size 0 -type f                         # empty files
find /var/log -size +500M                      # large log files

# Tìm theo permission
find / -perm -4000 -type f 2>/dev/null         # SUID files
find /opt -perm /o+w -type f                   # world-writable files

# Execute actions
find /tmp -mtime +7 -type f -delete            # delete old files
find . -name "*.log" -exec gzip {} \;          # compress logs
find . -name "*.log" -exec ls -lh {} +         # batch với +

# Xargs combination
find . -name "*.py" | xargs grep -l "TODO"
find /var/log -name "*.log" -mtime +30 | xargs -I{} mv {} /archive/
```

1.5 xargs - Build Command Lines

```
# Cơ bản
echo "file1 file2 file3" | xargs rm
ls *.log | xargs -I{} gzip {}          # -I{} placeholder
```

```
# Parallel execution
find . -name "*.jpg" | xargs -P 4 -I{} convert {} -resize 800x600 {}

# Null-delimited (an toàn với filenames có spaces)
find . -name "*.log" -print0 | xargs -0 grep "ERROR"

# Limit args per command
echo {1..100} | xargs -n 10 echo      # 10 args per command

# Interactive confirmation
ls *.bak | xargs -I{} -p rm {}        # -p: prompt before each command
```

1.6 cut, sort, uniq, tr, tee

```
# cut - extract columns
cut -d: -f1,3 /etc/passwd             # delimiter :, fields 1 và 3
cut -c1-10 file.txt                   # first 10 characters
cut -d',' -f2- data.csv               # từ field 2 đến end

# sort
sort -k2 -n file.txt                  # sort by field 2, numeric
sort -t: -k3 -n /etc/passwd           # sort by UID
sort -u file.txt                      # unique + sort
sort -rn numbers.txt                  # reverse numeric sort
sort -h file.txt                      # human-readable (1K, 2M...)

# uniq (cần sort trước)
sort access.log | uniq -c | sort -rn | head -20   # count unique lines
sort ips.txt | uniq -d                            # show duplicates only
sort ips.txt | uniq -u                            # show unique only

# tr - translate/delete characters
echo "Hello World" | tr '[:upper:]' '[:lower:]'
cat file.txt | tr -d '\r'             # remove Windows carriage returns
echo "a:b:c" | tr ':' '\n'           # replace : with newline
cat file.txt | tr -s ' '              # squeeze multiple spaces

# tee - split output
command | tee output.log              # stdout + file
command | tee -a output.log           # append
command | tee /dev/stderr | next_cmd  # stderr + pipe
```

---

2. Shell Scripting Advanced

2.1 Parameter Expansion

```
var="Hello World"
echo ${#var}              # 11 (length)
echo ${var:6}             # World (substring from pos 6)
echo ${var:0:5}           # Hello (substring 0-5)
echo ${var// /_}          # Hello_World (replace all)
echo ${var/ /_}           # Hello_World (replace first)
echo ${var,,}             # hello world (lowercase)
echo ${var^^}             # HELLO WORLD (uppercase)

# Default values
echo ${VAR:-"default"}    # use default if unset/empty
echo ${VAR:="default"}    # set and use default if unset/empty
echo ${VAR:?"error msg"}  # error if unset/empty
echo ${VAR:+"alt"}        # use alt if set, else empty

# Path manipulation
path="/opt/app/conf/config.yaml"
echo ${path##*/}          # config.yaml (filename)
echo ${path%/*}           # /opt/app/conf (dirname)
echo ${path%%.*}          # /opt/app/conf/config (remove longest suffix)
echo ${path#*/}           # opt/app/conf/config.yaml (remove first /)
```

2.2 Arrays và Associative Arrays

```
# Indexed arrays
fruits=("apple" "banana" "cherry")
fruits+=("mango")                    # append
echo "${fruits[0]}"                  # apple
echo "${fruits[@]}"                  # all elements
echo "${#fruits[@]}"                 # length: 4
echo "${fruits[@]:1:2}"             # slice: banana cherry
unset fruits[1]                      # remove element

# Loop patterns
for item in "${fruits[@]}"; do
    echo "Processing: $item"
done

for i in "${!fruits[@]}"; do         # with index
    echo "$i: ${fruits[$i]}"
done

# Associative arrays (Bash 4+)
```

```
declare -A config
config[host]="db.example.com"
config[port]="5432"
config[name]="production"

echo "${config[host]}"
echo "${!config[@]}"                  # all keys
echo "${config[@]}"                   # all values

# Iterate
for key in "${!config[@]}"; do
    printf "%s=%s\n" "$key" "${config[$key]}"
done
```

2.3 Process Substitution

```
# <(command) - treat command output as file
diff <(ssh server1 cat /etc/hosts) <(ssh server2 cat /etc/hosts)
comm -23 <(sort file1.txt) <(sort file2.txt)  # lines in file1 not in file2

# >(command) - feed output of command into another command
tee >(gzip > backup.gz) >(wc -l > count.txt) < input.txt

# Practical: compare directory contents
diff <(ls /opt/app/v1) <(ls /opt/app/v2)
```

2.4 Here Documents và Here Strings

```
# Heredoc
cat << 'EOF' > /etc/nginx/sites-enabled/app.conf
server {
    listen 80;
    server_name example.com;
    location / {
        proxy_pass http://localhost:8080;
    }
}
EOF

# Heredoc với variable expansion (no quotes around EOF)
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
```

```
  APP_VERSION: "${APP_VERSION}"
EOF

# Here string
grep "pattern" <<< "$variable"
base64 --decode <<< "SGVsbG8gV29ybGQ="
```

---

3. Text Processing Pipelines Thực Tế

```
# Top 10 IP addresses từ nginx access log
awk '{print $1}' /var/log/nginx/access.log \
    | sort | uniq -c \
    | sort -rn \
    | head -10 \
    | awk '{printf "%-15s %s\n", $2, $1}'

# HTTP status code distribution
awk '{print $9}' /var/log/nginx/access.log \
    | sort | uniq -c \
    | sort -rn \
    | awk '{printf "%s: %d (%.1f%%)\n", $2, $1, $1/NR*100}'

# Find large files và format output
find /var -size +50M -type f 2>/dev/null \
    | xargs ls -lh \
    | awk '{print $5, $9}' \
    | sort -h -r \
    | head -20

# Processes consuming most memory
ps aux \
    | awk 'NR>1 {print $4, $11}' \
    | sort -rn \
    | head -10 \
    | column -t

# Extract unique error messages từ log
grep "ERROR" /var/log/app/*.log \
    | sed 's/.*ERROR: //' \
    | sort | uniq -c \
    | sort -rn \
    | head -20

# Monitor failed login attempts
journalctl -u sshd --since "24 hours ago" \
```

```
    | grep "Failed password" \
    | awk '{print $11}' \
    | sort | uniq -c \
    | sort -rn \
    | awk '$1 > 10 {print "ALERT:", $1, "attempts from", $2}'
```

---

4. Job Control

4.1 Background Jobs

```
# Run in background
command &                           # background, stdin disconnected
nohup command &                     # immune to SIGHUP, output to nohup.out
nohup command > output.log 2>&1 &   # custom output file

# Job control
jobs                                # list background jobs
fg %1                               # bring job 1 to foreground
bg %1                               # continue job 1 in background
kill %1                             # kill job 1 (by job number)
disown %1                           # detach từ shell (survive logout)

# Ctrl+Z: suspend foreground job
# Ctrl+C: interrupt/kill foreground job
```

4.2 tmux - Terminal Multiplexer

```
# Sessions
tmux new -s deploy                  # new session named "deploy"
tmux attach -t deploy               # attach to session
tmux ls                             # list sessions
tmux kill-session -t old            # kill session

# Key bindings (prefix: Ctrl+b)
# Ctrl+b c    - new window
# Ctrl+b n/p  - next/previous window
# Ctrl+b %    - split vertical
# Ctrl+b "    - split horizontal
# Ctrl+b d    - detach session
# Ctrl+b [    - scroll mode (q to exit)
# Ctrl+b z    - zoom pane

# Useful config (~/.tmux.conf)
```bash
```
```

```
# ~/.tmux.conf
set -g mouse on                       # mouse support
set -g history-limit 50000
set -g default-terminal "screen-256color"
# Ctrl+b r: reload config
bind r source-file ~/.tmux.conf \; display "Config reloaded"
# Vim-like pane navigation
bind h select-pane -L
bind j select-pane -D
bind k select-pane -U
bind l select-pane -R
```

4.3 screen

```
screen -S session_name              # new named session
screen -r session_name              # reattach
screen -ls                          # list sessions
# Ctrl+a d: detach
# Ctrl+a c: new window
# Ctrl+a n: next window
```

---

5. File Permissions

5.1 chmod, chown

```
# Symbolic mode
chmod u+x script.sh                 # add execute for user
chmod g-w file.txt                  # remove write for group
chmod o=r file.txt                  # set others to read only
chmod a+r file.txt                  # all: add read
chmod ug+rw,o-rwx secret.conf       # complex

# Octal mode: r=4, w=2, x=1
chmod 755 /opt/app/bin/server       # rwxr-xr-x
chmod 644 /etc/app/config.yaml      # rw-r--r--
chmod 600 ~/.ssh/id_rsa             # rw------- (private key!)
chmod 700 ~/.ssh                    # rwx------ (ssh dir)

# Recursive
chmod -R 755 /opt/app
find /opt/app -type d -exec chmod 755 {} \;   # only directories
find /opt/app -type f -exec chmod 644 {} \;   # only files

# chown
```

```
chown app:app /opt/app                # user:group
chown -R app:app /opt/app             # recursive
chown root:docker /var/run/docker.sock
```

5.2 Special Permissions

```
# SUID (4): run với owner's UID
chmod u+s /usr/bin/passwd             # passwd needs root access
chmod 4755 /opt/special               # octal: 4xxx

# SGID (2): run với group's GID / new files inherit group
chmod g+s /shared/project             # collaborative directory
chmod 2775 /shared/project

# Sticky bit (1): only owner can delete in directory
chmod +t /tmp                         # only file owner can delete
chmod 1777 /tmp                       # rwxrwxrwt

# Find SUID/SGID files (security audit)
find / -perm -4000 -type f 2>/dev/null    # SUID
find / -perm -2000 -type f 2>/dev/null    # SGID
```

5.3 ACL - Access Control Lists

```
# View ACL
getfacl /opt/app/config.yaml

# Set ACL
setfacl -m u:deploy:r-- /opt/app/config.yaml    # user deploy: read only
setfacl -m g:devteam:rw- /shared/file.txt       # group devteam: rw
setfacl -m o::--- /sensitive/data               # others: no access
setfacl -R -m u:ci:rx /opt/app                  # recursive

# Default ACL (inherited by new files)
setfacl -d -m g:devteam:rw /shared/project

# Remove ACL
setfacl -x u:olduser /opt/app/config.yaml       # remove specific
setfacl -b /opt/app/config.yaml                 # remove all ACL
```

---

6. Package Management

```
# Debian/Ubuntu (apt)
apt update && apt upgrade -y
apt install -y nginx curl jq
apt remove --purge nginx             # remove + config files
apt autoremove                       # remove unused dependencies
apt list --installed | grep nginx    # check installed
apt-cache show nginx                 # package info
apt-cache policy nginx               # available versions

# RHEL/CentOS (yum/dnf)
dnf update -y
dnf install -y nginx curl jq
dnf remove nginx
dnf list installed | grep nginx
dnf info nginx
dnf history                          # installation history

# Package pinning (apt)
echo "nginx hold" | dpkg --set-selections
apt-mark hold nginx                  # prevent upgrade

# Snap
snap install kubectl --classic
snap list
snap refresh kubectl                 # update
```

---

7. System Administration Commands

```
# Disk management
lsblk                                # block devices tree
blkid                                # UUIDs và filesystems
fdisk -l                             # partition table
df -h --output=source,size,used,avail,pcent,target   # disk usage
du -sh /var/log/*                    # directory sizes
du -sh * | sort -h | tail -20        # largest items

# Network
ip addr show                         # network interfaces
ip route show                        # routing table
ss -tlnp                             # listening TCP ports (modern netstat)
ss -s                                # socket summary
netstat -tlnp                        # legacy alternative
nmap -sV localhost                   # port scan + service detection
curl -v -o /dev/null http://example.com  # debug HTTP
traceroute example.com
```

```
mtr --report example.com             # better traceroute

# User management
useradd -m -s /bin/bash -G sudo deploy    # create user
usermod -aG docker $USER             # add to group
passwd deploy                        # set password
id deploy                            # user/group info
last                                 # login history
lastb                                # failed logins
who                                  # currently logged in
w                                    # detailed who + what

# Hardware info
lscpu                                # CPU architecture
lsmem                                # memory ranges
lspci                                # PCI devices
lsusb                                # USB devices
dmidecode -t memory                  # RAM details (requires root)
```

---

8. Pipeline Phức Tạp Thực Tế

```
# Real-time error monitoring với alerting
tail -f /var/log/app/error.log \
    | grep --line-buffered "CRITICAL" \
    | while IFS= read -r line; do
        echo "$line"
        curl -s -X POST "$SLACK_WEBHOOK" \
            -H 'Content-type: application/json' \
            --data "{\"text\":\"CRITICAL: $line\"}"
    done

# Detect config drift giữa servers
for host in web-01 web-02 web-03; do
    echo "=== $host ==="
    ssh "$host" "md5sum /etc/nginx/nginx.conf"
done | awk '/==/{server=$2} /[a-f0-9]{32}/{print $1, server}' \
    | sort | uniq -c | awk '$1 > 1 {print "MISMATCH:", $0}'

# Phân tích deploy frequency từ git log
git log --format="%ad" --date=format:"%Y-%m-%d" v1.0..HEAD \
    | sort | uniq -c \
    | awk '{printf "%s: %s\n", $2, substr("####################", 1, $1)}'

# Find và kill zombie processes
ps -eo pid,ppid,stat,cmd | awk '$3 ~ /Z/ {print $1, $2, $4}' \
```

```
    | while read pid ppid cmd; do
        echo "Zombie: PID=$pid (parent=$ppid) $cmd"
        kill -SIGCHLD "$ppid" 2>/dev/null
    done
```

Chapter 4: Git Advanced cho DevOps

Git không chỉ là version control - đối với DevOps engineer, Git là nền tảng của CI/CD pipeline, release management, và team collaboration. Chapter này tập trung vào những gì thực sự quan trọng trong môi trường production.

---

1. Git Internals

1.1 Object Model

Git lưu trữ data như một content-addressable filesystem. Mọi thứ đều là **object**, identified bởi SHA-1 hash.

```
Blob      → file content
Tree      → directory listing (blob + tree references)
Commit    → snapshot (tree + metadata + parent commits)
Tag       → named pointer to a commit
```

```
# Xem raw objects
git cat-file -t HEAD              # object type (commit)
git cat-file -p HEAD              # content của commit
git cat-file -p HEAD^{tree}       # tree object
git cat-file -p HEAD:src/main.go  # blob content

# Object database
ls .git/objects/                  # packed + loose objects
git count-objects -vH             # object count + size
git gc --aggressive               # garbage collect, repack

# Xem full history của file kể cả sau rename
git log --follow --all -- path/to/file
```

1.2 Refs và Reflog

```
# Refs: pointers to commits
cat .git/HEAD                     # current branch pointer
cat .git/refs/heads/main          # branch SHA
cat .git/refs/remotes/origin/main # remote tracking branch

# Reflog: lịch sử thay đổi của HEAD (safety net)
```

```
git reflog                        # HEAD movement history
git reflog show main              # history của branch main
git reflog expire --expire=90.days refs/heads/main  # cleanup

# Recovery: khôi phục commit đã "mất"
git reflog | head -20             # tìm SHA của commit muốn khôi phục
git checkout -b recovery-branch SHA  # tạo branch từ lost commit
```

1.3 Packfiles

```
# Git compact objects vào packfiles để tiết kiệm space
ls .git/objects/pack/             # .pack + .idx files
git verify-pack -v .git/objects/pack/pack-*.idx | sort -k3 -rn | head -10
# → tìm largest objects trong repo

# Tìm large files trong history (ví dụ ai đó commit binary)
git rev-list --all --objects \
    | git cat-file --batch-check='%(objecttype) %(objectname) %(objectsize) %(rest)' \
    | awk '/^blob/ {print substr($3,1), $4}' \
    | sort -rn | head -20
```

---

2. Branching Strategies

2.1 Git Flow

Phù hợp cho: Versioned software, scheduled releases.

```
main         → production code, always stable
develop      → integration branch
feature/*    → new features (from develop)
release/*    → release preparation (from develop)
hotfix/*     → urgent production fixes (from main)
```

```
# Feature branch workflow
git checkout develop
git checkout -b feature/user-authentication
# ... work ...
git rebase develop                    # sync với develop
git checkout develop
git merge --no-ff feature/user-authentication  # preserve merge commit
git branch -d feature/user-authentication

# Release
git checkout -b release/1.2.0 develop
# bump version, changelog
git checkout main && git merge --no-ff release/1.2.0
```

```
git tag -a v1.2.0 -m "Release 1.2.0"
git checkout develop && git merge --no-ff release/1.2.0
git branch -d release/1.2.0

# Hotfix
git checkout -b hotfix/fix-auth-bug main
# fix bug
git checkout main && git merge --no-ff hotfix/fix-auth-bug
git tag -a v1.2.1 -m "Hotfix: auth bug"
git checkout develop && git merge --no-ff hotfix/fix-auth-bug
```

2.2 GitHub Flow

Phù hợp cho: SaaS, continuous deployment.

```
main         → always deployable
feature/*    → short-lived branches, merge via PR
```

```
git checkout -b feature/add-caching
# code, commit
git push origin feature/add-caching
# Create PR → review → merge → auto-deploy
```

2.3 Trunk-Based Development

Phù hợp cho: High-frequency releases, experienced teams.

```
main/trunk   → everyone commits here, multiple times/day
release/*    → cut from trunk when needed
```

```
# Short-lived feature flags để merge incomplete features
git checkout -b feat/new-ui        # max 1-2 days
# merge to main quickly
# dùng feature flags trong code để control rollout
```

---

3. Git Hooks

Hooks chạy tự động ở các điểm trong Git workflow. Lưu trong `.git/hooks/` (local) hoặc quản lý với tools như **pre-commit**, **husky**.

3.1 Client-side Hooks

```
# .git/hooks/pre-commit (chạy trước khi commit)
#!/usr/bin/env bash
set -e
```

```
echo "Running pre-commit checks..."

# 1. Check for debug statements
if git diff --cached --name-only | xargs grep -l "console.log\|debugger\|pdb.set_trace" 2>/dev/null; then
    echo "ERROR: Debug statements found. Remove them before committing."
    exit 1
fi

# 2. Run linter on staged files
staged_py=$(git diff --cached --name-only --diff-filter=ACM | grep '\.py$' || true)
if [[ -n "$staged_py" ]]; then
    echo "$staged_py" | xargs ruff check || exit 1
fi

# 3. Check for secrets (basic pattern)
if git diff --cached | grep -qE '(password|secret|api_key)\s*=\s*["\x27][^"\x27]+["\x27]'; then
    echo "WARNING: Potential secret detected in diff"
    exit 1
fi
```

```
# .git/hooks/commit-msg (validate commit message)
#!/usr/bin/env bash
commit_msg=$(cat "$1")
pattern='^(feat|fix|docs|style|refactor|perf|test|chore|ci)(\(.+\))?: .{10,}'

if ! echo "$commit_msg" | grep -qE "$pattern"; then
    echo "ERROR: Commit message does not follow Conventional Commits format."
    echo "Pattern: type(scope): description (min 10 chars)"
    echo "Example: feat(auth): add JWT token refresh endpoint"
    exit 1
fi
```

```
# .git/hooks/pre-push (chạy trước khi push)
#!/usr/bin/env bash
set -e

protected_branch="main"
current_branch=$(git symbolic-ref HEAD | sed 's|refs/heads/||')

if [[ "$current_branch" == "$protected_branch" ]]; then
    echo "ERROR: Direct push to '$protected_branch' is not allowed."
    echo "Please create a pull request."
    exit 1
fi

echo "Running tests before push..."
npm test || exit 1
```

3.2 Quản lý Hooks với pre-commit Tool

```yaml
# .pre-commit-config.yaml
repos:
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v4.5.0
    hooks:
      - id: trailing-whitespace
      - id: end-of-file-fixer
      - id: check-yaml
      - id: check-json
      - id: check-merge-conflict
      - id: detect-private-key

  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.1.9
    hooks:
      - id: ruff
      - id: ruff-format

  - repo: https://github.com/Yelp/detect-secrets
    rev: v1.4.0
    hooks:
      - id: detect-secrets
```

```bash
pre-commit install          # install hooks
pre-commit run --all-files  # run on all files
pre-commit autoupdate       # update hook versions
```

---

4. Advanced Git Commands

4.1 Interactive Rebase

```bash
git rebase -i HEAD~5          # rebase last 5 commits

# Editor opens với:
# pick a1b2c3d feat: add login
# pick e4f5g6h fix: typo
# pick h7i8j9k feat: add logout

# Commands:
# pick   → keep commit as-is
# reword → keep but edit message
# edit   → stop for amending
# squash → meld into previous commit
# fixup  → squash, discard message
```

```
# drop   → remove commit

# Squash feature commits thành một commit sạch
git rebase -i main
# → squash/fixup tất cả thành 1 commit có message rõ ràng
```

4.2 Cherry-pick

```
# Copy specific commit sang branch khác
git cherry-pick abc1234                          # single commit
git cherry-pick abc1234 def5678                  # multiple commits
git cherry-pick v1.0..v1.5                      # range
git cherry-pick --no-commit abc1234             # apply changes không commit

# Ví dụ thực tế: backport hotfix sang release branch
git checkout release/1.5
git cherry-pick abc1234   # cherry-pick hotfix từ main
git push origin release/1.5
```

4.3 git bisect - Tìm Bug Commit

```
# Binary search để tìm commit gây ra bug
git bisect start
git bisect bad HEAD                    # current commit is bad
git bisect good v1.2.0                 # this version was good

# Git checkout commit ở giữa, test, rồi:
git bisect good                        # commit này OK
git bisect bad                         # commit này có bug
# Lặp lại → Git thu hẹp đến exact commit

# Automate với test script
git bisect run ./test-script.sh        # script: exit 0 = good, exit 1 = bad
git bisect reset                       # khi xong
```

4.4 Stash

```
git stash                              # stash uncommitted changes
git stash push -m "WIP: login feature" # với message
git stash push -u                      # include untracked files
git stash list                         # list stashes
git stash pop                          # apply + remove latest stash
git stash apply stash@{2}             # apply specific stash
git stash drop stash@{0}              # remove specific stash
git stash branch feature/login        # create branch từ stash
```

5. Merge Strategies và Conflict Resolution

5.1 Merge Options

```
# Regular merge (creates merge commit)
git merge feature/branch

# Fast-forward only (no merge commit if possible)
git merge --ff-only feature/branch

# No fast-forward (always create merge commit)
git merge --no-ff feature/branch        # recommended cho feature branches

# Squash merge (không preserve history)
git merge --squash feature/branch       # stage changes, bạn tự commit
git commit -m "feat: add user authentication"

# Rebase merge (linear history)
git checkout feature/branch
git rebase main
git checkout main
git merge --ff-only feature/branch
```

5.2 Conflict Resolution

```
# Khi conflict xảy ra
git status                              # xem conflicted files
git diff --diff-filter=U                # xem conflicts

# Conflict markers trong file:
# <<<<<<< HEAD
# current branch code
# =======
# incoming branch code
# >>>>>>> feature/branch

# Tools
git mergetool                           # launch configured merge tool
git mergetool --tool=vimdiff            # specific tool

# Resolve và continue
git add resolved-file.txt
git merge --continue
```

```
# Abort merge
git merge --abort

# Accept "ours" hoặc "theirs" cho toàn file
git checkout --ours   config.yaml      # keep current branch version
git checkout --theirs config.yaml      # use incoming version
git add config.yaml
```

---

6. Monorepo vs Polyrepo

6.1 Monorepo

```
# Câú trúc điển hình
mycompany/
├── services/
│   ├── api/
│   ├── worker/
│   └── frontend/
├── libs/
│   ├── shared-utils/
│   └── common-types/
├── infra/
│   └── terraform/
└── tools/

# Tools hỗ˜trợ Monorepo
# - Nx (JavaScript/TypeScript)
# - Turborepo (JavaScript)
# - Bazel (polyglot, Google)
# - Pants (Python)
# - Earthly (build automation)

# Sparse checkout - chỉ checkout phần cần
git sparse-checkout init --cone
git sparse-checkout set services/api libs/shared-utils
git sparse-checkout list
```

---

7. Git trong CI/CD

7.1 Shallow Clone - Tối ưu CI/CD

```
# Shallow clone: chỉ lấy N commits gần nhất
git clone --depth 1 https://github.com/org/repo.git
git clone --depth 1 --branch main https://github.com/org/repo.git

# Trong GitHub Actions
- uses: actions/checkout@v4
  with:
    fetch-depth: 1        # shallow clone

# Fetch thêm history khi cần (cho git bisect, versioning)
git fetch --unshallow
git fetch --depth 100 origin main     # fetch thêm 100 commits
```

7.2 Sparse Checkout trong CI

```
# Chỉ checkout phần cần build
git clone --filter=blob:none --sparse https://github.com/org/monorepo.git
cd monorepo
git sparse-checkout set services/api

# Dùng trong CI để giảm clone time cho monorepo lớn
```

7.3 Detecting Changed Files trong CI

```
# Tìm files thay đổi giữa commits
git diff --name-only HEAD~1 HEAD
git diff --name-only origin/main...HEAD      # changed vs main

# Determine what to build/deploy
changed_services=$(git diff --name-only origin/main...HEAD \
    | grep '^services/' \
    | cut -d'/' -f2 \
    | sort -u)

for service in $changed_services; do
    echo "Building $service..."
    make -C "services/$service" build test
done
```

---

8. .gitattributes và .gitignore

8.1 .gitattributes

```
# .gitattributes - control file handling

# Normalize line endings (important for cross-platform teams)
*           text=auto
*.sh        text eol=lf
*.bat       text eol=crlf
*.ps1       text eol=crlf

# Binary files - không diff/merge
*.png       binary
*.jpg       binary
*.pdf       binary
*.zip       binary

# Custom diff drivers
*.py        diff=python
*.md        diff=markdown

# Export ignore (không include trong git archive)
tests/      export-ignore
.github/    export-ignore
docs/       export-ignore

# Merge strategy cho generated files
package-lock.json   merge=ours
yarn.lock           merge=ours

# LFS tracking
*.psd               filter=lfs diff=lfs merge=lfs -text
*.mp4               filter=lfs diff=lfs merge=lfs -text
```

8.2 .gitignore Patterns

```
# .gitignore

# Environment và secrets - CRITICAL
.env
.env.*
!.env.example
*.key
*.pem
*.p12
credentials.json
```

```
secrets/

# Build artifacts
dist/
build/
*.egg-info/
__pycache__/
*.pyc
node_modules/
vendor/

# IDE
.idea/
.vscode/
*.swp
*.swo
.DS_Store

# Test coverage
.coverage
htmlcov/
.pytest_cache/
coverage.xml

# Logs
*.log
logs/

# OS
Thumbs.db
.DS_Store
```

---

9. Signing Commits với GPG

```
# Setup GPG signing
gpg --full-generate-key            # tạo GPG key
gpg --list-secret-keys --keyid-format=long

# Output:
# sec   rsa4096/3AA5C34371567BD2 2024-01-01 [SC]
#       ABCD1234...

# Configure git
git config --global user.signingkey 3AA5C34371567BD2
git config --global commit.gpgsign true    # sign all commits
```

```
git config --global tag.gpgsign true       # sign all tags

# Export public key cho GitHub/GitLab
gpg --armor --export 3AA5C34371567BD2

# Sign specific commit
git commit -S -m "feat: add secure feature"

# Verify signatures
git log --show-signature -1
git verify-commit HEAD
git verify-tag v1.0.0
```

---

10. Best Practices cho Team DevOps

10.1 Commit Message Convention

```
<type>(<scope>): <description>

[optional body]

[optional footer(s)]

Types: feat, fix, docs, style, refactor, perf, test, chore, ci, revert
```

```
# Ví dụ tốt
git commit -m "feat(auth): add OAuth2 integration with Google"
git commit -m "fix(deploy): handle timeout in health check retry logic"
git commit -m "ci: add Docker layer caching to GitHub Actions workflow"
git commit -m "chore(deps): upgrade boto3 from 1.28 to 1.34"

# Breaking changes
git commit -m "feat(api)!: change response format to JSON:API spec

BREAKING CHANGE: API response structure changed.
Clients must update to use data.attributes instead of direct fields."
```

10.2 Branch Protection Rules

```
# Enforce via GitHub/GitLab settings, hoặc document cho team:
# - main: require PR + 1 review + status checks pass
# - release/*: require PR + 2 reviews + all checks pass
# - Require signed commits
# - Require linear history (no merge commits)
# - Restrict force pushes
```

10.3 Useful Aliases

```
# ~/.gitconfig
[alias]
    # Short log
    lg = log --oneline --graph --decorate --all
    # Show changed files
    st = status -sb
    # Undo last commit (keep changes staged)
    undo = reset --soft HEAD~1
    # Show recent branches
    recent = branch --sort=-committerdate --format='%(committerdate:relative) %(refname:short)'
    # Find commit by message
    find = log --all --oneline --grep
    # Show contributors
    contrib = shortlog -sn --no-merges
    # Cleanup merged branches
    cleanup = "!git branch --merged | grep -v '\\*\\|main\\|develop' | xargs -n 1 git branch -d"
    # Stash pop with diff
    unstash = "!git stash show -p | git apply && git stash drop"
```

10.4 Repository Maintenance

```
# Cleanup stale remote tracking branches
git remote prune origin
git fetch --prune

# Cleanup local merged branches
git branch --merged main | grep -v "main\|develop" | xargs git branch -d

# Find large files và remove từ history (dùng BFG Repo Cleaner)
java -jar bfg.jar --strip-blobs-bigger-than 10M repo.git
git reflog expire --expire=now --all
git gc --prune=now --aggressive

# Verify repo integrity
git fsck --full

# Generate stats
git shortlog -sn --no-merges --since="6 months ago"
git log --oneline --since="1 week ago" | wc -l  # commit count last week
```

Chương 5: Networking cho DevOps

1. Mô hình OSI 7 Layers

Layer	Tên	Chức năng	Ví dụ giao thức
7	Application	Giao tiếp với ứng dụng	HTTP, FTP, SMTP, DNS
6	Presentation	Mã hóa, nén, định dạng dữ liệu	SSL/TLS, JPEG, ASCII
5	Session	Quản lý phiên kết nối	NetBIOS, RPC
4	Transport	Truyền tải end-to-end, port	TCP, UDP
3	Network	Định tuyến, địa chỉ IP	IP, ICMP, OSPF, BGP
2	Data Link	Frame, MAC address	Ethernet, Wi-Fi, ARP
1	Physical	Tín hiệu vật lý	Cáp đồng, quang, radio

Ghi nhớ thực tế: Khi debug, hỏi từng lớp: "Ping được không? (L3) → Port mở không? (L4) → Certificate hợp lệ không? (L6)"

---

2. TCP/IP Stack

3-Way Handshake

```
Client                    Server
  |  ---- SYN (seq=x) ---->  |   Bước 1: Client gửi SYN
  |  <-- SYN-ACK (seq=y) --  |   Bước 2: Server xác nhận + gửi SYN
  |  ---- ACK (seq=y+1) -->  |   Bước 3: Client xác nhận
  |  (Connection Established) |
```

TCP vs UDP

Đặc điểm	TCP	UDP
Kết nối	Connection-oriented	Connectionless
Độ tin cậy	Đảm bảo delivery	Best-effort
Thứ tự	Có	Không
Overhead	Cao	Thấp
Use case	HTTP, SSH, DB	DNS, Video stream, Gaming

TCP Congestion Control

- **Slow Start**: Bắt đầu với cwnd=1 MSS, tăng gấp đôi mỗi RTT
- **Congestion Avoidance**: Sau khi đạt ssthresh, tăng tuyến tính
- **Fast Retransmit**: 3 duplicate ACK → retransmit ngay
- **Fast Recovery**: Không về Slow Start sau Fast Retransmit

```
# Xem TCP stats
ss -s
netstat -s | grep -i retransmit
```

```
# Kiểm tra TCP buffer size
sysctl net.core.rmem_max
sysctl net.core.wmem_max

# Tăng TCP buffer cho high-throughput
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
```

---

3. DNS

Các loại DNS Records

Record	Mục đích	Ví dụ
A	IPv4 address	`api.example.com → 192.168.1.10`
AAAA	IPv6 address	`api.example.com → 2001:db8::1`
CNAME	Alias	`www.example.com → example.com`
MX	Mail server	`example.com → mail.example.com (priority 10)`
TXT	Metadata, SPF, DKIM	`"v=spf1 include:_spf.google.com ~all"`
SRV	Service locator	`_http._tcp.example.com`
NS	Name server	`example.com → ns1.provider.com`
PTR	Reverse DNS	`10.1.168.192.in-addr.arpa → api.example.com`

Recursive vs Iterative DNS

```
Recursive (thông thường):
Client → Resolver → Root NS → TLD NS → Authoritative NS → Client

Iterative (resolver tự xử lý từng bước):
Client → Root NS (trả về TLD NS address)
Client → TLD NS (trả về Auth NS address)
Client → Auth NS (trả về IP)
```

DNS Caching

```
# Kiểm tra TTL của record
dig example.com A +ttl

# Xem cache trên systemd-resolved
resolvectl statistics

# Flush DNS cache
systemd-resolve --flush-caches          # systemd
sudo killall -HUP mDNSResponder         # macOS
ipconfig /flushdns                      # Windows
```

Split-Horizon DNS

Cùng tên domain trả về IP khác nhau tùy vào nguồn query (internal/external):

```
# Bind9 - internal view
view "internal" {
    match-clients { 10.0.0.0/8; };
    zone "example.com" {
        file "internal.example.com.zone";  # Returns 10.0.1.5
    };
};

# External view
view "external" {
    zone "example.com" {
        file "external.example.com.zone";  # Returns 203.0.113.5
    };
};
```

---

4. HTTP/HTTPS

HTTP Methods

Method	Idempotent	Safe	Mục đích
GET	Yes	Yes	Lấy resource
POST	No	No	Tạo resource
PUT	Yes	No	Cập nhật toàn bộ
PATCH	No	No	Cập nhật một phần
DELETE	Yes	No	Xóa resource
HEAD	Yes	Yes	Lấy headers only
OPTIONS	Yes	Yes	Kiểm tra capabilities

Status Codes quan trọng

```
2xx Success:   200 OK, 201 Created, 204 No Content
3xx Redirect:  301 Moved Permanently, 302 Found, 304 Not Modified
4xx Client:    400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 429 Too Many Requests
5xx Server:    500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable, 504 Gateway Timeout
```

HTTP/2 vs HTTP/3

Tính năng	HTTP/1.1	HTTP/2	HTTP/3
Multiplexing	Không	Có (TCP)	Có (QUIC/UDP)
Header compression	Không	HPACK	QPACK
Server push	Không	Có	Có

Tính năng	HTTP/1.1	HTTP/2	HTTP/3
Head-of-line blocking	Có	Có (TCP level)	Không
Transport	TCP	TCP	UDP (QUIC)

---

5. SSH

Key-based Authentication

```
# Tạo SSH key pair
ssh-keygen -t ed25519 -C "devops@company.com" -f ~/.ssh/id_ed25519

# Copy public key lên server
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@server

# Hoặc thủ công
cat ~/.ssh/id_ed25519.pub >> ~/.ssh/authorized_keys

# Phân quyền đúng
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```

SSH Tunneling & Port Forwarding

```
# Local port forwarding: truy cập DB qua bastion
# Bind local port 5432 → forward qua bastion → tới DB server
ssh -L 5432:db.internal:5432 user@bastion.example.com

# Remote port forwarding: expose local service ra ngoài
ssh -R 8080:localhost:3000 user@public-server.com

# Dynamic SOCKS proxy
ssh -D 1080 user@server.com
# Dùng với curl: curl --socks5 localhost:1080 http://internal-service

# Jump host (ProxyJump)
ssh -J user@bastion.com user@internal-server.com
```

SSH Config file (~/.ssh/config)

```
Host bastion
    HostName bastion.example.com
    User ec2-user
    IdentityFile ~/.ssh/prod-key.pem
```

```
Host internal-*
    User ubuntu
    ProxyJump bastion
    IdentityFile ~/.ssh/prod-key.pem

Host *
    ServerAliveInterval 60
    ServerAliveCountMax 3
    AddKeysToAgent yes
```

ssh-agent

```
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_ed25519
ssh-add -l  # Liệt kê keys đã add

# Agent forwarding (truyền auth qua hop)
ssh -A user@bastion.com
```

---

6. Subnetting, CIDR, VLAN, VPN

CIDR Notation

```
192.168.1.0/24  → 254 hosts (256 - 2 reserved)
10.0.0.0/16     → 65534 hosts
172.16.0.0/12   → 1,048,574 hosts

# Tính subnet nhanh:
/24 = 256 IPs     /25 = 128 IPs
/26 = 64 IPs      /27 = 32 IPs
/28 = 16 IPs      /30 = 4 IPs (point-to-point links)
```

VLAN

```
# Tạo VLAN interface (Linux)
ip link add link eth0 name eth0.100 type vlan id 100
ip addr add 192.168.100.1/24 dev eth0.100
ip link set eth0.100 up
```

WireGuard VPN

```
# Tạo key pair
wg genkey | tee server_private.key | wg pubkey > server_public.key
```

```
# /etc/wireguard/wg0.conf (Server)
[Interface]
Address = 10.10.0.1/24
ListenPort = 51820
PrivateKey = <server_private_key>

[Peer]
PublicKey = <client_public_key>
AllowedIPs = 10.10.0.2/32

# Khởi động
wg-quick up wg0
systemctl enable wg-quick@wg0
```

---

7. Network Troubleshooting

Checklist debug theo layer

```
# L3 - Kết nối IP
ping -c 4 8.8.8.8
traceroute -n 8.8.8.8
mtr --report 8.8.8.8          # Kết hợp ping + traceroute

# L4 - Port connectivity
nc -zv server.com 443          # TCP port check
nc -zvu server.com 53          # UDP port check
ss -tlnp                       # Xem listening ports
ss -tnp state established      # Xem active connections

# DNS
dig @8.8.8.8 example.com A     # Query cụ thể DNS server
dig example.com +trace         # Trace full DNS resolution
nslookup -type=MX example.com

# Packet capture
tcpdump -i eth0 -n port 80 -w capture.pcap
tcpdump -i any host 10.0.1.5 and port 443

# Port scan
nmap -sV -p 22,80,443 server.com
nmap -sU -p 53 8.8.8.8         # UDP scan
```

Debug thực tế: “Website không load được”

```
# Bước 1: DNS resolve được không?
dig example.com A
# → Nếu NXDOMAIN: kiểm tra DNS config, propagation

# Bước 2: IP reachable không?
ping -c 3 $(dig +short example.com A)
# → Nếu timeout: kiểm tra firewall, routing

# Bước 3: Port 443 mở không?
nc -zv example.com 443
# → Nếu Connection refused: web server chưa chạy hoặc sai port

# Bước 4: SSL handshake OK không?
openssl s_client -connect example.com:443 -servername example.com
# → Kiểm tra certificate validity, chain

# Bước 5: HTTP response code
curl -I https://example.com
# → Xem response headers, status code

# Bước 6: Full trace
curl -v --trace-time https://example.com 2>&1 | head -50
```

---

8. Best Practices

- Dùng **IPv6 dual-stack** cho infrastructure mới
- Không hard-code IP, dùng DNS + service discovery
- Tách **management network** khỏi data plane
- Enable **TCP BBR** congestion control trên Linux kernels mới
- Monitor **DNS TTL** khi thay đổi infrastructure (hạ TTL trước)
- Dùng **MTR** thay cho traceroute trong production (ít noise hơn)
- Document **subnetting scheme** rõ ràng trước khi deploy

```
# Enable TCP BBR (Google's congestion control)
sysctl -w net.core.default_qdisc=fq
sysctl -w net.ipv4.tcp_congestion_control=bbr
```

Chương 6: Security cho DevOps

1. SSL/TLS

TLS Handshake (TLS 1.3)

```
Client                          Server
  | ---- ClientHello ----------> |   (cipher suites, TLS version, random)
  | <--- ServerHello ----------- |   (chosen cipher, certificate)
  | <--- Certificate ----------- |
  | <--- CertificateVerify ------ |
  | <--- Finished -------------- |
  | ---- Finished ------------> |
  | ===== Encrypted Data ======= |
```

TLS 1.3 chỉ còn **1 RTT** (thay vì 2 RTT ở TLS 1.2), cải thiện performance đáng kể.

Certificate Chain

```
Root CA (self-signed, offline storage)
  └── Intermediate CA (online, signs end-entity certs)
        └── End-entity Certificate (your domain)
```

```bash
# Kiểm tra certificate
openssl x509 -in cert.pem -text -noout

# Kiểm tra chain
openssl verify -CAfile ca-bundle.pem cert.pem

# Xem certificate từ server
openssl s_client -connect example.com:443 -showcerts

# Kiểm tra expiry
echo | openssl s_client -connect example.com:443 2>/dev/null \
  | openssl x509 -noout -dates
```

Let’s Encrypt với Certbot

```bash
# Cài certbot
apt install certbot python3-certbot-nginx

# Lấy certificate (standalone mode)
certbot certonly --standalone -d example.com -d www.example.com

# Auto-renew (crontab)
echo "0 0,12 * * * root certbot renew --quiet" >> /etc/cron.d/certbot

# Wildcard certificate (cần DNS challenge)
```

```
certbot certonly --manual --preferred-challenges dns \
  -d "*.example.com" -d "example.com"
```

---

2. PKI (Public Key Infrastructure)

Tạo Internal CA

```
# Tạo Root CA key và certificate
openssl genrsa -aes256 -out ca.key 4096
openssl req -new -x509 -days 3650 -key ca.key -out ca.crt \
  -subj "/C=VN/O=Company/CN=Internal CA"

# Tạo Intermediate CA
openssl genrsa -out intermediate.key 4096
openssl req -new -key intermediate.key -out intermediate.csr \
  -subj "/C=VN/O=Company/CN=Intermediate CA"
openssl x509 -req -days 1825 -in intermediate.csr -CA ca.crt \
  -CAkey ca.key -CAcreateserial -out intermediate.crt

# Ký certificate cho service
openssl genrsa -out service.key 2048
openssl req -new -key service.key -out service.csr \
  -subj "/CN=api.internal.example.com"
openssl x509 -req -days 365 -in service.csr -CA intermediate.crt \
  -CAkey intermediate.key -CAcreateserial -out service.crt
```

---

3. Secrets Management

HashiCorp Vault

```
# Khởi động Vault (dev mode - chỉ cho testing)
vault server -dev

# Câú hình
export VAULT_ADDR='http://127.0.0.1:8200'
export VAULT_TOKEN='root'

# Lưu và đọc secrets
vault kv put secret/myapp db_password="super_secret" api_key="abc123"
vault kv get secret/myapp
vault kv get -field=db_password secret/myapp

# Dynamic secrets cho PostgreSQL
```

```
vault secrets enable database
vault write database/config/mydb \
  plugin_name=postgresql-database-plugin \
  connection_url="postgresql://{{username}}:{{password}}@postgres:5432/mydb" \
  username="vault" password="vaultpassword"
```

SOPS (Secrets OPerationS)

```
# Mã hóa file secrets với AWS KMS
sops --kms arn:aws:kms:us-east-1:123456789:key/mrk-xxx -e secrets.yaml > secrets.enc.yaml

# Mã hóa với Age key
age-keygen -o key.txt
sops --age $(cat key.txt | grep "public key" | cut -d: -f2 | tr -d ' ') \
  -e secrets.yaml > secrets.enc.yaml

# Decrypt và dùng
sops -d secrets.enc.yaml | kubectl apply -f -
```

Sealed Secrets (Kubernetes)

```
# Cài sealed-secrets controller
kubectl apply -f https://github.com/bitnami-labs/sealed-secrets/releases/latest/download/controller.yaml

# Tạo sealed secret
kubectl create secret generic my-secret --from-literal=password=secret123 \
  --dry-run=client -o yaml | kubeseal -o yaml > sealed-secret.yaml

# Apply (chỉ controller mới decrypt được)
kubectl apply -f sealed-secret.yaml
```

---

4. SSH Hardening

```
# /etc/ssh/sshd_config - cấu hình bảo mật
PermitRootLogin no                  # Tắt root login
PasswordAuthentication no           # Chỉ dùng key
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
MaxAuthTries 3
LoginGraceTime 30
ClientAliveInterval 300
ClientAliveCountMax 2
AllowUsers devops deploy            # Chỉ cho phép user cụ thể
```

```
Protocol 2                          # Chỉ dùng SSH protocol 2

# Áp dụng
systemctl reload sshd
```

Fail2ban

```
# /etc/fail2ban/jail.local
[sshd]
enabled = true
port    = ssh
filter  = sshd
logpath = /var/log/auth.log
maxretry = 3
bantime = 3600      # 1 giờ
findtime = 600      # Trong 10 phút

# Kiểm tra
fail2ban-client status sshd
fail2ban-client set sshd unbanip 1.2.3.4  # Unban IP
```

---

5. Container Security

Image Scanning với Trivy

```
# Scan image
trivy image nginx:latest
trivy image --severity HIGH,CRITICAL nginx:latest

# Scan Dockerfile
trivy config Dockerfile

# Scan trong CI/CD
trivy image --exit-code 1 --severity CRITICAL myapp:latest
```

Rootless Container

```
# Dockerfile best practices
FROM node:18-alpine

# Tạo user không có root privilege
RUN addgroup -S appgroup && adduser -S appuser -G appgroup
```

```
WORKDIR /app
COPY --chown=appuser:appgroup . .
RUN npm ci --only=production

USER appuser  # QUAN TRỌNG: chạy với non-root user

EXPOSE 3000
CMD ["node", "server.js"]
```

Seccomp và AppArmor

```
# Chạy container với seccomp profile
docker run --security-opt seccomp=seccomp-profile.json myapp

# Chạy với AppArmor profile
docker run --security-opt apparmor=docker-default myapp

# Kubernetes - securityContext
# security-context.yaml
# spec:
#   securityContext:
#     runAsNonRoot: true
#     runAsUser: 1000
#     readOnlyRootFilesystem: true
#     allowPrivilegeEscalation: false
```

---

6. Network Security: Zero Trust & mTLS

mTLS (Mutual TLS)

Cả client và server đều xác thực lẫn nhau bằng certificate:

```
# Tạo client certificate
openssl genrsa -out client.key 2048
openssl req -new -key client.key -out client.csr -subj "/CN=service-a"
openssl x509 -req -days 365 -in client.csr -CA ca.crt \
  -CAkey ca.key -CAcreateserial -out client.crt

# Test mTLS với curl
curl --cert client.crt --key client.key --cacert ca.crt \
  https://api.internal.example.com
```

Zero Trust Checklist

- Không tin tưởng bất kỳ request nào dù từ internal network

- Mọi service-to-service communication đều dùng mTLS
- Phân quyền theo **least privilege**
- Log và monitor mọi access
- Dùng **service mesh** (Istio/Linkerd) để enforce zero trust

---

7. OWASP Top 10 (DevOps Awareness)

#	Tên	Biện pháp trong DevOps
1	Broken Access Control	IAM policy review, RBAC
2	Cryptographic Failures	Không dùng MD5/SHA1, dùng TLS 1.2+
3	Injection	Parameterized queries, input validation
4	Insecure Design	Threat modeling, security review
5	Security Misconfigura-tion	CIS benchmarks, hardening scripts
6	Vulnerable Components	Trivy, Snyk scanning trong CI/CD
7	Auth Failures	MFA, strong password policy
8	Software Integrity Failures	Sign Docker images, verify checksums
9	Security Logging Failures	Centralized logging, SIEM
10	SSRF	Network policies, egress filtering

---

8. Security Scanning trong CI/CD

```yaml
# GitHub Actions - security scanning pipeline
name: Security Scan
on: [push, pull_request]

jobs:
  scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
```

```yaml
      # SAST với SonarQube
      - name: SonarQube Scan
        uses: SonarSource/sonarqube-scan-action@master
        env:
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}

      # Container scanning với Trivy
      - name: Build image
        run: docker build -t myapp:${{ github.sha }} .

      - name: Trivy scan
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: myapp:${{ github.sha }}
          severity: 'HIGH,CRITICAL'
          exit-code: '1'

      # Dependency scanning với Snyk
      - name: Snyk scan
        uses: snyk/actions/node@master
        env:
          SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
```

---

9. CIS Benchmarks

```bash
# Chạy CIS benchmark audit với Lynis
lynis audit system

# Docker CIS benchmark
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock \
  docker/docker-bench-security

# Kubernetes CIS benchmark với kube-bench
kubectl apply -f https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job.yaml
kubectl logs job/kube-bench
```

---

10. Best Practices

Principle of Least Privilege

```
# IAM - chỉ cấp quyền cần thiết
# Xấu: AmazonS3FullAccess
# Tốt: chỉ cấp GetObject, PutObject cho bucket cụ thể
```

```
# Linux - chạy service với user riêng
useradd -r -s /bin/false appuser
chown appuser:appuser /opt/myapp
chmod 750 /opt/myapp
```

Defense in Depth

```
Layer 1: Perimeter firewall (block unknown traffic)
Layer 2: WAF (block OWASP attacks)
Layer 3: Network segmentation (VLAN/security groups)
Layer 4: Host-based firewall (iptables/nftables)
Layer 5: Application authentication/authorization
Layer 6: Data encryption at rest and in transit
Layer 7: Monitoring và alerting
```

Security Checklist trước Production

- ☐ Tắt tất cả unused ports và services
- ☐ Rotate tất cả default passwords
- ☐ Enable audit logging
- ☐ Cài đặt intrusion detection (AIDE, Falco)
- ☐ Test backup restoration
- ☐ Verify certificate expiry monitoring
- ☐ Review IAM permissions (least privilege)
- ☐ Enable MFA cho tất cả admin accounts

Chương 7: Web Servers

1. Nginx

Architecture: Event-Driven Model

Nginx dùng **event-driven, asynchronous, non-blocking** model:

```
Master Process
  ├── Worker Process 1  (xử lý nhiều connections đồng thời qua epoll/kqueue)
  ├── Worker Process 2
  ├── Worker Process N  (thường = số CPU cores)
  └── Cache Manager Process
```

Mỗi worker process xử lý hàng nghìn connections mà không cần tạo thread/process mới → tốn ít memory hơn Apache.

Cấu trúc Config

```
# /etc/nginx/nginx.conf
user www-data;
worker_processes auto;            # = sô´CPU cores
worker_rlimit_nofile 65535;       # Max file descriptors per worker
pid /run/nginx.pid;

events {
    worker_connections 4096;      # Max connections per worker
    use epoll;                    # Linux - dùng epoll (hiệu quả nhất)
    multi_accept on;              # Accept nhiêù connections cùng lúc
}

http {
    # MIME types, logging, gzip...
    include /etc/nginx/mime.types;
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

Server Blocks và Locations

```
# /etc/nginx/sites-available/example.com
server {
    listen 80;
    listen [::]:80;
    server_name example.com www.example.com;

    # Redirect HTTP → HTTPS
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    server_name example.com www.example.com;

    root /var/www/example.com;
    index index.html;

    # Location matching theo thứ tự ưu tiên:
    # = (exact) > ^~ (prefix, stop) > ~ (regex) > ~* (regex insensitive) > / (prefix)

    location = /health {
        return 200 "OK";
        add_header Content-Type text/plain;
    }
```

```
    location /api/ {
        proxy_pass http://backend_upstream;
    }

    location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
        expires 30d;
        add_header Cache-Control "public, immutable";
    }

    location / {
        try_files $uri $uri/ /index.html;  # SPA routing
    }
}
```

Static Files, Gzip, SSL

```
server {
    listen 443 ssl http2;
    server_name example.com;

    # SSL Configuration
    ssl_certificate     /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256;
    ssl_prefer_server_ciphers off;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 1d;
    ssl_stapling on;
    ssl_stapling_verify on;

    # HSTS
    add_header Strict-Transport-Security "max-age=63072000" always;

    # Gzip compression
    gzip on;
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 6;
    gzip_types text/plain text/css application/json application/javascript
               text/xml application/xml application/xml+rss text/javascript;

    # Static files
    location /static/ {
        root /var/www;
        expires 1y;
        add_header Cache-Control "public, immutable";
```

```
        access_log off;
    }
}
```

Rate Limiting và Access Control

```
http {
    # Định nghĩa zone (key=IP, size=10m, rate=10 req/s)
    limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;
    limit_req_zone $binary_remote_addr zone=login:10m rate=5r/m;

    # Giới hạn connections đồng thời
    limit_conn_zone $binary_remote_addr zone=perip:10m;

    server {
        location /api/ {
            limit_req zone=api burst=20 nodelay;
            limit_conn perip 10;
        }

        location /login {
            limit_req zone=login burst=5;
        }

        # IP whitelist/blacklist
        location /admin/ {
            allow 10.0.0.0/8;
            allow 192.168.0.0/16;
            deny all;
        }
    }
}
```

---

2. Apache HTTP Server

MPM Models

MPM	Model	Use Case
prefork	Multi-process, 1 process/connection	PHP mod_php (non-thread-safe)
worker	Multi-threaded, threads/process	Balanced performance
event	Async + thread pool	High concurrency (default mới)

```
# Kiểm tra MPM đang dùng
apache2ctl -V | grep MPM

# Cấu hình MPM event
# /etc/apache2/mods-enabled/mpm_event.conf
# StartServers              2
# MinSpareThreads           25
# MaxSpareThreads           75
# ThreadLimit               64
# ThreadsPerChild           25
# MaxRequestWorkers         150
# MaxConnectionsPerChild    0
```

VirtualHost và mod_rewrite

```
# /etc/apache2/sites-available/example.com.conf
<VirtualHost *:80>
    ServerName example.com
    ServerAlias www.example.com
    Redirect permanent / https://example.com/
</VirtualHost>

<VirtualHost *:443>
    ServerName example.com
    DocumentRoot /var/www/example.com
    DirectoryIndex index.html index.php

    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/example.pem
    SSLCertificateKeyFile /etc/ssl/private/example.key

    <Directory /var/www/example.com>
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    # mod_rewrite - URL rewriting
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule ^(.*)$ /index.php/$1 [L]

    ErrorLog ${APACHE_LOG_DIR}/example-error.log
    CustomLog ${APACHE_LOG_DIR}/example-access.log combined
</VirtualHost>
```

.htaccess

```
# .htaccess - dùng khi không access được server config
Options -Indexes
Options +FollowSymLinks

# Force HTTPS
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]

# Cache static assets
<FilesMatch "\.(css|js|png|jpg|gif|ico|woff2)$">
    Header set Cache-Control "max-age=31536000, public"
</FilesMatch>

# Security headers
Header always set X-Frame-Options "SAMEORIGIN"
Header always set X-Content-Type-Options "nosniff"
Header always set Referrer-Policy "strict-origin-when-cross-origin"
```

---

3. Tomcat (Java Servlet Container)

Kiến trúc Tomcat

```
Tomcat Server
  └── Service
        ├── Connector (HTTP/1.1 port 8080, AJP port 8009)
        └── Engine (Catalina)
              └── Host (Virtual Host)
                    └── Context (Web Application /myapp)
```

Connection Pool và JVM Tuning

```
<!-- server.xml - Connector configuration -->
<Connector port="8080" protocol="HTTP/1.1"
           maxThreads="200"
           minSpareThreads="25"
           acceptCount="100"
           connectionTimeout="20000"
           maxConnections="8192"
           compression="on"
           compressionMinSize="2048"
           URIEncoding="UTF-8" />
```

```
# JVM tuning (setenv.sh)
export JAVA_OPTS="-Xms512m -Xmx2g \
  -XX:+UseG1GC \
  -XX:MaxGCPauseMillis=200 \
  -XX:+HeapDumpOnOutOfMemoryError \
  -XX:HeapDumpPath=/var/log/tomcat/heapdump.hprof \
  -Djava.security.egd=file:/dev/./urandom"
```

---

4. Performance Tuning

Nginx - Tối ưu hóa

```
# nginx.conf
worker_processes auto;
worker_cpu_affinity auto;
worker_rlimit_nofile 65535;

events {
    worker_connections 4096;
    use epoll;
    multi_accept on;
}

http {
    # Sendfile - dùng kernel sendfile() thay vì read()+write()
    sendfile on;
    tcp_nopush on;          # Gộp header vào response đầu tiên
    tcp_nodelay on;         # Tắt Nagle's algorithm

    # Keepalive
    keepalive_timeout 65;
    keepalive_requests 1000;

    # Buffers
    client_body_buffer_size 128k;
    client_max_body_size 10m;
    client_header_buffer_size 1k;

    # File cache
    open_file_cache max=200000 inactive=20s;
    open_file_cache_valid 30s;
    open_file_cache_min_uses 2;
    open_file_cache_errors on;
}
```

---

5. Logging

Nginx Log Format

```
http {
    # Log format tùy chỉnh
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for" '
                    'rt=$request_time uct=$upstream_connect_time '
                    'uht=$upstream_header_time urt=$upstream_response_time';

    # JSON format (dễ parse với ELK stack)
    log_format json_combined escape=json
        '{'
        '"time":"$time_iso8601",'
        '"remote_addr":"$remote_addr",'
        '"method":"$request_method",'
        '"uri":"$request_uri",'
        '"status":$status,'
        '"bytes":$body_bytes_sent,'
        '"duration":$request_time,'
        '"upstream":"$upstream_addr"'
        '}';

    access_log /var/log/nginx/access.log json_combined;
    error_log  /var/log/nginx/error.log warn;
}
```

Log Rotation

```
# /etc/logrotate.d/nginx
/var/log/nginx/*.log {
    daily
    missingok
    rotate 14
    compress
    delaycompress
    notifempty
    sharedscripts
    postrotate
        nginx -s reopen
    endscript
}
```

---

6. So sánh Nginx vs Apache vs Tomcat

Tiêu chí	Nginx	Apache	Tomcat
Architecture	Event-driven	Process/Thread-based	Thread-based
Performance cao tải	Xuất sắc	Tốt	Tốt cho Java
Static files	Rất nhanh	Nhanh	Không phù hợp
Dynamic content	Qua FastCGI/proxy	Native mod_php	Native Java
Memory	Thấp	Cao hơn	Cao (JVM)
Config	Đơn giản, block	Phức tạp hơn	XML-heavy
.htaccess	Không hỗ trợ	Hỗ trợ	Không áp dụng
Use case	Reverse proxy, static	PHP apps, .htaccess	Java web apps

---

7. Production-Ready Config Example

Nginx - Microservices Production

```
upstream api_backend {
    least_conn;                  # Load balancing algorithm
    server 10.0.1.10:8080 weight=3;
    server 10.0.1.11:8080 weight=2;
    server 10.0.1.12:8080 backup;

    keepalive 32;                # Persistent connections tới backend
}

server {
    listen 443 ssl http2;
    server_name api.example.com;

    ssl_certificate /etc/ssl/certs/api.example.com.crt;
    ssl_certificate_key /etc/ssl/private/api.example.com.key;
    ssl_protocols TLSv1.2 TLSv1.3;

    # Security headers
    add_header X-Frame-Options "DENY" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
    add_header Content-Security-Policy "default-src 'self'" always;

    # Rate limiting
    limit_req zone=api burst=50 nodelay;
```

```
    location / {
        proxy_pass http://api_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";  # WebSocket support
        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;

        proxy_connect_timeout 5s;
        proxy_send_timeout 30s;
        proxy_read_timeout 30s;
        proxy_buffering off;       # Streaming responses
    }

    location /health {
        access_log off;
        return 200 "healthy\n";
    }
}
```

Kiểm tra và reload config

```
nginx -t                    # Kiểm tra syntax
nginx -s reload             # Reload config không downtime
systemctl status nginx

# Debug
nginx -T                    # Dump full config
tail -f /var/log/nginx/error.log
```

Chương 8: Reverse Proxy

1. Khái niệm

Forward Proxy vs Reverse Proxy

```
Forward Proxy (Client-side):
[Client] → [Forward Proxy] → [Internet/Server]
Mục đích: Ẩn danh client, bypass firewall, caching

Reverse Proxy (Server-side):
[Client] → [Reverse Proxy] → [Backend Servers]
Mục đích: Load balancing, SSL termination, caching, security
```

Reverse Proxy cung cấp: - Load balancing giữa nhiều backend instances - SSL termination (giải mã TLS một lần, backend dùng plain HTTP) - Health checking và failover tự động - Rate limiting, WAF - Request/response modification - Centralized logging

---

2. Nginx Reverse Proxy

Cấu hình cơ bản

```
# Upstream block - định nghĩa backend pool
upstream backend {
    least_conn;               # Thuật toán: least connections
    # Các thuật toán khác: round-robin (default), ip_hash, hash $request_uri

    server 10.0.1.10:8080 max_fails=3 fail_timeout=30s;
    server 10.0.1.11:8080 max_fails=3 fail_timeout=30s;
    server 10.0.1.12:8080 backup;        # Dùng khi primary down

    keepalive 32;             # Duy trì 32 persistent connections
}

server {
    listen 80;
    server_name app.example.com;

    location / {
        proxy_pass http://backend;

        # Headers quan trọng
        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;
        proxy_set_header X-Forwarded-Host  $host;
        proxy_set_header X-Forwarded-Port  $server_port;

        # Connection settings
        proxy_http_version 1.1;
        proxy_set_header Connection "";    # Enable keepalive tới upstream

        # Timeouts
        proxy_connect_timeout  5s;
        proxy_send_timeout     60s;
        proxy_read_timeout     60s;

        # Buffers
        proxy_buffer_size          4k;
```

```
        proxy_buffers              4 32k;
        proxy_busy_buffers_size    64k;
    }
}
```

Nginx - Multiple Upstreams / Path-based Routing

```
upstream api_service {
    server api-1:3000;
    server api-2:3000;
}

upstream auth_service {
    server auth-1:4000;
}

upstream static_service {
    server static:8080;
}

server {
    listen 443 ssl http2;
    server_name gateway.example.com;

    location /api/ {
        proxy_pass http://api_service/;    # Trailing slash strips /api prefix
    }

    location /auth/ {
        proxy_pass http://auth_service/;
    }

    location /static/ {
        proxy_pass http://static_service/;
        proxy_cache_valid 200 1d;
    }
}
```

---

3. HAProxy

Frontend/Backend Architecture

`Client → Frontend (ACL routing) → Backend (server pool)`

```
# /etc/haproxy/haproxy.cfg
global
    log /dev/log local0
    maxconn 50000
    user haproxy
    group haproxy
    daemon

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

# Frontend - nhận traffic từ client
frontend http_in
    bind *:80
    bind *:443 ssl crt /etc/ssl/certs/example.pem
    http-request redirect scheme https unless { ssl_fc }

    # ACL rules - điều hướng theo domain/path
    acl is_api hdr(host) -i api.example.com
    acl is_app hdr(host) -i app.example.com
    acl is_static path_beg /static

    use_backend api_backend  if is_api
    use_backend static_cache if is_static
    default_backend app_backend

# Backend cho API
backend api_backend
    balance roundrobin
    option httpchk GET /health
    http-check expect status 200

    server api-1 10.0.1.10:3000 check inter 5s rise 2 fall 3
    server api-2 10.0.1.11:3000 check inter 5s rise 2 fall 3
    server api-3 10.0.1.12:3000 check inter 5s rise 2 fall 3 backup

# Backend cho App với sticky sessions
backend app_backend
    balance source              # IP hash - đảm bảo sticky sessions
    cookie SERVERID insert indirect nocache  # Cookie-based sticky sessions
```

```
    option httpchk GET /ping

    server app-1 10.0.2.10:8080 check cookie server1
    server app-2 10.0.2.11:8080 check cookie server2
```

HAProxy Stats Dashboard

```
# Thêm vào haproxy.cfg
frontend stats
    bind *:9000
    stats enable
    stats uri /stats
    stats refresh 10s
    stats auth admin:password
    stats hide-version
    stats show-legends
```

---

4. Traefik

Auto-discovery với Docker

```yaml
# docker-compose.yml
version: '3.8'

services:
  traefik:
    image: traefik:v3.0
    command:
      - "--api.dashboard=true"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
      - "--certificatesresolvers.letsencrypt.acme.httpchallenge=true"
      - "--certificatesresolvers.letsencrypt.acme.httpchallenge.entrypoint=web"
      - "--certificatesresolvers.letsencrypt.acme.email=admin@example.com"
      - "--certificatesresolvers.letsencrypt.acme.storage=/letsencrypt/acme.json"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - letsencrypt:/letsencrypt
    labels:
```

```
      - "traefik.enable=true"
      - "traefik.http.routers.dashboard.rule=Host(`traefik.example.com`)"
      - "traefik.http.routers.dashboard.tls.certresolver=letsencrypt"
      - "traefik.http.routers.dashboard.service=api@internal"

  myapp:
    image: myapp:latest
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.myapp.rule=Host(`app.example.com`)"
      - "traefik.http.routers.myapp.tls.certresolver=letsencrypt"
      - "traefik.http.services.myapp.loadbalancer.server.port=3000"
      # Rate limiting middleware
      - "traefik.http.middlewares.ratelimit.ratelimit.average=100"
      - "traefik.http.middlewares.ratelimit.ratelimit.burst=50"
      - "traefik.http.routers.myapp.middlewares=ratelimit"

volumes:
  letsencrypt:
```

Traefik với Kubernetes IngressRoute

```
# ingressroute.yaml
apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: myapp-ingress
spec:
  entryPoints:
    - websecure
  routes:
    - match: Host(`app.example.com`) && PathPrefix(`/api`)
      kind: Rule
      services:
        - name: api-service
          port: 3000
      middlewares:
        - name: rate-limit
    - match: Host(`app.example.com`)
      kind: Rule
      services:
        - name: frontend-service
          port: 80
  tls:
    certResolver: letsencrypt
```

5. SSL Termination Modes

Mode	Mô tả	Nginx config	Use case
SSL Termination	Proxy decrypt TLS, gửi plain HTTP tới backend	`proxy_pass http://backend`	Thông thường, đơn giản nhất
SSL Passthrough	Proxy chuyển TLS encrypted traffic thẳng	`stream { proxy_pass backend:443 }`	Backend cần giữ encryption
SSL Bridging	Proxy decrypt rồi re-encrypt khi gửi backend	`proxy_pass https://backend`	Compliance, full encryption

```
# SSL Termination (Layer 7)
server {
    listen 443 ssl;
    location / {
        proxy_pass http://backend;  # Plain HTTP to backend
    }
}

# SSL Passthrough (Layer 4 - stream module)
stream {
    upstream backend_ssl {
        server 10.0.1.10:443;
    }
    server {
        listen 443;
        proxy_pass backend_ssl;  # Pass raw TCP stream
    }
}
```

---

6. WebSocket Proxying

```
# Nginx WebSocket proxy
map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}

server {
    location /ws/ {
```

```
        proxy_pass http://websocket_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
        proxy_set_header Host $host;

        # WebSocket timeout - tăng lên để giữ long-lived connections
        proxy_read_timeout 3600s;
        proxy_send_timeout 3600s;
    }
}
```

---

7. gRPC Proxying

```
# Nginx gRPC proxy (HTTP/2 required)
upstream grpc_backend {
    server 10.0.1.10:50051;
    server 10.0.1.11:50051;
}

server {
    listen 443 ssl http2;
    server_name grpc.example.com;

    location / {
        grpc_pass grpc://grpc_backend;

        # gRPC-specific settings
        grpc_set_header X-Real-IP $remote_addr;
        grpc_read_timeout 600s;
        grpc_send_timeout 600s;

        # Error handling
        error_page 502 = /error502grpc;
    }

    location = /error502grpc {
        internal;
        default_type application/grpc;
        add_header grpc-status 14;
        add_header content-length 0;
        return 204;
    }
}
```

---

8. Headers Quan Trọng

```
X-Forwarded-For: <client-ip>, <proxy1-ip>, <proxy2-ip>
  → Chuỗi IP qua các proxies, IP đầu tiên là client gốc

X-Real-IP: <client-ip>
  → IP client gốc (chỉ đặt bởi proxy đầu tiên)

X-Forwarded-Proto: https
  → Protocol gốc (quan trọng để biết client dùng HTTP hay HTTPS)

X-Forwarded-Host: original-host.com
  → Host header gốc từ client

Host: backend-internal.example.com
  → Nginx tự đặt, override Host header tới backend
```

```
# Nginx - đặt headers đúng cách
proxy_set_header X-Real-IP         $remote_addr;
proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;  # Append IP
proxy_set_header X-Forwarded-Proto $scheme;
proxy_set_header Host              $host;

# Xóa header nhạy cảm từ client
proxy_hide_header X-Powered-By;
proxy_hide_header Server;
```

---

9. Rate Limiting và Circuit Breaking

Nginx Rate Limiting nâng cao

```
http {
    limit_req_zone $binary_remote_addr zone=per_ip:10m rate=100r/s;
    limit_req_zone $http_x_api_key zone=per_key:10m rate=1000r/s;

    server {
        location /api/ {
            # Per-IP: 100 req/s, burst 200, không delay burst
            limit_req zone=per_ip burst=200 nodelay;

            # Per API key: 1000 req/s
            limit_req zone=per_key burst=2000 nodelay;

            limit_req_status 429;  # Trả về 429 thay vì 503
```

```
        }
    }
}
```

Circuit Breaker với Nginx (passive health check)

```
upstream backend {
    server 10.0.1.10:8080 max_fails=5 fail_timeout=30s;
    server 10.0.1.11:8080 max_fails=5 fail_timeout=30s;

    # Nginx Plus: active health check
    # health_check interval=5s fails=3 passes=2;
}

# Open source alternative: ngx_http_upstream_module
server {
    location / {
        proxy_pass http://backend;
        proxy_next_upstream error timeout http_500 http_502 http_503;
        proxy_next_upstream_tries 3;
        proxy_next_upstream_timeout 10s;
    }
}
```

---

10. High Availability Setup

Keepalived + Nginx (Active-Passive)

```
# /etc/keepalived/keepalived.conf (Master node)
vrrp_script check_nginx {
    script "pidof nginx"
    interval 2
    weight 2
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 101            # Master có priority cao hơn

    virtual_ipaddress {
        192.168.1.100/24    # Virtual IP (VIP) - floating IP
    }
```

```
    track_script {
        check_nginx
    }
}

# Backup node: state BACKUP, priority 100
```

Microservices Production Example

```
# Gateway cho microservices architecture

upstream user_service    { server user-svc:8001;  server user-svc-2:8001; }
upstream order_service   { server order-svc:8002; server order-svc-2:8002; }
upstream payment_service { server payment-svc:8003; }
upstream notification_service { server notif-svc:8004; }

server {
    listen 443 ssl http2;
    server_name api.example.com;

    # Global rate limit
    limit_req zone=api burst=100 nodelay;

    # Service routing
    location /v1/users/       { proxy_pass http://user_service/; }
    location /v1/orders/      { proxy_pass http://order_service/; }
    location /v1/payments/    {
        proxy_pass http://payment_service/;
        # Payment service cần timeout dài hơn
        proxy_read_timeout 120s;
    }
    location /v1/notifications/ { proxy_pass http://notification_service/; }

    # Health endpoint không log để giảm noise
    location /health {
        access_log off;
        return 200 '{"status":"ok"}';
        add_header Content-Type application/json;
    }

    # Centralized error handling
    error_page 502 503 504 /50x.json;
    location = /50x.json {
        internal;
        default_type application/json;
        return 503 '{"error":"Service temporarily unavailable"}';
```

```
    }
}
```

---

11. So sánh Nginx vs HAProxy vs Traefik

Tiêu chí	Nginx	HAProxy	Traefik
Performance	Rất cao	Cao nhất	Tốt
Layer 4/7	Cả hai	Cả hai	Layer 7 chủ yếu
Auto-discovery	Không	Không	Có (Docker/K8s)
Let’s Encrypt	Cần certbot	Không native	Tích hợp sẵn
Config reload	Graceful	Hitless	Dynamic
Learning curve	Trung bình	Cao	Thấp
Best for	Web serving + proxy	High-perf TCP LB	Cloud-native/containers

Chương 9: Caching Strategies

Caching là kỹ thuật lưu trữ tạm thời dữ liệu đã xử lý để tái sử dụng, giảm latency và tải cho hệ thống backend.

---

1. Các tầng Caching

```
User Browser
     |
     ▼
  CDN Edge (CloudFlare, CloudFront)
     |
     ▼
  Reverse Proxy Cache (Varnish, Nginx)
     |
     ▼
  Application Cache (Redis, Memcached)
     |
     ▼
  Database Query Cache (MySQL Query Cache, pg_bouncer)
     |
     ▼
  Storage (SSD, HDD)
```

Tầng	Latency	Dung lượng	Use case
Browser	0ms	MB	Static assets, API responses
CDN	1-10ms	TB	Static files, media
Reverse Proxy	1-5ms	GB	HTML pages, API responses
Application	0.1-1ms	GB	Session, computed data
Database	1-10ms	GB	Query results

---

2. Redis

2.1 Data Types

```
# String - counter, session token
SET user:1:name "Nguyen Van A"
GET user:1:name
INCR page:views
SETEX session:abc123 3600 "user_data"   # TTL 1 giờ

# Hash - object fields
HSET user:1 name "Nguyen Van A" email "a@example.com" age 30
HGET user:1 name
HGETALL user:1
HINCRBY user:1 age 1

# List - queue, recent items
LPUSH notifications:user1 "New message"
RPUSH queue:emails "job1" "job2"
LRANGE notifications:user1 0 9    # lấy 10 item mới nhất
BLPOP queue:emails 30             # blocking pop, timeout 30s

# Set - unique items, tags
SADD user:1:tags "developer" "golang" "devops"
SMEMBERS user:1:tags
SINTER user:1:tags user:2:tags    # intersection
SUNION user:1:tags user:2:tags

# Sorted Set - leaderboard, rate limiting
ZADD leaderboard 1500 "player1" 2000 "player2"
ZRANGE leaderboard 0 -1 WITHSCORES
ZREVRANK leaderboard "player1"

# Bitmap - feature flags, analytics
SETBIT user:active:2024-01-15 userId 1
BITCOUNT user:active:2024-01-15

# HyperLogLog - unique visitors (approximate)
```

```
PFADD visitors "user1" "user2" "user3"
PFCOUNT visitors
```

2.2 Persistence

RDB (Redis Database Backup) - snapshot:

```
# redis.conf
save 900 1      # 900s nếu có >= 1 thay đổi
save 300 10     # 300s nếu có >= 10 thay đổi
save 60 10000   # 60s nếu có >= 10000 thay đổi
dbfilename dump.rdb
dir /var/lib/redis

# Manual snapshot
redis-cli BGSAVE
redis-cli LASTSAVE    # timestamp của snapshot cuối
```

AOF (Append-Only File) - log mọi write:

```
# redis.conf
appendonly yes
appendfilename "appendonly.aof"
appendfsync everysec    # everysec (balance), always (safe), no (fast)
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

Mode	Durability	Performance	File size
RDB only	Thấp	Cao	Nhỏ
AOF everysec	Trung bình	Trung bình	Lớn
AOF always	Cao	Thấp	Lớn
RDB + AOF	Cao	Trung bình	Trung bình

2.3 Replication

```
# redis.conf trên replica
replicaof 192.168.1.10 6379
replica-read-only yes

# Kiểm tra replication status
redis-cli INFO replication
```

2.4 Sentinel (High Availability)

```
# sentinel.conf
sentinel monitor mymaster 192.168.1.10 6379 2    # quorum = 2
sentinel down-after-milliseconds mymaster 5000
sentinel failover-timeout mymaster 60000
sentinel parallel-syncs mymaster 1

# Khởi động
redis-sentinel /etc/redis/sentinel.conf

# Kiểm tra
redis-cli -p 26379 SENTINEL masters
redis-cli -p 26379 SENTINEL get-master-addr-by-name mymaster
```

2.5 Cluster

```
# Tạo cluster 6 nodes (3 master + 3 replica)
redis-cli --cluster create \
  192.168.1.10:7000 192.168.1.10:7001 192.168.1.10:7002 \
  192.168.1.11:7000 192.168.1.11:7001 192.168.1.11:7002 \
  --cluster-replicas 1

# Kiểm tra cluster
redis-cli -c -p 7000 CLUSTER INFO
redis-cli -c -p 7000 CLUSTER NODES

# Hash slots: 16384 slots chia đều cho 3 master
# slot = CRC16(key) % 16384
```

2.6 Pub/Sub

```
# Subscriber
redis-cli SUBSCRIBE channel:notifications

# Publisher
redis-cli PUBLISH channel:notifications '{"type":"order","id":123}'

# Pattern subscribe
redis-cli PSUBSCRIBE "user:*:events"
```

2.7 Lua Scripting

```
# Atomic check-and-set
redis-cli EVAL "
  local val = redis.call('GET', KEYS[1])
  if val == ARGV[1] then
```

```
    redis.call('SET', KEYS[1], ARGV[2])
    return 1
  end
  return 0
" 1 mykey oldvalue newvalue

# Rate limiting script
redis-cli EVAL "
  local key = KEYS[1]
  local limit = tonumber(ARGV[1])
  local current = tonumber(redis.call('INCR', key))
  if current == 1 then
    redis.call('EXPIRE', key, 60)
  end
  if current > limit then
    return 0
  end
  return 1
" 1 ratelimit:user1 100
```

---

3. Memcached

3.1 Architecture

- Multi-threaded, không hỗ trợ persistence
- Slab allocation: chia memory thành slabs với size cố định
- Consistent hashing để phân phối key giữa các node

```
# Cài đặt và khởi động
memcached -d -m 1024 -u memcache -l 0.0.0.0 -p 11211

# Kết nối và test
telnet localhost 11211
set mykey 0 300 5    # flags=0, ttl=300s, bytes=5
hello
get mykey
```

3.2 So sánh Redis vs Memcached

Tiêu chí	Redis	Memcached
Data types	Phong phú	String only
Persistence	RDB + AOF	Không
Replication	Master-Replica	Không native
Cluster	Native	Client-side
Threading	Single-threaded*	Multi-threaded

Tiêu chí	Redis	Memcached
Lua	Có	Không
Use case	Complex caching	Simple caching

---

4. Varnish Cache

4.1 VCL (Varnish Configuration Language)

```
# /etc/varnish/default.vcl
vcl 4.1;

backend default {
    .host = "127.0.0.1";
    .port = "8080";
    .probe = {
        .url = "/health";
        .interval = 5s;
        .timeout = 1s;
        .window = 5;
        .threshold = 3;
    }
}

sub vcl_recv {
    # Không cache POST/PUT/DELETE
    if (req.method != "GET" && req.method != "HEAD") {
        return(pass);
    }

    # Không cache authenticated requests
    if (req.http.Authorization || req.http.Cookie ~ "session") {
        return(pass);
    }

    # Chuẩn hóa URL
    set req.url = regsuball(req.url, "\?.*$", "");
    return(hash);
}

sub vcl_backend_response {
    # Cache 1 giờ cho static files
    if (bereq.url ~ "\.(css|js|png|jpg|gif|ico)$") {
        set beresp.ttl = 1h;
        unset beresp.http.Set-Cookie;
    }
```

```
    # Grace mode: phục vụ stale content khi backend down
    set beresp.grace = 1h;
}

sub vcl_deliver {
    # Thêm header debug
    if (obj.hits > 0) {
        set resp.http.X-Cache = "HIT";
    } else {
        set resp.http.X-Cache = "MISS";
    }
}
```

4.2 Cache Invalidation

```
# Purge single URL
varnishadm "ban req.url == /api/products"

# Purge by tag (cần BAN lurker)
curl -X PURGE http://varnish/api/products/123

# Soft purge (grace mode)
varnishadm "ban req.url ~ ^/api/"
```

---

5. CDN Caching

5.1 CloudFlare

```
# Page Rules
*.example.com/static/*  →  Cache Level: Cache Everything, Edge TTL: 1 month
*.example.com/api/*     →  Cache Level: Bypass

# Cache-Control headers
Cache-Control: public, max-age=31536000, immutable    # static assets
Cache-Control: public, max-age=3600, s-maxage=86400   # pages
Cache-Control: private, no-cache                       # authenticated
```

5.2 AWS CloudFront

```
{
  "CacheBehaviors": [{
    "PathPattern": "/static/*",
    "TTL": { "DefaultTTL": 86400, "MaxTTL": 31536000 },
```

```
    "Compress": true,
    "ViewerProtocolPolicy": "redirect-to-https"
  }]
}
```

---

6. Cache Patterns

6.1 Cache-Aside (Lazy Loading)

```
def get_user(user_id):
    # 1. Kiểm tra cache
    cached = redis.get(f"user:{user_id}")
    if cached:
        return json.loads(cached)

    # 2. Cache miss - lấy từ DB
    user = db.query("SELECT * FROM users WHERE id = %s", user_id)

    # 3. Lưu vào cache
    redis.setex(f"user:{user_id}", 3600, json.dumps(user))
    return user
```

6.2 Write-Through

```
def update_user(user_id, data):
    # Cập nhật DB trước
    db.execute("UPDATE users SET ... WHERE id = %s", user_id)
    # Cập nhật cache ngay lập tức
    redis.setex(f"user:{user_id}", 3600, json.dumps(data))
```

6.3 Write-Behind (Write-Back)

```
# Ghi vào cache trước, async write vào DB sau
def update_user_async(user_id, data):
    redis.setex(f"user:{user_id}", 3600, json.dumps(data))
    redis.lpush("write_queue", json.dumps({"id": user_id, "data": data}))
    # Worker riêng sẽ flush queue vào DB
```

6.4 Read-Through

Cache tự động load từ DB khi miss (thường dùng với Redis modules hoặc ORMs).

---

7. Cache Invalidation Strategies

```
# TTL-based - đơn giản nhất
SETEX product:123 3600 "..."

# Event-based - invalidate khi có thay đổi
# Publish event khi update
PUBLISH product:updated '{"id": 123}'
# Subscriber xóa cache
DEL product:123

# Versioning - thêm version vào key
SET product:123:v42 "..."   # không cần xóa key cũ

# Cache tags - nhóm các key liên quan
SADD tag:category:electronics "product:123" "product:456"
# Khi category thay đổi, xóa tất cả tagged keys
```

---

8. Cache Stampede Prevention

```
# Probabilistic early expiration
import random
import time

def get_with_early_expiry(key, ttl, beta=1.0):
    value, expire_time = cache.get_with_expiry(key)
    if value is None:
        return recompute_and_cache(key, ttl)

    # Tính xác suất recompute sớm
    current_time = time.time()
    delta = ttl - (expire_time - current_time)
    if current_time - delta * beta * math.log(random.random()) >= expire_time:
        # Recompute trước khi hết hạn
        return recompute_and_cache(key, ttl)

    return value

# Mutex lock - chỉ một request recompute
import redis
import time

def get_or_set_with_lock(key, compute_func, ttl=3600):
    value = redis.get(key)
    if value:
```

```
        return value

    lock_key = f"lock:{key}"
    if redis.set(lock_key, 1, nx=True, ex=30):   # acquire lock
        try:
            value = compute_func()
            redis.setex(key, ttl, value)
        finally:
            redis.delete(lock_key)
    else:
        # Chờ lock release
        time.sleep(0.1)
        return redis.get(key)
    return value
```

---

9. Monitoring Cache

```
# Redis metrics
redis-cli INFO stats | grep -E "keyspace_hits|keyspace_misses|evicted_keys"
# keyspace_hits:1234567
# keyspace_misses:12345
# evicted_keys:0

# Tính hit ratio
# hit_ratio = hits / (hits + misses)

# Memory usage
redis-cli INFO memory | grep -E "used_memory_human|mem_fragmentation_ratio"

# Slow queries
redis-cli SLOWLOG GET 10

# Monitor real-time commands
redis-cli MONITOR

# Key statistics
redis-cli --bigkeys    # tìm key lớn nhất
redis-cli --hotkeys    # tìm key được access nhiều nhất (cần maxmemory-policy allkeys-lfu)
```

Prometheus metrics quan trọng

```
# redis_exporter metrics
redis_keyspace_hits_total
redis_keyspace_misses_total
```

```
redis_evicted_keys_total
redis_memory_used_bytes
redis_connected_clients
redis_replication_lag
```

---

10. Ví dụ: Setup Redis Cluster Production

```yaml
# docker-compose.yml cho Redis Cluster 6 nodes
version: '3.8'
services:
  redis-1:
    image: redis:7-alpine
    command: >
      redis-server
      --port 7001
      --cluster-enabled yes
      --cluster-config-file nodes.conf
      --cluster-node-timeout 5000
      --appendonly yes
      --appendfsync everysec
      --maxmemory 2gb
      --maxmemory-policy allkeys-lru
    ports:
      - "7001:7001"
    volumes:
      - redis1_data:/data
    networks:
      - redis-cluster

  redis-2:
    image: redis:7-alpine
    command: >
      redis-server
      --port 7002
      --cluster-enabled yes
      --cluster-config-file nodes.conf
      --cluster-node-timeout 5000
      --appendonly yes
      --maxmemory 2gb
      --maxmemory-policy allkeys-lru
    ports:
      - "7002:7002"
    volumes:
      - redis2_data:/data
    networks:
```

```
      - redis-cluster

  # ... redis-3 đến redis-6 tương tự

  redis-cluster-init:
    image: redis:7-alpine
    depends_on:
      - redis-1
      - redis-2
    command: >
      redis-cli --cluster create
      redis-1:7001 redis-2:7002 redis-3:7003
      redis-4:7004 redis-5:7005 redis-6:7006
      --cluster-replicas 1 --cluster-yes
    networks:
      - redis-cluster

volumes:
  redis1_data:
  redis2_data:

networks:
  redis-cluster:
    driver: bridge
```

```
# Kiểm tra cluster sau khi tạo
redis-cli -c -p 7001 CLUSTER INFO
redis-cli -c -p 7001 CLUSTER NODES

# Test failover
redis-cli -p 7001 DEBUG SLEEP 30   # mô phỏng node failure
redis-cli -p 7002 CLUSTER FAILOVER

# Thêm node mới vào cluster
redis-cli --cluster add-node new-node:7007 existing-node:7001
redis-cli --cluster reshard existing-node:7001
```

Best Practices

- Đặt TTL cho mọi key, tránh memory leak
- Dùng namespace cho key: `{service}:{entity}:{id}` (vd: `shop:product:123`)
- Monitor hit ratio > 80% là tốt, < 60% cần review
- Tránh lưu objects lớn (> 1MB) trong Redis
- Dùng Redis Cluster khi data > RAM của một node
- Không dùng `KEYS *` trong production (dùng `SCAN` thay thế)
- Bật maxmemory-policy phù hợp: `allkeys-lru` cho cache, `volatile-lru` cho mixed

- Tách biệt Redis instance cho cache và persistent data

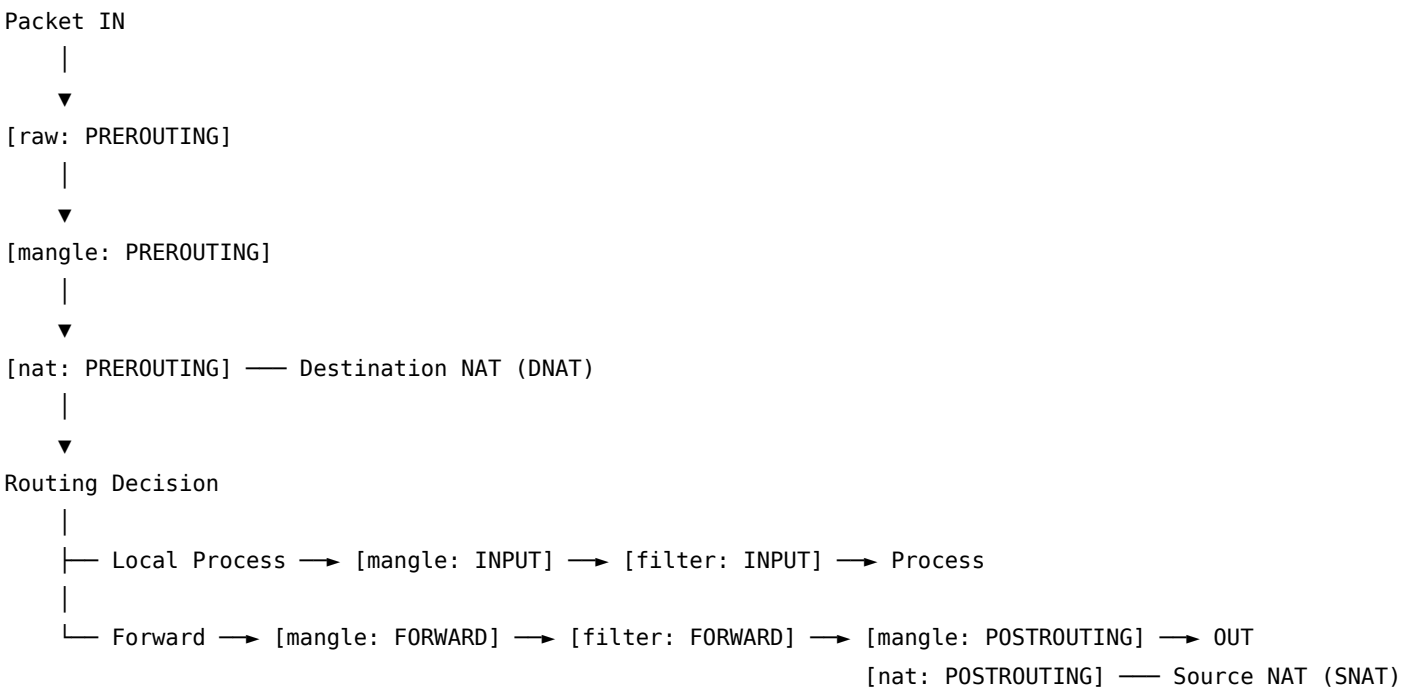
Chương 10: Firewall và Network Security

Firewall là lớp bảo vệ quan trọng nhất trong hệ thống mạng, kiểm soát luồng traffic dựa trên rules được định nghĩa sẵn.

---

1. iptables

1.1 Architecture: Tables và Chains



4 Tables chính: - `filter`: Lọc packets (INPUT, OUTPUT, FORWARD) - `nat`: Network Address Translation (PREROUTING, OUTPUT, POSTROUTING) - `mangle`: Chỉnh sửa packet headers - `raw`: Bypass connection tracking

1.2 Commands cơ bản

```
# Xem rules hiện tại
iptables -L -v -n                    # filter table
iptables -t nat -L -v -n             # nat table
iptables -L INPUT --line-numbers     # có số thứ tự

# Targets: ACCEPT, DROP, REJECT, LOG, MASQUERADE, DNAT, SNAT

# Cho phép traffic cơ bản
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
```

```
iptables -A INPUT -p tcp --dport 22 -j ACCEPT    # SSH
iptables -A INPUT -p tcp --dport 80 -j ACCEPT    # HTTP
iptables -A INPUT -p tcp --dport 443 -j ACCEPT   # HTTPS

# Default DROP policy
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# DROP vs REJECT
# DROP: không trả lời, attacker không biết port có tồn tại
# REJECT: trả về ICMP error, kết nối nhanh hơn cho legitimate users
iptables -A INPUT -p tcp --dport 3306 -j REJECT --reject-with tcp-reset

# Xóa rule
iptables -D INPUT -p tcp --dport 80 -j ACCEPT
iptables -D INPUT 3    # xóa rule số 3

# Flush tất cả rules
iptables -F
iptables -X    # xóa custom chains
```

1.3 NAT và Port Forwarding

```
# SNAT (Source NAT) - masquerade cho outbound traffic
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
# Hoặc với IP tĩnh:
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 203.0.113.1

# DNAT (Destination NAT) - port forwarding
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination 192.168.1.10:8080
# Cho phép forward packet
iptables -A FORWARD -p tcp -d 192.168.1.10 --dport 8080 -j ACCEPT

# Port forwarding từ public IP đến internal server
iptables -t nat -A PREROUTING -d 203.0.113.1 -p tcp --dport 443 \
  -j DNAT --to-destination 10.0.0.10:443
iptables -t nat -A POSTROUTING -d 10.0.0.10 -p tcp --dport 443 \
  -j MASQUERADE

# Bật IP forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward
# Permanent:
echo "net.ipv4.ip_forward = 1" >> /etc/sysctl.conf
sysctl -p
```

1.4 Connection Tracking (conntrack)

```
# Xem connections đang active
conntrack -L
conntrack -L --proto tcp --dport 80

# Stateful firewall - quan trọng nhất
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -m conntrack --ctstate INVALID -j DROP

# conntrack states:
# NEW: packet đầu tiên của connection mới
# ESTABLISHED: connection đã được thiết lập
# RELATED: connection liên quan (FTP data, ICMP error)
# INVALID: packet không match bất kỳ state nào

# Tuning conntrack table
sysctl -w net.netfilter.nf_conntrack_max=262144
sysctl -w net.netfilter.nf_conntrack_tcp_timeout_established=86400
```

1.5 Rate Limiting và Logging

```
# Rate limiting SSH (chống brute force)
iptables -A INPUT -p tcp --dport 22 -m state --state NEW \
  -m recent --set --name SSH
iptables -A INPUT -p tcp --dport 22 -m state --state NEW \
  -m recent --update --seconds 60 --hitcount 4 --name SSH \
  -j DROP

# Hoặc dùng limit module
iptables -A INPUT -p icmp -m limit --limit 5/min --limit-burst 10 -j ACCEPT

# Logging
iptables -A INPUT -j LOG --log-prefix "iptables-dropped: " --log-level 4
# Log đến /var/log/kern.log hoặc /var/log/syslog

# Lưu và restore rules
iptables-save > /etc/iptables/rules.v4
iptables-restore < /etc/iptables/rules.v4
```

---

2. nftables

nftables là replacement cho iptables, ip6tables, arptables, ebtables. Dùng single framework thay vì nhiều tools riêng biệt.

```
# Xem ruleset hiện tại
nft list ruleset

# Cú pháp nftables - rõ ràng hơn iptables
nft add table inet filter
nft add chain inet filter input { type filter hook input priority 0 \; policy drop \; }
nft add chain inet filter forward { type filter hook forward priority 0 \; policy drop \; }
nft add chain inet filter output { type filter hook output priority 0 \; policy accept \; }

# Rules
nft add rule inet filter input ct state established,related accept
nft add rule inet filter input iif lo accept
nft add rule inet filter input tcp dport { 22, 80, 443 } accept
nft add rule inet filter input drop

# Xem với line numbers
nft -a list chain inet filter input

# File config /etc/nftables.conf
```

```
# /etc/nftables.conf - câú trúc rõ ràng hơn
table inet firewall {
    chain inbound {
        type filter hook input priority 0; policy drop;

        # Cho phép established connections
        ct state { established, related } accept
        ct state invalid drop

        # Loopback
        iif lo accept

        # ICMP
        ip protocol icmp accept
        ip6 nexthdr icmpv6 accept

        # SSH với rate limiting
        tcp dport 22 ct state new limit rate 15/minute accept

        # Web
        tcp dport { 80, 443 } accept

        # Log và drop phầ̀n còn lại
        log prefix "nft-dropped: " drop
    }

    chain outbound {
        type filter hook output priority 0; policy accept;
```

```
    }

    chain forward {
        type filter hook forward priority 0; policy drop;
    }
}
```

---

3. UFW (Uncomplicated Firewall)

UFW là frontend cho iptables, phù hợp cho Ubuntu servers.

```
# Cài đặt và bật
ufw enable
ufw status verbose

# Default policies
ufw default deny incoming
ufw default allow outgoing

# Allow rules
ufw allow ssh                       # by service name
ufw allow 80/tcp
ufw allow 443/tcp
ufw allow from 192.168.1.0/24       # entire subnet
ufw allow from 10.0.0.1 to any port 5432    # PostgreSQL từ IP cụ thể

# Deny rules
ufw deny 3306/tcp
ufw deny from 192.168.1.100

# Rate limiting
ufw limit ssh    # giới hạn 6 connections/30s

# Xóa rule
ufw delete allow 80/tcp
ufw delete 5    # theo số thứ tự

# Reset tất cả
ufw reset

# Logging
ufw logging on
ufw logging medium    # levels: off, low, medium, high, full
```

---

4. firewalld

firewalld dùng zone-based model, phổ biến trên RHEL/CentOS.

```
# Quản lý service
systemctl enable --now firewalld
firewall-cmd --state

# Zones: drop, block, public, external, internal, dmz, work, home, trusted
firewall-cmd --get-default-zone
firewall-cmd --set-default-zone=public
firewall-cmd --get-active-zones

# Gán interface vào zone
firewall-cmd --zone=public --add-interface=eth0 --permanent

# Mở services
firewall-cmd --zone=public --add-service=https --permanent
firewall-cmd --zone=public --add-service=http --permanent
firewall-cmd --zone=public --add-port=8080/tcp --permanent

# Reload để áp dụng thay đổi permanent
firewall-cmd --reload

# Rich rules - rules phức tạp
firewall-cmd --zone=public --add-rich-rule='
  rule family="ipv4"
  source address="192.168.1.0/24"
  service name="ssh"
  accept' --permanent

# Rate limiting với rich rules
firewall-cmd --zone=public --add-rich-rule='
  rule family="ipv4"
  service name="ssh"
  limit value="4/m"
  accept' --permanent

# Port forwarding
firewall-cmd --zone=public --add-forward-port=port=80:proto=tcp:toport=8080 --permanent

# Xem tất cả rules
firewall-cmd --zone=public --list-all
```

---

5. Stateful vs Stateless Firewalls

Loại	Mô tả	Pros	Cons
Stateless	Kiểm tra từng packet độc lập	Nhanh, đơn giản	Không theo dõi sessions
Stateful	Theo dõi connection state	An toàn hơn	Tốn memory
Application	Hiểu protocol (HTTP, DNS, etc.)	Phát hiện deep threats	Chậm, tốn CPU

---

6. Network Segmentation và DMZ

```
Internet
    |
    ▼
[Firewall ngoài]
    |
    ├──► DMZ Zone (Public-facing servers)
    |    ├── Web Server :80/443
    |    ├── Mail Server :25/465/587
    |    └── DNS Server :53
    |
[Firewall trong]
    |
    ├──► Application Zone
    |    ├── App Server :8080
    |    └── API Server :3000
    |
    └──► Database Zone
         ├── PostgreSQL :5432
         └── Redis :6379
```

```bash
# Rules cho DMZ setup
# Firewall ngoài - chỉ cho phép public traffic vào DMZ
iptables -A FORWARD -i eth0 -o eth1 -p tcp --dport 80 -j ACCEPT   # eth1 = DMZ
iptables -A FORWARD -i eth0 -o eth1 -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Ngăn DMZ access vào Internal
iptables -A FORWARD -i eth1 -o eth2 -j DROP   # eth2 = Internal

# Cho phép DMZ servers gửi request đến App Zone (cụ thể)
iptables -A FORWARD -i eth1 -o eth2 -p tcp -s 10.0.1.10 -d 10.0.2.0/24 --dport 8080 -j ACCEPT
```

---

7. Cloud Firewalls

7.1 AWS Security Groups

```
# Security Groups: stateful, instance-level
aws ec2 create-security-group \
  --group-name web-sg \
  --description "Web server security group"

# Inbound rules
aws ec2 authorize-security-group-ingress \
  --group-id sg-12345678 \
  --protocol tcp --port 443 \
  --cidr 0.0.0.0/0

# Cho phép từ security group khác (app tier)
aws ec2 authorize-security-group-ingress \
  --group-id sg-db-12345678 \
  --protocol tcp --port 5432 \
  --source-group sg-app-12345678
```

7.2 AWS Network ACLs

```
# NACLs: stateless, subnet-level, rules evaluated by number order
# Phải define cả inbound VÀ outbound rules cho established connections

Inbound Rules:
100  ALLOW  TCP  0.0.0.0/0  :443
200  ALLOW  TCP  0.0.0.0/0  :1024-65535  # ephemeral ports
*    DENY   ALL  0.0.0.0/0

Outbound Rules:
100  ALLOW  TCP  0.0.0.0/0  :1024-65535  # ephemeral ports response
*    DENY   ALL  0.0.0.0/0
```

---

8. Best Practices

```
# 1. Default deny policy
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# 2. Luôn cho phép loopback và established connections
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
# 3. Log dropped packets
iptables -A INPUT -j LOG --log-prefix "DROPPED: " --log-level warning

# 4. Bảo vệ chống SYN flood
iptables -A INPUT -p tcp --syn -m limit --limit 1/s --limit-burst 3 -j ACCEPT
sysctl -w net.ipv4.tcp_syncookies=1

# 5. Ngăn IP spoofing
iptables -A INPUT -s 10.0.0.0/8 -i eth0 -j DROP     # private IP từ public interface
iptables -A INPUT -s 172.16.0.0/12 -i eth0 -j DROP
iptables -A INPUT -s 192.168.0.0/16 -i eth0 -j DROP
iptables -A INPUT -s 127.0.0.0/8 -i eth0 -j DROP

# 6. Chặn ICMP redirect
sysctl -w net.ipv4.conf.all.accept_redirects=0
sysctl -w net.ipv4.conf.all.send_redirects=0
```

---

9. Ví dụ: Config Firewall Production

Web Server

```
#!/bin/bash
# web-server-firewall.sh

# Flush existing rules
iptables -F
iptables -X
iptables -t nat -F

# Default policies
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# Loopback và established
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -m conntrack --ctstate INVALID -j DROP

# ICMP ping (rate limited)
iptables -A INPUT -p icmp --icmp-type echo-request -m limit --limit 1/s -j ACCEPT

# SSH từ management network only
iptables -A INPUT -p tcp --dport 22 -s 10.0.100.0/24 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j DROP
```

```
# HTTP/HTTPS
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Log và drop phần còn lại
iptables -A INPUT -j LOG --log-prefix "WEB-FW-DROPPED: "
iptables -A INPUT -j DROP

# Save rules
iptables-save > /etc/iptables/rules.v4
```

Database Server

```
#!/bin/bash
# database-server-firewall.sh

iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP

# Loopback và established
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# SSH từ management network
iptables -A INPUT -p tcp --dport 22 -s 10.0.100.0/24 -j ACCEPT

# PostgreSQL chỉ từ application servers
iptables -A INPUT -p tcp --dport 5432 -s 10.0.1.0/24 -j ACCEPT

# Redis chỉ từ application servers
iptables -A INPUT -p tcp --dport 6379 -s 10.0.1.0/24 -j ACCEPT

# Log drops
iptables -A INPUT -j LOG --log-prefix "DB-FW-DROPPED: " --log-level 4

iptables-save > /etc/iptables/rules.v4
```

---

10. Monitoring và Audit

```
# Xem traffic stats theo rule
iptables -L -v -n | awk '{print $1, $2, $11}'
```

```
# Theo dõi log real-time
tail -f /var/log/kern.log | grep "FW-DROPPED"

# Kiểm tra connections hiện tại
ss -tulpn                          # listening ports
conntrack -L --proto tcp           # active connections
netstat -an | grep ESTABLISHED | wc -l

# Audit với fail2ban
fail2ban-client status
fail2ban-client status sshd

# Kiểm tra open ports từ ngoài
nmap -sT -O target-ip
nmap -sV --version-intensity 5 target-ip
```

Chương 11: Load Balancing

Load balancer phân phối traffic đến nhiều servers, tăng availability, scalability và performance.

---

1. L4 vs L7 Load Balancing

Layer 4 (Transport Layer)

- Load balance dựa trên IP + Port
- Không đọc nội dung packet (không decrypt TLS)
- Nhanh hơn, ít CPU hơn
- Dùng khi cần throughput cao: gaming, streaming, database

```
Client → LB (sees IP:port only) → Backend
192.168.1.1:52134 → 10.0.0.1:3306 (MySQL)
```

Layer 7 (Application Layer)

- Load balance dựa trên HTTP headers, URL, cookies, body
- Cần decrypt TLS để đọc content
- Routing thông minh hơn: `/api/*` → API servers, `/static/*` → CDN
- Dùng cho web applications

```
Client → LB (reads HTTP headers/URL) → Backend
GET /api/users → API cluster
GET /images/logo.png → Static cluster
```

---

2. Algorithms

Round-Robin

```
Request 1 → Server A
Request 2 → Server B
Request 3 → Server C
Request 4 → Server A  (vòng lại)
```

Phù hợp khi các servers có cùng specs và requests có thời gian xử lý tương đương.

Weighted Round-Robin

```
Server A (weight 3): nhận 3/6 requests
Server B (weight 2): nhận 2/6 requests
Server C (weight 1): nhận 1/6 requests
```

Dùng khi servers có specs khác nhau.

Least Connections

```
Server A: 100 active connections
Server B: 50 active connections  ← Request mới đến đây
Server C: 80 active connections
```

Tốt cho long-lived connections (WebSocket, database).

IP Hash

```
hash(client_ip) % num_servers = server_index
```

Cùng client IP luôn đến cùng server. Dùng cho sticky sessions không cần cookie.

Consistent Hashing

Khi thêm/xóa server, chỉ một phần nhỏ keys bị redistribute (khác với mod hashing). Phổ biến trong distributed caching (Redis Cluster, Memcached).

---

3. HAProxy

3.1 Cấu trúc Config

```
# /etc/haproxy/haproxy.cfg

global
    log /dev/log local0
    log /dev/log local1 notice
    chroot /var/lib/haproxy
```

```
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon
    maxconn 50000

defaults
    log     global
    mode    http                    # http hoặc tcp
    option  httplog
    option  dontlognull
    option  forwardfor              # thêm X-Forwarded-For header
    option  http-server-close       # tắt keep-alive với backend
    timeout connect 5s
    timeout client  50s
    timeout server  50s
    retries 3

#--- Stats Page ---
listen stats
    bind *:8404
    stats enable
    stats uri /stats
    stats refresh 30s
    stats auth admin:secretpassword
    stats show-legends
    stats show-node

#--- Frontend ---
frontend http-in
    bind *:80
    bind *:443 ssl crt /etc/ssl/certs/example.com.pem

    # Redirect HTTP to HTTPS
    redirect scheme https code 301 if !{ ssl_fc }

    # ACL-based routing
    acl is_api path_beg /api/
    acl is_static path_beg /static/ /images/ /css/ /js/
    acl is_websocket hdr(Upgrade) -i websocket

    use_backend api-servers if is_api
    use_backend static-servers if is_static
    use_backend ws-servers if is_websocket
    default_backend web-servers

#--- Backends ---
```

```
backend web-servers
    balance roundrobin
    option httpchk GET /health HTTP/1.1\r\nHost:\ example.com
    http-check expect status 200

    server web1 10.0.1.10:8080 check inter 2s rise 2 fall 3 weight 3
    server web2 10.0.1.11:8080 check inter 2s rise 2 fall 3 weight 3
    server web3 10.0.1.12:8080 check inter 2s rise 2 fall 3 weight 1

backend api-servers
    balance leastconn
    option httpchk GET /api/health
    http-check expect status 200
    timeout server 30s        # API có thể cần xử lý lâu hơn

    server api1 10.0.2.10:3000 check
    server api2 10.0.2.11:3000 check
    # Backup server - chỉ dùng khi các server chính down
    server api-backup 10.0.2.12:3000 check backup

backend static-servers
    balance uri                  # same URI luôn đến cùng server (CDN-like)
    server static1 10.0.3.10:80 check
    server static2 10.0.3.11:80 check

backend ws-servers
    balance source               # sticky bằng IP hash
    timeout tunnel 1h            # WebSocket cần timeout dài
    server ws1 10.0.4.10:8080 check
    server ws2 10.0.4.11:8080 check
```

3.2 HAProxy ACLs

```
frontend http-in
    bind *:443 ssl crt /etc/ssl/certs/bundle.pem

    # ACL theo IP
    acl trusted_ips src 10.0.0.0/8 192.168.0.0/16
    acl blocked_ips src -f /etc/haproxy/blocked-ips.txt

    # ACL theo HTTP header
    acl is_mobile hdr_sub(User-Agent) -i mobile android iphone
    acl has_auth req.hdr(Authorization) -m found
    acl is_json req.hdr(Content-Type) -i application/json

    # ACL theo URL path và query
    acl admin_path path_beg /admin
```

```
    acl api_v2 path_beg /api/v2/

    # Kết hợp ACLs
    http-request deny if blocked_ips
    http-request deny if admin_path !trusted_ips
    use_backend api-v2 if api_v2

    # Rate limiting
    http-request track-sc0 src table per_ip_rates
    http-request deny deny_status 429 if { sc_http_req_rate(0) gt 100 }

backend per_ip_rates
    stick-table type ip size 1m expire 10s store http_req_rate(10s)
```

3.3 Health Checks

```
backend api-servers
    # HTTP health check
    option httpchk GET /health HTTP/1.1\r\nHost:\ api.example.com
    http-check expect status 200
    http-check expect string "ok"

    # TCP health check (layer 4)
    option tcp-check

    # External health check
    external-check path "/usr/bin:/bin"
    external-check command /usr/lib/haproxy/healthcheck.sh

    server api1 10.0.2.10:3000 check inter 3s rise 2 fall 3
    # inter: interval giữa checks
    # rise: số lần check success để mark UP
    # fall: số lần check fail để mark DOWN
```

3.4 SSL Offloading

```
frontend https-in
    bind *:443 ssl crt /etc/ssl/private/example.pem \
        ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256 \
        no-sslv3 no-tlsv10 no-tlsv11

    # Thêm security headers
    http-response set-header Strict-Transport-Security "max-age=31536000; includeSubDomains"
    http-response set-header X-Content-Type-Options nosniff

    # Pass proto info đến backend
```

```
    http-request set-header X-Forwarded-Proto https
    default_backend web-servers

backend web-servers
    # Backend nhận plain HTTP (SSL đã được offload tại LB)
    server web1 10.0.1.10:8080 check
```

---

4. Nginx Load Balancing

```
# /etc/nginx/nginx.conf

upstream api_backend {
    least_conn;                    # algorithm: least connections

    server 10.0.1.10:3000 weight=3;
    server 10.0.1.11:3000 weight=2;
    server 10.0.1.12:3000 backup;  # backup server

    keepalive 32;                  # persistent connections đến backend
}

upstream static_backend {
    ip_hash;                       # sticky session bằng IP

    server 10.0.2.10:80;
    server 10.0.2.11:80;
}

server {
    listen 80;
    server_name example.com;

    location /api/ {
        proxy_pass http://api_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        # Health check (Nginx Plus feature, hoặc dùng ngx_http_upstream_check_module)
        # health_check interval=5s fails=3 passes=2;
    }

    location /static/ {
        proxy_pass http://static_backend;
    }
```

```
}
```

---

5. Cloud Load Balancers (AWS)

Application Load Balancer (ALB) - L7

```
# Tạo ALB
aws elbv2 create-load-balancer \
  --name my-alb \
  --type application \
  --subnets subnet-12345 subnet-67890 \
  --security-groups sg-12345678

# Target Group
aws elbv2 create-target-group \
  --name api-targets \
  --protocol HTTP \
  --port 3000 \
  --vpc-id vpc-12345678 \
  --health-check-path /health \
  --healthy-threshold-count 2 \
  --unhealthy-threshold-count 3

# Listener với rules
aws elbv2 create-listener \
  --load-balancer-arn arn:aws:elasticloadbalancing:... \
  --protocol HTTPS --port 443 \
  --certificates CertificateArn=arn:aws:acm:... \
  --default-actions Type=forward,TargetGroupArn=arn:aws:...

# Path-based routing rule
aws elbv2 create-rule \
  --listener-arn arn:... \
  --priority 10 \
  --conditions Field=path-pattern,Values='/api/*' \
  --actions Type=forward,TargetGroupArn=arn:...
```

Network Load Balancer (NLB) - L4

```
# NLB: ultra-low latency, millions of requests/sec, static IP
aws elbv2 create-load-balancer \
  --name my-nlb \
  --type network \
  --subnets subnet-12345 subnet-67890
```

---

6. Session Persistence (Sticky Sessions)

Cookie-based (khuyến nghị)

```
backend web-servers
    balance roundrobin
    cookie SERVERID insert indirect nocache
    server web1 10.0.1.10:8080 check cookie web1
    server web2 10.0.1.11:8080 check cookie web2
    # LB inject cookie SERVERID=web1 vào response
    # Requests sau với SERVERID=web1 luôn đến web1
```

```
# Nginx với sticky module
upstream backend {
    sticky cookie srv_id expires=1h domain=.example.com path=/;
    server 10.0.1.10:8080;
    server 10.0.1.11:8080;
}
```

---

7. Connection Draining và Graceful Shutdown

```
# Graceful shutdown: đợi requests hiện tại hoàn thành
systemctl stop haproxy   # gửi SIGTERM, HAProxy drain connections trước khi stop

# Tạm thời disable server (maintenance)
echo "disable server web-servers/web1" | socat stdio /run/haproxy/admin.sock
# Sau maintenance:
echo "enable server web-servers/web1" | socat stdio /run/haproxy/admin.sock

# Drain mode: không nhận request mới nhưng hoàn thành request cũ
echo "set server web-servers/web1 state drain" | socat stdio /run/haproxy/admin.sock
```

---

8. High Availability với Keepalived

```
# keepalived.conf trên LB1 (MASTER)
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 100              # MASTER có priority cao hơn
```

```
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass secretpass
    }

    virtual_ipaddress {
        192.168.1.100/24      # Virtual IP (VIP) - floating IP
    }

    track_script {
        chk_haproxy
    }
}

vrrp_script chk_haproxy {
    script "killall -0 haproxy"
    interval 2
    weight -20    # giảm priority nếu haproxy chết
}
```

```
# keepalived.conf trên LB2 (BACKUP)
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 51
    priority 90               # thấp hơn MASTER
    ...
}
```

---

9. Blue/Green và Canary Deployment với LB

Blue/Green với HAProxy

```
backend web-servers
    # Blue (production) - nhận 100% traffic
    server blue1 10.0.1.10:8080 check weight 100
    server blue2 10.0.1.11:8080 check weight 100

    # Green (new version) - weight 0, sẵn sàng nhưng không nhận traffic
    server green1 10.0.2.10:8080 check weight 0
    server green2 10.0.2.11:8080 check weight 0

# Khi deploy:
```

```
# 1. Test green servers
# 2. Chuyển traffic: set server web-servers/green1 weight 100
# 3. Drain blue: set server web-servers/blue1 weight 0
```

Canary Deployment

```
backend web-servers
    balance roundrobin
    # 90% production, 10% canary
    server prod1 10.0.1.10:8080 check weight 45
    server prod2 10.0.1.11:8080 check weight 45
    server canary1 10.0.2.10:8080 check weight 10
```

```
# Nginx canary với split_clients
split_clients "${remote_addr}${http_user_agent}" $variant {
    10%     canary;
    *       production;
}

upstream canary { server 10.0.2.10:8080; }
upstream production {
    server 10.0.1.10:8080;
    server 10.0.1.11:8080;
}

server {
    location / {
        proxy_pass http://$variant;
    }
}
```

---

10. Ví dụ: HAProxy Production Config

```
# /etc/haproxy/haproxy.cfg - Production setup

global
    log /dev/log local0 info
    maxconn 100000
    nbthread 4                    # dùng 4 CPU threads
    cpu-map auto:1/1-4 0-3        # bind threads vào CPU cores
    tune.ssl.default-dh-param 2048
    ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256
    ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets
```

```
defaults
    mode http
    log global
    option httplog
    option dontlognull
    option forwardfor except 127.0.0.0/8
    option http-server-close
    option redispatch              # nêú server down, thử server khác
    retries 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s

listen stats
    bind *:8404
    stats enable
    stats uri /
    stats refresh 5s
    stats auth haproxy:$(cat /etc/haproxy/stats-password)
    stats show-legends
    acl local_admin src 10.0.100.0/24
    stats http-request deny unless local_admin

frontend www-http
    bind *:80
    redirect scheme https code 301

frontend www-https
    bind *:443 ssl crt /etc/ssl/haproxy/bundle.pem alpn h2,http/1.1
    http-request set-header X-Forwarded-Proto https
    http-request add-header X-Real-IP %[src]

    # Security headers
    http-response set-header X-Frame-Options SAMEORIGIN
    http-response set-header X-XSS-Protection "1; mode=block"
    http-response set-header X-Content-Type-Options nosniff

    # Rate limiting
    stick-table type binary len 8 size 1m expire 10s store http_req_rate(10s)
    http-request track-sc0 src
    http-request deny deny_status 429 if { sc_http_req_rate(0) gt 200 }

    acl is_api path_beg /api/
    use_backend api-cluster if is_api
```

```
    default_backend web-cluster

backend web-cluster
    balance roundrobin
    option httpchk GET /ping
    http-check expect status 200
    default-server inter 3s rise 2 fall 3 slowstart 60s maxconn 300 maxqueue 128

    server web-01 10.0.1.10:8080 check
    server web-02 10.0.1.11:8080 check
    server web-03 10.0.1.12:8080 check

backend api-cluster
    balance leastconn
    timeout server 60s
    option httpchk GET /api/health
    http-check expect status 200

    server api-01 10.0.2.10:3000 check inter 5s
    server api-02 10.0.2.11:3000 check inter 5s
    server api-03 10.0.2.12:3000 check inter 5s backup
```

```
# Validate config
haproxy -c -f /etc/haproxy/haproxy.cfg

# Reload không drop connections
systemctl reload haproxy
# Hoặc:
haproxy -sf $(cat /var/run/haproxy.pid) -f /etc/haproxy/haproxy.cfg

# Monitoring via socket
echo "show info" | socat stdio /run/haproxy/admin.sock
echo "show stat" | socat stdio /run/haproxy/admin.sock | cut -d',' -f1,2,5,18,19
```

Best Practices

- Luôn dùng health checks với rise/fall thresholds để tránh flapping
- Cấu hình timeouts phù hợp với ứng dụng (APIs vs WebSockets khác nhau)
- Đặt maxconn để tránh overload backend servers
- Dùng connection draining trước khi deploy/restart
- Cấu hình Keepalived cho HA của chính load balancer
- Monitor queue depth, response time, error rate thay vì chỉ connection count
- SSL offloading tại LB giúp backend đơn giản hơn và giảm CPU overhead
- Tách biệt LB cho internal và external traffic

Chương 12: Docker Deep Dive

Docker sử dụng các tính năng của Linux kernel để tạo isolated environments cho applications mà không cần full virtual machine.

---

1. Container Internals

1.1 Linux Namespaces

Namespaces cô lập các resources của container khỏi host và containers khác:

Namespace	Cô lập	Lệnh kiểm tra
pid	Process IDs	`ls /proc/[container_pid]/ns/pid`
net	Network stack	`ip netns list`
mnt	Filesystem mounts	`cat /proc/mounts`
uts	Hostname	`hostname`
ipc	IPC, message queues	`ipcs`
user	User/group IDs	`id`
cgroup	cgroup hierarchy	`/proc/self/cgroup`

```
# Container chạy process với PID 1 trong namespace riêng
# Nhưng host thấy PID thực của container process
docker run -d nginx
ps aux | grep nginx    # thấy PID thực trên host

# Xem namespaces của container
docker inspect <container-id> | grep -i pid
ls -la /proc/<host-pid>/ns/
```

1.2 cgroups (Control Groups)

cgroups giới hạn và theo dõi resource usage:

```
# Xem cgroup của container
cat /proc/$(docker inspect --format '{{.State.Pid}}' mycontainer)/cgroup

# Container resources trong cgroup filesystem
ls /sys/fs/cgroup/memory/docker/<container-id>/
cat /sys/fs/cgroup/memory/docker/<container-id>/memory.limit_in_bytes
cat /sys/fs/cgroup/cpu/docker/<container-id>/cpu.cfs_quota_us
```

1.3 Union Filesystem (OverlayFS)

```
Layer 5: Container writable layer (thin)    ← Mọi thay đổi ghi vào đây
Layer 4: App code (image layer, read-only)
```

```
Layer 3: Python packages (image layer, read-only)
Layer 2: Ubuntu packages (image layer, read-only)
Layer 1: Base Ubuntu (image layer, read-only)
```

```
# Xem layers của image
docker history my-image

# OverlayFS trên host
docker inspect <container-id> | jq '.[0].GraphDriver'
# Shows: upperdir (writable), lowerdir (image layers), mergeddir (union view)

ls /var/lib/docker/overlay2/<layer-id>/diff/
```

---

2. Dockerfile Best Practices

2.1 Multi-stage Build

```
# Go application - multi-stage build
# Stage 1: Build
FROM golang:1.22-alpine AS builder
WORKDIR /app

# Copy go.mod và go.sum trước (tận dụng layer cache)
COPY go.mod go.sum ./
RUN go mod download

# Copy source code
COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo \
    -ldflags="-w -s" \    # strip debug symbols, giảm size
    -o server ./cmd/server

# Stage 2: Production image (distroless)
FROM gcr.io/distroless/static-debian12
WORKDIR /app
COPY --from=builder /app/server .
COPY --from=builder /app/config ./config

# Không có shell trong distroless, không thể bị exploited
USER nonroot:nonroot
EXPOSE 8080
ENTRYPOINT ["/app/server"]
```

```
# Python application - multi-stage
FROM python:3.12-slim AS base
WORKDIR /app
```

```
# Build stage: cài dev dependencies
FROM base AS builder
COPY requirements.txt .
RUN pip install --user --no-cache-dir -r requirements.txt

# Production stage
FROM base AS production
# Copy chỉ installed packages, không copy pip cache
COPY --from=builder /root/.local /root/.local
ENV PATH=/root/.local/bin:$PATH

COPY . .

# Non-root user
RUN addgroup --system appgroup && adduser --system --group appuser
USER appuser

EXPOSE 8000
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

2.2 Layer Caching Optimization

```
# Sai: copy tất cả trước - mọi thay đổi code invalidate cache pip install
FROM python:3.12-slim
COPY . .
RUN pip install -r requirements.txt

# Đúng: copy dependency files trước
FROM python:3.12-slim
COPY requirements.txt .
RUN pip install -r requirements.txt    # cached nếu requirements.txt không đổi
COPY . .                               # chỉ invalidate layers sau
```

2.3 .dockerignore

```
# .dockerignore
.git
.gitignore
**/*.md
**/__pycache__
**/.pytest_cache
**/*.pyc
.env
.env.*
node_modules
```

```
*.log
.DS_Store
coverage/
dist/
build/
```

2.4 Security trong Dockerfile

```
FROM node:20-alpine

WORKDIR /app

# Cài dependencies với exact versions
COPY package.json package-lock.json ./
RUN npm ci --only=production && npm cache clean --force

COPY . .

# Không chạy với root
RUN addgroup -S appgroup && adduser -S appuser -G appgroup
USER appuser

# Readonly filesystem (khai báo intent)
VOLUME ["/tmp"]

EXPOSE 3000
CMD ["node", "server.js"]
```

---

3. Docker Networking

```
# Các network drivers
docker network ls
# bridge: default, isolated network trên host
# host: dùng network namespace của host (performance cao)
# overlay: cho Swarm/multi-host networking
# macvlan: container có MAC address riêng, trực tiếp trên physical network
# none: không có network

# Bridge network (default)
docker run -d --name web nginx
docker run -d --name api myapp
# web và api có thể communicate qua container name trong cùng network

# Custom bridge network
```

```
docker network create --driver bridge \
  --subnet 172.20.0.0/16 \
  --ip-range 172.20.240.0/20 \
  app-network

docker run -d --name web --network app-network nginx
docker run -d --name api --network app-network myapp
# Trong app-network: curl http://api:3000 (DNS tự động)

# Host networking - performance cao nhất
docker run -d --network host nginx
# Nginx lắng nghe trực tiếp trên port 80 của host

# Overlay network cho multi-host (Swarm)
docker network create -d overlay --attachable my-overlay
```

---

4. Docker Volumes

```
# Bind mounts: ánh xạ thư mục host vào container
docker run -v /host/path:/container/path:ro nginx
# :ro = read-only, :rw = read-write (default)

# Named volumes: Docker quản lý location
docker volume create mydata
docker run -v mydata:/app/data myapp
docker volume inspect mydata    # xem mountpoint: /var/lib/docker/volumes/mydata/_data

# tmpfs mount: lưu trong memory, không persist
docker run --tmpfs /tmp:size=100m,mode=1777 myapp

# Volume với backup
docker run --rm \
  -v mydata:/source:ro \
  -v $(pwd)/backup:/backup \
  alpine tar czf /backup/mydata-$(date +%Y%m%d).tar.gz -C /source .
```

---

5. Docker Compose

```
# docker-compose.yml - production-ready setup
version: '3.8'

services:
```

```yaml
  web:
    build:
      context: .
      dockerfile: Dockerfile
      target: production          # multi-stage target
      args:
        - APP_VERSION=${APP_VERSION}
    image: myapp:${APP_VERSION}
    restart: unless-stopped
    ports:
      - "8080:8080"
    environment:
      - NODE_ENV=production
      - DATABASE_URL=postgresql://user:pass@postgres:5432/mydb
    env_file:
      - .env.production
    volumes:
      - uploads:/app/uploads
    networks:
      - frontend
      - backend
    depends_on:
      postgres:
        condition: service_healthy
      redis:
        condition: service_healthy
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
    deploy:
      resources:
        limits:
          cpus: '1.0'
          memory: 512M
        reservations:
          cpus: '0.25'
          memory: 128M
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"

  postgres:
    image: postgres:16-alpine
```

```
    restart: unless-stopped
    environment:
      POSTGRES_DB: mydb
      POSTGRES_USER: user
      POSTGRES_PASSWORD_FILE: /run/secrets/db_password
    secrets:
      - db_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init-scripts:/docker-entrypoint-initdb.d:ro
    networks:
      - backend
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U user -d mydb"]
      interval: 10s
      timeout: 5s
      retries: 5

  redis:
    image: redis:7-alpine
    restart: unless-stopped
    command: redis-server --maxmemory 256mb --maxmemory-policy allkeys-lru
    volumes:
      - redis_data:/data
    networks:
      - backend
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 3s
      retries: 3

  nginx:
    image: nginx:alpine
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./ssl:/etc/nginx/ssl:ro
      - static_files:/var/www/static:ro
    networks:
      - frontend
    depends_on:
      - web

volumes:
```

```
  postgres_data:
  redis_data:
  uploads:
  static_files:

networks:
  frontend:
    driver: bridge
  backend:
    driver: bridge
    internal: true    # không có external connectivity

secrets:
  db_password:
    file: ./secrets/db_password.txt
```

```
# Compose commands
docker compose up -d                    # start in background
docker compose up -d --build            # rebuild images
docker compose up -d --scale web=3      # scale service
docker compose logs -f web              # follow logs
docker compose exec web sh              # shell vào container
docker compose ps                       # trạng thái services
docker compose down -v                  # stop và xóa volumes
docker compose config                   # validate và print config
```

---

6. Image Security

6.1 Scanning

```
# Docker Scout (built-in từ Docker Desktop 4.17+)
docker scout cves myimage:latest
docker scout recommendations myimage:latest

# Trivy - open source scanner
trivy image myimage:latest
trivy image --severity HIGH,CRITICAL myimage:latest
trivy image --ignore-unfixed myimage:latest

# Grype
grype myimage:latest
```

6.2 Image Signing

```
# Docker Content Trust
export DOCKER_CONTENT_TRUST=1
docker push myregistry/myapp:1.0.0    # auto-sign khi push
docker pull myregistry/myapp:1.0.0    # verify signature khi pull

# Cosign (Sigstore)
cosign generate-key-pair
cosign sign --key cosign.key myregistry/myapp:1.0.0
cosign verify --key cosign.pub myregistry/myapp:1.0.0
```

6.3 Minimal Base Images

```
# Distroless - không có shell, package manager
FROM gcr.io/distroless/java21-debian12
COPY app.jar /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]

# Alpine - nhỏ nhưng có shell và package manager
FROM alpine:3.19
RUN apk add --no-cache curl ca-certificates

# Scratch - hoàn toàn trống, dùng với statically-linked binaries
FROM scratch
COPY --from=builder /app/server /server
EXPOSE 8080
ENTRYPOINT ["/server"]
```

---

7. Docker Registry

```
# Private registry với Docker Registry
docker run -d -p 5000:5000 \
  -v /data/registry:/var/lib/registry \
  --name registry \
  registry:2

# Push/pull từ private registry
docker tag myapp:latest localhost:5000/myapp:latest
docker push localhost:5000/myapp:latest
docker pull localhost:5000/myapp:latest
```

```
# Harbor (enterprise registry) - docker-compose quickstart
version: '3'
```

```
services:
  harbor-core:
    image: goharbor/harbor-core:v2.10.0
    # ...
  # Harbor bao gồm: Core, Portal, Registry, JobService, Database, Redis, Notary, Trivy
```

---

8. Resource Limits

```
# Memory limit
docker run -m 512m --memory-swap 512m myapp    # --memory-swap = total memory + swap
# --memory-swap = memory = không dùng swap

# CPU limit
docker run --cpus="1.5" myapp          # tối đa 1.5 CPU cores
docker run --cpu-shares=512 myapp      # relative weight (default 1024)
docker run --cpuset-cpus="0,1" myapp   # chỉ dùng CPU 0 và 1

# PID limit (tránh fork bomb)
docker run --pids-limit 100 myapp

# Read-only filesystem
docker run --read-only --tmpfs /tmp myapp

# Drop capabilities
docker run --cap-drop ALL --cap-add NET_BIND_SERVICE nginx
```

---

9. Logging Drivers

```
# Default: json-file
docker run --log-driver json-file \
  --log-opt max-size=10m \
  --log-opt max-file=3 \
  myapp

# Syslog
docker run --log-driver syslog \
  --log-opt syslog-address=udp://logs.example.com:514 \
  myapp

# Fluentd
docker run --log-driver fluentd \
  --log-opt fluentd-address=localhost:24224 \
```

```
  --log-opt tag="docker.{{.Name}}" \
  myapp

# Không lưu logs (stateless services)
docker run --log-driver none myapp
```

```
// /etc/docker/daemon.json - set default log driver
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3",
    "labels": "production_status",
    "env": "APP_VERSION"
  }
}
```

---

10. Docker Production Configuration

```
// /etc/docker/daemon.json
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "50m",
    "max-file": "5"
  },
  "storage-driver": "overlay2",
  "default-ulimits": {
    "nofile": {
      "Name": "nofile",
      "Hard": 64000,
      "Soft": 64000
    }
  },
  "metrics-addr": "127.0.0.1:9323",
  "experimental": true,
  "live-restore": true,
  "userland-proxy": false,
  "no-new-privileges": true,
  "icc": false,
  "default-address-pools": [
    {"base": "172.17.0.0/12", "size": 24}
  ]
}
```

```
# Áp dụng config mà không restart daemon (nếu có thể)
kill -HUP $(pidof dockerd)
# Hoặc:
systemctl reload docker

# live-restore: containers tiếp tục chạy khi daemon restart
```

---

11. Debugging Containers

```
# Xem logs
docker logs -f --tail 100 mycontainer
docker logs --since 1h mycontainer

# Shell vào container
docker exec -it mycontainer sh           # alpine/busybox
docker exec -it mycontainer bash         # ubuntu/debian
docker exec -it mycontainer /bin/sh      # distroless không có shell!

# Debug distroless container với ephemeral container
docker debug mycontainer                 # Docker Desktop 4.27+
kubectl debug -it pod-name --image=busybox --target=container-name   # Kubernetes

# Inspect
docker inspect mycontainer
docker inspect mycontainer | jq '.[0].NetworkSettings'
docker inspect mycontainer | jq '.[0].Mounts'

# Resource usage real-time
docker stats
docker stats --no-stream --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}"

# Xem processes trong container
docker top mycontainer

# Copy files
docker cp mycontainer:/app/logs/error.log ./error.log
docker cp ./config.json mycontainer:/app/config.json

# Events
docker events --since 1h --filter 'type=container' --filter 'event=die'
```

---

12. Ví dụ: Multi-stage Dockerfile Production

```
# Go application - production ready
FROM golang:1.22-alpine AS base
RUN apk add --no-cache git ca-certificates tzdata
WORKDIR /build

FROM base AS dependencies
COPY go.mod go.sum ./
RUN go mod download && go mod verify

FROM dependencies AS builder
COPY . .
ARG VERSION=dev
ARG BUILD_TIME
RUN CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build \
    -ldflags="-w -s -X main.version=${VERSION} -X main.buildTime=${BUILD_TIME}" \
    -o server \
    ./cmd/server

# Security scan stage
FROM builder AS security
RUN go install golang.org/x/vuln/cmd/govulncheck@latest
RUN govulncheck ./...

# Final minimal image
FROM gcr.io/distroless/static-debian12 AS production
COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/
COPY --from=builder /usr/share/zoneinfo /usr/share/zoneinfo
COPY --from=builder /build/server /server
USER nonroot:nonroot
EXPOSE 8080
ENTRYPOINT ["/server"]
```

```
# Build với build args
docker build \
  --target production \
  --build-arg VERSION=$(git describe --tags) \
  --build-arg BUILD_TIME=$(date -u +%Y-%m-%dT%H:%M:%SZ) \
  -t myapp:$(git describe --tags) \
  --cache-from myregistry/myapp:latest \
  .

# Kiểm tra image size
docker images myapp
docker history myapp:latest

# Vulnerability scan
```

```
trivy image myapp:latest --exit-code 1 --severity CRITICAL
```

Best Practices

- Luôn specify exact image versions, không dùng `latest` trong production
- Dùng multi-stage builds để giảm image size và attack surface
- Chạy container với non-root user (`USER appuser`)
- Dùng `.dockerignore` để tránh copy unnecessary files
- Scan images trước khi deploy (Trivy, Docker Scout)
- Đặt resource limits (`--memory`, `--cpus`) để tránh container eat all resources
- Dùng `--read-only` filesystem khi có thể
- `live-restore=true` trong daemon.json để containers survive Docker daemon restart
- Không hardcode secrets trong Dockerfile, dùng environment variables hoặc secrets
- Monitor với `docker stats` và export metrics qua Prometheus Docker exporter

Chương 13: Configuration Management

Configuration Management (CM) là nền tảng của DevOps, giúp tự động hóa việc cài đặt, cấu hình và duy trì hệ thống một cách nhất quán, có thể lặp lại.

---

1. Tổng Quan Configuration Management

Vấn đề "Snowflake Servers"

Server được cấu hình thủ công theo thời gian trở nên độc nhất, không thể tái tạo. CM giải quyết bằng cách mã hóa mọi thay đổi vào code.

Lợi ích chính

- **Idempotency**: Chạy nhiều lần cho cùng kết quả
- **Version control**: Mọi thay đổi được lưu trong Git
- **Audit trail**: Biết ai thay đổi gì, khi nào
- **Disaster recovery**: Rebuild hệ thống từ code trong vài phút

---

2. Ansible

Ansible là tool CM phổ biến nhất, agentless, dùng SSH để kết nối.

2.1 Inventory

```
# inventory/hosts.ini
[webservers]
web1.example.com ansible_user=ubuntu
web2.example.com ansible_user=ubuntu ansible_port=2222

[databases]
db1.example.com ansible_user=postgres
db2.example.com ansible_user=postgres

[production:children]
webservers
databases

[production:vars]
ansible_python_interpreter=/usr/bin/python3
env=production
```

Dynamic Inventory (cho AWS):

```
# Dùng plugin aws_ec2
ansible-inventory -i aws_ec2.yml --list
ansible-inventory -i aws_ec2.yml --graph
```

```
# aws_ec2.yml
plugin: amazon.aws.aws_ec2
regions:
  - us-east-1
filters:
  tag:Environment: production
keyed_groups:
  - key: tags.Role
    prefix: role
```

2.2 Playbooks

```
# playbooks/deploy-webapp.yml
---
- name: Deploy Web Application
  hosts: webservers
  become: true
  vars:
    app_version: "2.1.0"
    app_port: 8080
    deploy_dir: /opt/webapp

  pre_tasks:
```

```
  - name: Update apt cache
    apt:
      update_cache: true
      cache_valid_time: 3600

tasks:
  - name: Install dependencies
    apt:
      name:
        - nginx
        - python3-pip
        - git
      state: present

  - name: Create app directory
    file:
      path: "{{ deploy_dir }}"
      state: directory
      owner: www-data
      group: www-data
      mode: '0755'

  - name: Clone application
    git:
      repo: "https://github.com/company/webapp.git"
      dest: "{{ deploy_dir }}"
      version: "{{ app_version }}"
      force: true
    notify: restart webapp

  - name: Deploy Nginx config
    template:
      src: templates/nginx.conf.j2
      dest: /etc/nginx/sites-available/webapp
      mode: '0644'
    notify: reload nginx

  - name: Enable Nginx site
    file:
      src: /etc/nginx/sites-available/webapp
      dest: /etc/nginx/sites-enabled/webapp
      state: link

handlers:
  - name: restart webapp
    systemd:
      name: webapp
      state: restarted
```

```
        daemon_reload: true

    - name: reload nginx
      service:
        name: nginx
        state: reloaded

  post_tasks:
    - name: Verify application is running
      uri:
        url: "http://localhost:{{ app_port }}/health"
        status_code: 200
      retries: 3
      delay: 5
```

2.3 Roles

```
roles/
└── webapp/
    ├── defaults/
    │   └── main.yml          # Biến mặc định (priority thấp nhất)
    ├── vars/
    │   └── main.yml          # Biến cứng (priority cao)
    ├── tasks/
    │   ├── main.yml          # Task chính
    │   ├── install.yml
    │   └── configure.yml
    ├── handlers/
    │   └── main.yml
    ├── templates/
    │   └── nginx.conf.j2
    ├── files/
    │   └── app.service
    ├── meta/
    │   └── main.yml          # Dependencies, metadata
    └── README.md
```

```
# roles/webapp/tasks/main.yml
---
- name: Install webapp
  import_tasks: install.yml
  tags: install

- name: Configure webapp
  import_tasks: configure.yml
  tags: configure
```

2.4 Ansible Galaxy

```
# Cài role từ Galaxy
ansible-galaxy install geerlingguy.docker
ansible-galaxy install -r requirements.yml

# requirements.yml
- name: geerlingguy.docker
  version: "6.1.0"
- src: https://github.com/company/ansible-role-webapp
  name: webapp
  version: main
```

2.5 Ansible Vault

```
# Encrypt file
ansible-vault encrypt vars/secrets.yml

# Encrypt string inline
ansible-vault encrypt_string 'mysecretpassword' --name 'db_password'

# Chạy playbook với vault
ansible-playbook deploy.yml --ask-vault-pass
ansible-playbook deploy.yml --vault-password-file ~/.vault_pass

# Rotate password
ansible-vault rekey vars/secrets.yml
```

```
# vars/secrets.yml (encrypted)
db_password: !vault |
  $ANSIBLE_VAULT;1.1;AES256
  66386439653236336462626566653063...
```

2.6 Jinja2 Templates

```
# templates/nginx.conf.j2
server {
    listen {{ app_port }};
    server_name {{ ansible_fqdn }};

    location / {
        proxy_pass http://127.0.0.1:{{ backend_port }};
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
```

```
    {% if ssl_enabled %}
    listen 443 ssl;
    ssl_certificate {{ ssl_cert_path }};
    ssl_certificate_key {{ ssl_key_path }};
    {% endif %}

    {% for path in blocked_paths %}
    location {{ path }} {
        deny all;
    }
    {% endfor %}
}
```

2.7 Modules Quan Trọng

```
# apt / yum - Package management
- apt:
    name: ["nginx", "curl"]
    state: present    # present, absent, latest

# copy - Copy files
- copy:
    src: files/app.conf
    dest: /etc/app/app.conf
    owner: root
    mode: '0644'
    backup: true

# template - Jinja2 template
- template:
    src: templates/config.j2
    dest: /etc/app/config.yml
    validate: /usr/bin/python3 -c "import yaml; yaml.safe_load(open('%s'))"

# service / systemd
- systemd:
    name: nginx
    state: started
    enabled: true
    daemon_reload: true

# docker_container - Docker management
- community.docker.docker_container:
    name: webapp
    image: "myapp:{{ app_version }}"
    state: started
    restart_policy: unless-stopped
```

```
    ports:
      - "8080:80"
    env:
      DATABASE_URL: "{{ db_url }}"

# lineinfile - Modify file lines
- lineinfile:
    path: /etc/ssh/sshd_config
    regexp: '^PasswordAuthentication'
    line: 'PasswordAuthentication no'

# command / shell
- command: /opt/app/migrate.sh
  args:
    chdir: /opt/app
    creates: /opt/app/.migrated  # Skip nêú file tôn tại
```

2.8 Best Practices Ansible

Directory Structure:

```
ansible/
├── inventories/
│   ├── production/
│   │   ├── hosts.ini
│   │   └── group_vars/
│   │       ├── all.yml
│   │       └── webservers.yml
│   └── staging/
│       └── hosts.ini
├── roles/
│   └── webapp/
├── playbooks/
│   ├── deploy.yml
│   └── maintenance.yml
├── ansible.cfg
└── requirements.yml
```

```
# ansible.cfg
[defaults]
inventory = inventories/production
roles_path = roles
stdout_callback = yaml
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /tmp/ansible_facts
fact_caching_timeout = 86400
```

```
[ssh_connection]
pipelining = true
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

Tips: - Luôn dùng `--check` (dry-run) trước khi chạy production - Dùng tags để chạy subset của tasks - Dùng `block/rescue/always` cho error handling - Dùng `when` conditions để control flow - Không hardcode passwords - dùng Vault

2.9 Ansible AWX / Tower

AWX là open-source version của Ansible Tower, cung cấp Web UI và REST API cho Ansible.

```
# Deploy AWX bằng Operator
kubectl apply -f awx-operator.yml

# Tạo AWX instance
cat <<EOF | kubectl apply -f -
apiVersion: awx.ansible.com/v1beta1
kind: AWX
metadata:
  name: awx-demo
spec:
  service_type: NodePort
  ingress_type: ingress
  hostname: awx.example.com
EOF
```

Tính năng AWX: - Role-based access control (RBAC) - Job scheduling và notifications - Credential management (không expose secrets) - Workflow jobs (chain multiple playbooks) - REST API cho automation

---

3. Chef

3.1 Khái Niệm

```
# Recipe: recipes/default.rb
package 'nginx' do
  action :install
end

service 'nginx' do
  action [:enable, :start]
end

template '/etc/nginx/nginx.conf' do
  source 'nginx.conf.erb'
```

```
  owner 'root'
  group 'root'
  mode '0644'
  notifies :reload, 'service[nginx]'
end
```

```
# Cookbook structure
cookbooks/webapp/
├── recipes/
│   ├── default.rb
│   └── database.rb
├── templates/
│   └── nginx.conf.erb
├── attributes/
│   └── default.rb      # Default attribute values
├── resources/
│   └── app_deploy.rb   # Custom resources (LWRP)
├── test/
│   └── integration/
└── metadata.rb
```

3.2 Data Bags

```
# Data bag cho secrets
knife data bag create secrets
knife data bag from file secrets db_credentials.json

# Encrypted data bag
knife data bag create --secret-file ~/.chef/secret secrets
```

```
# Dùng trong recipe
db_creds = data_bag_item('secrets', 'db_credentials')
db_password = db_creds['password']
```

---

4. Puppet

4.1 Manifests

```
# manifests/site.pp
node 'web1.example.com' {
  include profile::webserver
  include profile::monitoring
}
```

```
# modules/profile/manifests/webserver.pp
class profile::webserver {
  package { 'nginx':
    ensure => installed,
  }

  service { 'nginx':
    ensure  => running,
    enable  => true,
    require => Package['nginx'],
  }

  file { '/etc/nginx/nginx.conf':
    ensure  => file,
    content => template('profile/nginx.conf.erb'),
    notify  => Service['nginx'],
  }
}
```

4.2 Hiera

```
# hiera.yaml (data hierarchy)
version: 5
defaults:
  datadir: data
hierarchy:
  - name: "Node-specific data"
    path: "nodes/%{trusted.certname}.yaml"
  - name: "OS-specific data"
    path: "os/%{facts.os.family}.yaml"
  - name: "Common data"
    path: "common.yaml"
```

```
# data/common.yaml
profile::webserver::port: 80
profile::webserver::worker_processes: 4
```

---

5. So Sánh Ansible vs Chef vs Puppet vs Salt

Tiêu chí	Ansible	Chef	Puppet	Salt
Language	YAML/Jinja2	Ruby DSL	Puppet DSL	YAML/Jinja2
Agent	Agentless (SSH)	Agent required	Agent required	Agent/Agentless

Tiêu chí	Ansible	Chef	Puppet	Salt
Learning curve	Thấp	Cao	Trung bình	Trung bình
Push/Pull	Push	Pull	Pull	Push+Pull
Community	Rất lớn	Lớn	Lớn	Trung bình
Enterprise	Tower/AWX	Chef Automate	Puppet Enterprise	SaltStack
Ideal for	Simple automation	Complex app configs	Large enterprise	Real-time events

Lựa chọn thực tế: - **Ansible**: Team nhỏ, cần nhanh, infrastructure đơn giản - **Chef**: Dev team quen Ruby, cần flexibility cao - **Puppet**: Enterprise, hàng ngàn server, compliance - **Salt**: Cần event-driven automation, real-time response

---

6. Immutable Infrastructure

Khái niệm

Thay vì update server running, ta tạo image mới và replace hoàn toàn.

```
Mutable (truyền thống):
Server → apt-get upgrade → config changes → service restart

Immutable:
Code change → Packer build image → Deploy new instances → Terminate old
```

Packer Example

```
# packer/webapp.pkr.hcl
source "amazon-ebs" "webapp" {
  ami_name      = "webapp-${formatdate("YYYYMMDD-HHmm", timestamp())}"
  instance_type = "t3.medium"
  region        = "us-east-1"
  source_ami_filter {
    filters = {
      name                = "ubuntu/images/hvm-ssd/ubuntu-22.04-amd64-server-*"
      root-device-type    = "ebs"
      virtualization-type = "hvm"
    }
    owners      = ["099720109477"]
    most_recent = true
  }
  ssh_username = "ubuntu"
}

build {
```

```
  sources = ["source.amazon-ebs.webapp"]

  provisioner "ansible" {
    playbook_file = "ansible/playbooks/webapp.yml"
  }
}
```

---

7. Testing Configuration

Molecule (Ansible Testing)

```
# Setup
pip install molecule molecule-docker

# Tạo role với molecule
molecule init role myapp --driver-name docker

# Chạy test
molecule test          # Full test cycle
molecule converge      # Apply playbook
molecule verify        # Run verifier
molecule lint          # Lint checks
```

```
# molecule/default/molecule.yml
driver:
  name: docker
platforms:
  - name: instance
    image: "geerlingguy/docker-ubuntu2204-ansible"
    pre_build_image: true
provisioner:
  name: ansible
  playbooks:
    converge: converge.yml
verifier:
  name: ansible
```

Test Kitchen (Chef)

```
# .kitchen.yml
driver:
  name: docker
provisioner:
  name: chef_zero
```

```yaml
platforms:
  - name: ubuntu-22.04
suites:
  - name: default
    run_list:
      - recipe[webapp::default]
    verifier:
      inspec_tests:
        - test/integration/default
```

---

8. Best Practices Tổng Hợp

1. **Treat infrastructure as code**: Mọi thay đổi qua Pull Request
2. **Idempotency first**: Luôn test chạy nhiều lần cho cùng kết quả
3. **Least privilege**: Ansible user chỉ có quyền cần thiết
4. **Secret management**: Không bao giờ commit plaintext secrets
5. **Environment parity**: Dev, staging, production dùng cùng playbooks
6. **Test before apply**: Dùng `--check`, molecule, kitchen để test
7. **Documentation**: Comment phức tạp logic, giải thích "why" không chỉ "what"
8. **Modular design**: Roles/modules nhỏ, tái sử dụng được
9. **Pin versions**: Lock tool versions để đảm bảo reproducibility
10. **Monitoring integration**: Notify monitoring khi deploy (disable alerts, re-enable sau)

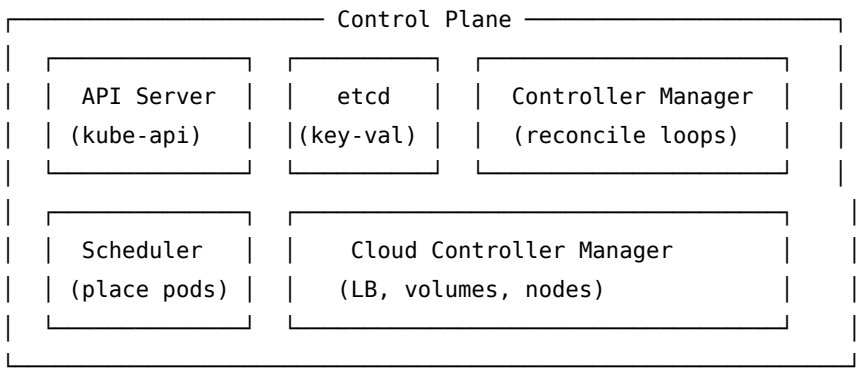
Chương 14: Kubernetes Mastery

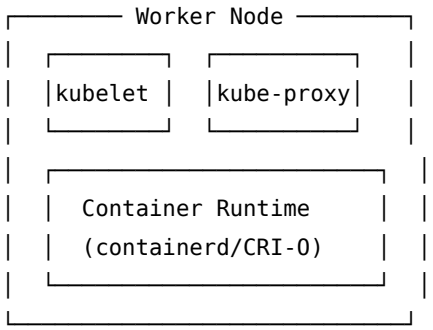
Kubernetes (K8s) là nền tảng orchestration container mạnh nhất hiện nay, quản lý deployment, scaling, và operations của containerized applications.

---

1. Kiến Trúc Kubernetes

Control Plane Components





- **API Server**: Gateway duy nhất, validate và process mọi request
- **etcd**: Distributed key-value store, lưu toàn bộ cluster state
- **Controller Manager**: Chạy controllers (ReplicaSet, Deployment, Node...)
- **Scheduler**: Quyết định pod chạy trên node nào dựa vào resources, affinity, taints
- **kubelet**: Agent trên node, đảm bảo containers chạy theo PodSpec
- **kube-proxy**: Quản lý network rules, implement Services

---

2. Workloads

Pod

```yaml
# pod.yml - Unit nhỏ nhất trong K8s
apiVersion: v1
kind: Pod
metadata:
  name: webapp
  labels:
    app: webapp
    version: "2.1"
spec:
  containers:
    - name: webapp
      image: myapp:2.1
      ports:
        - containerPort: 8080
      resources:
        requests:
          memory: "128Mi"
          cpu: "250m"
        limits:
          memory: "256Mi"
          cpu: "500m"
      livenessProbe:
        httpGet:
          path: /health
          port: 8080
```

```
        initialDelaySeconds: 30
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /ready
          port: 8080
        initialDelaySeconds: 5
        periodSeconds: 5
      env:
        - name: DB_HOST
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: host
  initContainers:
    - name: wait-for-db
      image: busybox
      command: ['sh', '-c', 'until nc -z db-service 5432; do sleep 2; done']
```

Deployment

```
# deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
  namespace: production
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: webapp
        version: "2.1"
    spec:
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
            - weight: 100
```

```
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                  - key: app
                    operator: In
                    values: [webapp]
              topologyKey: kubernetes.io/hostname
      containers:
        - name: webapp
          image: myapp:2.1
          # ... (same as pod spec)
```

StatefulSet

```
# statefulset.yml - Cho stateful apps (databases, queues)
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
spec:
  serviceName: postgres-headless
  replicas: 3
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:15
          volumeMounts:
            - name: data
              mountPath: /var/lib/postgresql/data
  volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        accessModes: ["ReadWriteOnce"]
        storageClassName: gp3
        resources:
          requests:
            storage: 20Gi
```

DaemonSet, Job, CronJob

```yaml
# daemonset.yml - Chạy một pod trên mỗi node
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-exporter
spec:
  selector:
    matchLabels:
      app: node-exporter
  template:
    spec:
      tolerations:
        - key: node-role.kubernetes.io/control-plane
          effect: NoSchedule
      containers:
        - name: node-exporter
          image: prom/node-exporter:latest

---
# cronjob.yml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: backup-job
spec:
  schedule: "0 2 * * *"     # 2 AM hàng ngày
  concurrencyPolicy: Forbid
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: OnFailure
          containers:
            - name: backup
              image: backup-tool:latest
              command: ["/scripts/backup.sh"]
```

---

3. Services

```yaml
# ClusterIP - Internal only (default)
apiVersion: v1
kind: Service
metadata:
```

```yaml
  name: webapp-service
spec:
  type: ClusterIP
  selector:
    app: webapp
  ports:
    - port: 80
      targetPort: 8080

---
# NodePort - Expose trên port của node
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 8080
      nodePort: 30080    # Range: 30000-32767

---
# LoadBalancer - Tạo cloud load balancer
spec:
  type: LoadBalancer
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
  ports:
    - port: 443
      targetPort: 8443
```

---

4. Ingress

```yaml
# ingress.yml - Nginx Ingress Controller
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webapp-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - app.example.com
      secretName: app-tls
```

```
  rules:
    - host: app.example.com
      http:
        paths:
          - path: /api
            pathType: Prefix
            backend:
              service:
                name: api-service
                port:
                  number: 80
          - path: /
            pathType: Prefix
            backend:
              service:
                name: frontend-service
                port:
                  number: 80
```

```
# Cài Nginx Ingress Controller
helm upgrade --install ingress-nginx ingress-nginx \
  --repo https://kubernetes.github.io/ingress-nginx \
  --namespace ingress-nginx --create-namespace

# Cài cert-manager
helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager --create-namespace \
  --set installCRDs=true
```

---

5. ConfigMaps và Secrets

```
# configmap.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  app.properties: |
    server.port=8080
    log.level=INFO
  DATABASE_HOST: "postgres-service"
  CACHE_TTL: "300"

---
# secret.yml
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
data:
  password: cGFzc3dvcmQxMjM=    # base64 encoded
  username: YWRtaW4=
stringData:
  connection-string: "postgresql://admin:password@db:5432/mydb"
```

```yaml
# Dùng trong pod
spec:
  containers:
    - name: app
      envFrom:
        - configMapRef:
            name: app-config
        - secretRef:
            name: db-secret
      volumeMounts:
        - name: config
          mountPath: /etc/config
  volumes:
    - name: config
      configMap:
        name: app-config
```

---

6. Storage

```yaml
# PersistentVolume (PV) - Admin tạo
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-data
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: gp3
  awsElasticBlockStore:
    volumeID: vol-0abc123
    fsType: ext4
```

```yaml
---
# PersistentVolumeClaim (PVC) - Developer tạo
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gp3
  resources:
    requests:
      storage: 20Gi

---
# StorageClass - Dynamic provisioning
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp3
provisioner: ebs.csi.aws.com
parameters:
  type: gp3
  iops: "3000"
  throughput: "125"
reclaimPolicy: Delete
allowVolumeExpansion: true
```

---

7. RBAC

```yaml
# serviceaccount.yml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: webapp-sa
  namespace: production

---
# role.yml - Namespace scoped
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: webapp-role
  namespace: production
```

```yaml
rules:
  - apiGroups: [""]
    resources: ["configmaps", "secrets"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["apps"]
    resources: ["deployments"]
    verbs: ["get", "list", "patch", "update"]

---
# rolebinding.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: webapp-rolebinding
  namespace: production
subjects:
  - kind: ServiceAccount
    name: webapp-sa
    namespace: production
roleRef:
  kind: Role
  name: webapp-role
  apiGroup: rbac.authorization.k8s.io
```

```bash
# Kiểm tra permissions
kubectl auth can-i create deployments --as=system:serviceaccount:production:webapp-sa
kubectl auth can-i get secrets -n production --as=user@company.com
```

---

8. Networking

CNI Plugins

- **Flannel**: Đơn giản, overlay network, phù hợp cho development
- **Calico**: Network policy mạnh, high performance, phổ biến production
- **Cilium**: eBPF-based, observability cao, Layer 7 policies

NetworkPolicy

```yaml
# Chỉ cho phép webapp traffic từ frontend và monitoring
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: webapp-netpol
  namespace: production
spec:
```

```yaml
  podSelector:
    matchLabels:
      app: webapp
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              name: frontend
        - podSelector:
            matchLabels:
              app: prometheus
      ports:
        - protocol: TCP
          port: 8080
  egress:
    - to:
        - podSelector:
            matchLabels:
              app: postgres
      ports:
        - port: 5432
    - to:    # Allow DNS
        - namespaceSelector: {}
      ports:
        - port: 53
          protocol: UDP
```

---

9. Helm

```bash
# Helm basics
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
helm search repo bitnami/postgresql

# Install / Upgrade
helm install my-postgres bitnami/postgresql \
  --namespace database --create-namespace \
  --set primary.persistence.size=50Gi \
  --values custom-values.yml

helm upgrade my-postgres bitnami/postgresql --values custom-values.yml
helm rollback my-postgres 1
```

```
# List và status
helm list -A
helm status my-postgres -n database
helm history my-postgres -n database
```

```
# Chart structure
my-app/
├── Chart.yaml
├── values.yaml          # Default values
├── values-prod.yaml     # Production overrides
├── templates/
│   ├── deployment.yaml
│   ├── service.yaml
│   ├── ingress.yaml
│   ├── _helpers.tpl     # Template helpers
│   └── NOTES.txt
└── charts/              # Dependencies
```

```
# templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "my-app.fullname" . }}
  labels:
    {{- include "my-app.labels" . | nindent 4 }}
spec:
  replicas: {{ .Values.replicaCount }}
  {{- with .Values.nodeSelector }}
  nodeSelector:
    {{- toYaml . | nindent 8 }}
  {{- end }}
```

---

10. Resource Management

```
# HPA - Horizontal Pod Autoscaler
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: webapp-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: webapp
```

```yaml
  minReplicas: 3
  maxReplicas: 20
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
    - type: Resource
      resource:
        name: memory
        target:
          type: Utilization
          averageUtilization: 80

---
# PodDisruptionBudget - Đảm bảo availability khi drain node
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: webapp-pdb
spec:
  minAvailable: 2    # Hoặc dùng maxUnavailable: 1
  selector:
    matchLabels:
      app: webapp
```

---

11. Troubleshooting

```bash
# Xem events của namespace
kubectl get events -n production --sort-by='.lastTimestamp'

# Debug pod
kubectl describe pod webapp-xyz -n production
kubectl logs webapp-xyz -n production --previous  # Log của container đã crash
kubectl logs webapp-xyz -n production -c sidecar  # Log của specific container

# Exec vào pod
kubectl exec -it webapp-xyz -n production -- /bin/sh

# Debug với ephemeral container (K8s 1.23+)
kubectl debug -it webapp-xyz --image=busybox --target=webapp

# Port forward để test locally
```

```
kubectl port-forward svc/webapp-service 8080:80 -n production

# Copy files từ/vào pod
kubectl cp production/webapp-xyz:/app/logs/error.log ./error.log

# Xem resource usage
kubectl top pods -n production
kubectl top nodes

# Check RBAC issues
kubectl auth can-i list pods --as=system:serviceaccount:production:webapp-sa

# Node issues
kubectl cordon node-1      # Ngăn schedule pod mới
kubectl drain node-1 --ignore-daemonsets --delete-emptydir-data
kubectl uncordon node-1
```

---

12. Production Best Practices

Namespace Strategy

```
# Tạo namespace với quotas
kubectl create namespace production
kubectl create namespace staging
kubectl create namespace monitoring
```

```
# ResourceQuota cho namespace
apiVersion: v1
kind: ResourceQuota
metadata:
  name: production-quota
  namespace: production
spec:
  hard:
    requests.cpu: "20"
    requests.memory: 40Gi
    limits.cpu: "40"
    limits.memory: 80Gi
    pods: "100"
    services: "20"

---
# LimitRange - Default resource limits
apiVersion: v1
kind: LimitRange
```

```yaml
metadata:
  name: default-limits
  namespace: production
spec:
  limits:
    - default:
        cpu: 500m
        memory: 256Mi
      defaultRequest:
        cpu: 100m
        memory: 128Mi
      type: Container
```

Pod Security

```yaml
# Pod Security Standards (K8s 1.25+)
# Label namespace thay vì PodSecurityPolicy (deprecated)
kubectl label namespace production \
  pod-security.kubernetes.io/enforce=restricted \
  pod-security.kubernetes.io/warn=restricted

# Trong pod spec
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    fsGroup: 2000
    seccompProfile:
      type: RuntimeDefault
  containers:
    - securityContext:
        allowPrivilegeEscalation: false
        readOnlyRootFilesystem: true
        capabilities:
          drop: ["ALL"]
```

---

13. Ví Dụ: Deploy Microservices App

```yaml
# namespace.yml
apiVersion: v1
kind: Namespace
metadata:
  name: ecommerce
  labels:
```

```yaml
    pod-security.kubernetes.io/enforce: baseline

---
# frontend deployment + service
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: ecommerce
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: ecommerce/frontend:1.5.0
          ports:
            - containerPort: 3000
          resources:
            requests: { cpu: "100m", memory: "128Mi" }
            limits: { cpu: "500m", memory: "256Mi" }
---
apiVersion: v1
kind: Service
metadata:
  name: frontend-svc
  namespace: ecommerce
spec:
  selector:
    app: frontend
  ports:
    - port: 80
      targetPort: 3000

---
# Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ecommerce-ingress
  namespace: ecommerce
  annotations:
```

```
    cert-manager.io/cluster-issuer: letsencrypt-prod
spec:
  ingressClassName: nginx
  tls:
    - hosts: [shop.example.com]
      secretName: ecommerce-tls
  rules:
    - host: shop.example.com
      http:
        paths:
          - path: /api
            pathType: Prefix
            backend:
              service:
                name: api-svc
                port:
                  number: 80
          - path: /
            pathType: Prefix
            backend:
              service:
                name: frontend-svc
                port:
                  number: 80
```

```
# Deploy toàn bộ
kubectl apply -f manifests/ --recursive -n ecommerce

# Kiểm tra
kubectl get all -n ecommerce
kubectl get ingress -n ecommerce
kubectl rollout status deployment/frontend -n ecommerce

# Rollback nếu có vấn đề
kubectl rollout undo deployment/frontend -n ecommerce
```

Chương 15: Infrastructure as Code (IaC)

Infrastructure as Code là thực hành quản lý và provision hạ tầng thông qua code thay vì quy trình thủ công. Mọi thứ từ network, compute, storage đến DNS đều được định nghĩa trong file code có thể version control.

---

1. Terraform

Terraform là tool IaC phổ biến nhất, dùng HashiCorp Configuration Language (HCL).

1.1 HCL Syntax

```
# main.tf - Cú pháp cơ bản

# Provider - Plugin kết nối với cloud/service
terraform {
  required_version = ">= 1.6.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "~> 2.23"
    }
  }
}

provider "aws" {
  region = var.aws_region
  default_tags {
    tags = {
      Project     = var.project_name
      Environment = var.environment
      ManagedBy   = "terraform"
    }
  }
}

# Variables
variable "aws_region" {
  description = "AWS region to deploy resources"
  type        = string
  default     = "us-east-1"
}

variable "environment" {
  description = "Environment name"
  type        = string
  validation {
    condition     = contains(["dev", "staging", "production"], var.environment)
    error_message = "Environment must be dev, staging, or production."
  }
}

variable "instance_config" {
  description = "EC2 instance configuration"
```

```
  type = object({
    type  = string
    count = number
  })
  default = {
    type  = "t3.medium"
    count = 2
  }
}

# Locals - Computed values
locals {
  name_prefix = "${var.project_name}-${var.environment}"
  common_tags = {
    Project     = var.project_name
    Environment = var.environment
  }
  azs = slice(data.aws_availability_zones.available.names, 0, 3)
}

# Data Sources - Query existing resources
data "aws_availability_zones" "available" {
  state = "available"
}

data "aws_ami" "ubuntu" {
  most_recent = true
  owners      = ["099720109477"]    # Canonical
  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-22.04-amd64-server-*"]
  }
}

# Resources
resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = merge(local.common_tags, {
    Name = "${local.name_prefix}-vpc"
  })
}

resource "aws_instance" "web" {
  count         = var.instance_config.count
  ami           = data.aws_ami.ubuntu.id
```

```
  instance_type = var.instance_config.type

  tags = {
    Name = "${local.name_prefix}-web-${count.index + 1}"
  }
}

# For_each - Preferred over count cho map/set
resource "aws_subnet" "private" {
  for_each = {
    for idx, az in local.azs : az => {
      cidr = cidrsubnet("10.0.0.0/16", 8, idx + 10)
      az   = az
    }
  }

  vpc_id            = aws_vpc.main.id
  cidr_block        = each.value.cidr
  availability_zone = each.value.az

  tags = {
    Name = "${local.name_prefix}-private-${each.key}"
    Type = "private"
  }
}

# Outputs
output "vpc_id" {
  description = "VPC ID"
  value       = aws_vpc.main.id
}

output "private_subnet_ids" {
  description = "List of private subnet IDs"
  value       = values(aws_subnet.private)[*].id
}
```

1.2 State Management

```
# backend.tf - Remote state (S3 + DynamoDB locking)
terraform {
  backend "s3" {
    bucket         = "company-terraform-state"
    key            = "production/vpc/terraform.tfstate"
    region         = "us-east-1"
    encrypt        = true
    kms_key_id     = "arn:aws:kms:us-east-1:123456789:key/abc-123"
```

```
    dynamodb_table = "terraform-state-lock"
  }
}
```

```
# State commands
terraform state list                              # Liệt kê resources trong state
terraform state show aws_vpc.main                 # Chi tiết một resource
terraform state mv aws_instance.web aws_instance.app   # Rename resource
terraform state rm aws_instance.web               # Remove khỏi state (không xóa resource)

# Import existing resource vào state
terraform import aws_vpc.main vpc-0abc123def456
terraform import 'aws_subnet.private["us-east-1a"]' subnet-0abc123

# State locking
terraform force-unlock LOCK_ID    # Dùng khi lock bị stuck
```

1.3 Modules

```
# modules/vpc/main.tf
variable "vpc_cidr" { type = string }
variable "environment" { type = string }
variable "availability_zones" { type = list(string) }

resource "aws_vpc" "this" {
  cidr_block = var.vpc_cidr
  tags = { Name = "${var.environment}-vpc" }
}

resource "aws_internet_gateway" "this" {
  vpc_id = aws_vpc.this.id
}

output "vpc_id" { value = aws_vpc.this.id }
output "public_subnet_ids" { value = aws_subnet.public[*].id }
```

```
# Dùng module
module "vpc" {
  source = "./modules/vpc"
  # Hoặc từ Terraform Registry
  # source  = "terraform-aws-modules/vpc/aws"
  # version = "~> 5.0"

  vpc_cidr           = "10.0.0.0/16"
  environment        = var.environment
  availability_zones = local.azs
}
```

```
module "eks" {
  source = "./modules/eks"
  vpc_id          = module.vpc.vpc_id
  subnet_ids      = module.vpc.private_subnet_ids
  cluster_version = "1.28"
}
```

1.4 Workspaces

```
# Terraform workspaces - Quản lý multiple environments
terraform workspace list
terraform workspace new staging
terraform workspace select production
terraform workspace show

# Dùng trong code
resource "aws_instance" "web" {
  instance_type = terraform.workspace == "production" ? "t3.large" : "t3.micro"
}

# Tốt hơn: Dùng .tfvars files thay workspace
terraform plan -var-file="environments/production.tfvars"
terraform apply -var-file="environments/staging.tfvars"
```

1.5 Best Practices Terraform

Directory Structure:

```
infrastructure/
├── modules/
│   ├── vpc/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── outputs.tf
│   │   └── README.md
│   ├── eks/
│   └── rds/
├── environments/
│   ├── production/
│   │   ├── main.tf          # Module calls
│   │   ├── variables.tf
│   │   ├── outputs.tf
│   │   ├── backend.tf
│   │   └── terraform.tfvars
│   └── staging/
├── .terraform-version        # tfenv version pin
```

```
└── Makefile
```

```
# Makefile cho Terraform workflows
ENV ?= staging

init:
	terraform -chdir=environments/$(ENV) init

plan:
	terraform -chdir=environments/$(ENV) plan -out=tfplan

apply:
	terraform -chdir=environments/$(ENV) apply tfplan

destroy:
	terraform -chdir=environments/$(ENV) destroy

fmt:
	terraform fmt -recursive

validate:
	terraform -chdir=environments/$(ENV) validate
```

Naming Conventions:

```
# Resource naming: {project}-{environment}-{service}-{type}
name = "${var.project}-${var.environment}-webapp-sg"

# Variable naming: snake_case, descriptive
variable "max_instance_count" {}     # Tốt
variable "n" {}                      # Tệ

# Output naming: rõ ràng về type
output "alb_dns_name" {}             # Tốt
output "dns" {}                      # Tệ
```

---

2. Pulumi

Pulumi cho phép viết IaC bằng programming languages quen thuộc.

2.1 Python Example

```
# __main__.py
import pulumi
import pulumi_aws as aws
```

```
config = pulumi.Config()
environment = config.require("environment")
project = pulumi.get_project()

# VPC
vpc = aws.ec2.Vpc("main-vpc",
    cidr_block="10.0.0.0/16",
    enable_dns_hostnames=True,
    tags={"Environment": environment, "ManagedBy": "pulumi"}
)

# Subnets với list comprehension
availability_zones = aws.get_availability_zones(state="available")

private_subnets = [
    aws.ec2.Subnet(f"private-subnet-{i}",
        vpc_id=vpc.id,
        cidr_block=f"10.0.{i+10}.0/24",
        availability_zone=availability_zones.names[i],
        tags={"Name": f"{project}-private-{i}", "Type": "private"}
    )
    for i in range(3)
]

# EKS Cluster
eks_role = aws.iam.Role("eks-role",
    assume_role_policy=aws.iam.get_policy_document(statements=[{
        "actions": ["sts:AssumeRole"],
        "principals": [{"type": "Service", "identifiers": ["eks.amazonaws.com"]}]
    }]).json
)

cluster = aws.eks.Cluster("main-cluster",
    role_arn=eks_role.arn,
    version="1.28",
    vpc_config=aws.eks.ClusterVpcConfigArgs(
        subnet_ids=[s.id for s in private_subnets]
    )
)

pulumi.export("vpc_id", vpc.id)
pulumi.export("cluster_name", cluster.name)
pulumi.export("kubeconfig", cluster.certificate_authority)
```

2.2 TypeScript Example

```typescript
// index.ts
import * as pulumi from "@pulumi/pulumi";
import * as aws from "@pulumi/aws";
import * as k8s from "@pulumi/kubernetes";

const config = new pulumi.Config();
const environment = config.require("environment");

// Stack references - lấy output từ stack khác
const networkStack = new pulumi.StackReference(`company/network/${environment}`);
const vpcId = networkStack.getOutput("vpcId");
const subnetIds = networkStack.getOutput("privateSubnetIds");

// EKS Cluster
const cluster = new aws.eks.Cluster("eks", {
    version: "1.28",
    vpcConfig: { subnetIds: subnetIds as pulumi.Output<string[]> },
});

// Deploy Helm chart sau khi EKS ready
const k8sProvider = new k8s.Provider("k8s", {
    kubeconfig: cluster.kubeconfig,
});

const nginxIngress = new k8s.helm.v3.Release("nginx-ingress", {
    chart: "ingress-nginx",
    repositoryOpts: { repo: "https://kubernetes.github.io/ingress-nginx" },
    namespace: "ingress-nginx",
    createNamespace: true,
}, { provider: k8sProvider });

export const clusterName = cluster.name;
export const clusterEndpoint = cluster.endpoint;
```

---

3. So Sánh IaC Tools

Tiêu chí	Terraform	Pulumi	CloudFormation	CDK
Language	HCL	Python/Go/TS/.	JSON/YAML	Python/TS/Java
Provider support	3000+	100+	AWS only	AWS only
State management	Remote backend	Pulumi Cloud/S3	S3 (managed)	S3 (managed)

Tiêu chí	Terraform	Pulumi	CloudFormation	CDK
Learning curve	Trung bình	Thấp (biết PL)	Cao (verbose)	Trung bình
Testing	terratest	jest/pytest	CDK assertions	CDK assertions
Drift detection	terraform plan	pulumi refresh	Cloud Control	drift-detect
Free	Tất cả	Self-hosted free	Free	Free
Best for	Multi-cloud	Dev-centric teams	AWS-native	AWS + type safety

---

4. GitOps cho IaC

Pull Request Workflow

```yaml
# .github/workflows/terraform.yml
name: Terraform CI/CD

on:
  pull_request:
    paths: ["infrastructure/**"]
  push:
    branches: [main]
    paths: ["infrastructure/**"]

jobs:
  validate:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: hashicorp/setup-terraform@v3
        with:
          terraform_version: "1.6.0"

      - name: Terraform Format Check
        run: terraform fmt -check -recursive infrastructure/

      - name: Terraform Validate
        run: |
          cd infrastructure/environments/production
          terraform init -backend=false
          terraform validate
```

```yaml
      - name: TFLint
        uses: terraform-linters/setup-tflint@v4
        run: tflint --recursive

      - name: Checkov Security Scan
        uses: bridgecrewio/checkov-action@master
        with:
          directory: infrastructure/
          framework: terraform

  plan:
    needs: validate
    runs-on: ubuntu-latest
    if: github.event_name == 'pull_request'
    permissions:
      id-token: write       # OIDC
      pull-requests: write  # PR comments
    steps:
      - uses: actions/checkout@v4
      - name: Configure AWS via OIDC
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::123456789:role/terraform-plan
          aws-region: us-east-1

      - name: Terraform Plan
        id: plan
        run: |
          cd infrastructure/environments/production
          terraform init
          terraform plan -out=tfplan -no-color 2>&1 | tee plan_output.txt

      - name: Comment Plan on PR
        uses: actions/github-script@v7
        with:
          script: |
            const fs = require('fs');
            const plan = fs.readFileSync('infrastructure/environments/production/plan_output.txt',
↪  'utf8');
            github.rest.issues.createComment({
              issue_number: context.issue.number,
              owner: context.repo.owner,
              repo: context.repo.repo,
              body: `## Terraform Plan\n\`\`\`\n${plan.slice(-50000)}\n\`\`\``
            });

      - name: Infracost Estimate
        uses: infracost/actions/setup@v2
```

```
        with:
          api-key: ${{ secrets.INFRACOST_API_KEY }}
        run: |
          infracost breakdown --path=infrastructure/environments/production \
            --format=json --out-file=/tmp/infracost.json
          infracost comment github --path=/tmp/infracost.json \
            --repo=$GITHUB_REPOSITORY --pull-request=$PR_NUMBER

  apply:
    needs: plan
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    environment: production    # Require manual approval
    steps:
      - name: Terraform Apply
        run: |
          cd infrastructure/environments/production
          terraform apply -auto-approve tfplan
```

---

5. Drift Detection

```
# Terraform - Kiểm tra drift
terraform plan -refresh-only    # Chỉ refresh state, không thay đổi
terraform apply -refresh-only   # Apply state refresh

# Tự động drift detection với Lambda (schedule hàng ngày)
# Script check drift
#!/bin/bash
cd infrastructure/environments/production
terraform init -backend-config=backend.hcl
terraform plan -detailed-exitcode -refresh-only
EXIT_CODE=$?

if [ $EXIT_CODE -eq 2 ]; then
    echo "DRIFT DETECTED!"
    # Send alert to Slack/PagerDuty
    curl -X POST -H 'Content-type: application/json' \
      --data '{"text":"⚠ Terraform drift detected in production!"}' \
      $SLACK_WEBHOOK_URL
fi
```

---

6. Testing IaC

Terratest (Go)

```go
// test/vpc_test.go
package test

import (
    "testing"
    "github.com/gruntwork-io/terratest/modules/terraform"
    "github.com/gruntwork-io/terratest/modules/aws"
    "github.com/stretchr/testify/assert"
)

func TestVPCModule(t *testing.T) {
    t.Parallel()

    terraformOptions := &terraform.Options{
        TerraformDir: "../modules/vpc",
        Vars: map[string]interface{}{
            "vpc_cidr":    "10.99.0.0/16",
            "environment": "test",
        },
    }

    defer terraform.Destroy(t, terraformOptions)
    terraform.InitAndApply(t, terraformOptions)

    vpcID := terraform.Output(t, terraformOptions, "vpc_id")
    assert.NotEmpty(t, vpcID)

    // Verify VPC exists in AWS
    vpc := aws.GetVpcById(t, vpcID, "us-east-1")
    assert.Equal(t, "10.99.0.0/16", aws.GetCidrBlockOfVpc(t, vpc))
}
```

Checkov (Security Scanning)

```bash
# Scan Terraform code
checkov -d infrastructure/ --framework terraform
checkov -d infrastructure/ --check CKV_AWS_111,CKV_AWS_119

# Scan với custom config
checkov -d . --config-file .checkov.yaml

# Output formats
checkov -d . --output json --output-file-path ./reports/
```

```
# .checkov.yaml
skip-check:
  - CKV_AWS_144    # S3 cross-region replication (intentional)
  - CKV2_AWS_6     # S3 public access (intentional for static site)
compact: true
output: cli
```

---

7. Ví Dụ: AWS VPC + EKS Module

```
# modules/eks-cluster/main.tf

# Data sources
data "aws_eks_cluster_auth" "cluster" {
  name = aws_eks_cluster.this.name
}

# EKS Cluster IAM Role
resource "aws_iam_role" "cluster" {
  name = "${var.cluster_name}-cluster-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action    = "sts:AssumeRole"
      Effect    = "Allow"
      Principal = { Service = "eks.amazonaws.com" }
    }]
  })
}

resource "aws_iam_role_policy_attachment" "cluster_policies" {
  for_each = toset([
    "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy",
    "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
  ])
  policy_arn = each.value
  role       = aws_iam_role.cluster.name
}

# EKS Cluster
resource "aws_eks_cluster" "this" {
  name     = var.cluster_name
  version  = var.kubernetes_version
  role_arn = aws_iam_role.cluster.arn

  vpc_config {
```

```
    subnet_ids              = var.private_subnet_ids
    endpoint_private_access = true
    endpoint_public_access  = var.public_endpoint
    public_access_cidrs     = var.allowed_cidr_blocks
  }

  enabled_cluster_log_types = ["api", "audit", "authenticator"]

  encryption_config {
    provider {
      key_arn = aws_kms_key.eks.arn
    }
    resources = ["secrets"]
  }

  depends_on = [aws_iam_role_policy_attachment.cluster_policies]
}

# Node Group
resource "aws_eks_node_group" "main" {
  cluster_name    = aws_eks_cluster.this.name
  node_group_name = "${var.cluster_name}-ng-main"
  node_role_arn   = aws_iam_role.node.arn
  subnet_ids      = var.private_subnet_ids

  instance_types = var.node_instance_types
  capacity_type  = "ON_DEMAND"

  scaling_config {
    desired_size = var.node_desired_count
    min_size     = var.node_min_count
    max_size     = var.node_max_count
  }

  update_config {
    max_unavailable_percentage = 25
  }

  labels = {
    role        = "general"
    environment = var.environment
  }

  lifecycle {
    ignore_changes = [scaling_config[0].desired_size]
  }
}
```

```
# KMS cho Secrets encryption
resource "aws_kms_key" "eks" {
  description             = "EKS Secret Encryption Key"
  deletion_window_in_days = 7
  enable_key_rotation     = true
}

# Outputs
output "cluster_endpoint" { value = aws_eks_cluster.this.endpoint }
output "cluster_name"     { value = aws_eks_cluster.this.name }
output "cluster_ca_data"  { value = aws_eks_cluster.this.certificate_authority[0].data }
output "kubeconfig_command" {
  value = "aws eks update-kubeconfig --name ${aws_eks_cluster.this.name} --region ${var.region}"
}
```

```
# environments/production/main.tf - Gọi modules

module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "~> 5.0"

  name = "${local.name_prefix}-vpc"
  cidr = "10.0.0.0/16"

  azs             = local.azs
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]

  enable_nat_gateway     = true
  single_nat_gateway     = false    # HA: one per AZ
  enable_dns_hostnames   = true

  # EKS tags required
  private_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb"             = "1"
  }
  public_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/elb"                      = "1"
  }
}

module "eks" {
  source = "../../modules/eks-cluster"

  cluster_name        = local.cluster_name
  kubernetes_version  = "1.28"
```

```
  environment         = var.environment
  region              = var.aws_region

  private_subnet_ids  = module.vpc.private_subnets
  public_endpoint     = false    # Private cluster
  allowed_cidr_blocks = ["10.0.0.0/8"]

  node_instance_types = ["m5.xlarge", "m5a.xlarge"]
  node_desired_count  = 3
  node_min_count      = 2
  node_max_count      = 10
}
```

Chương 16: CI/CD

CI/CD (Continuous Integration / Continuous Delivery / Deployment) là xương sống của DevOps, tự động hóa toàn bộ quá trình từ code commit đến production deployment.

---

1. Khái Niệm Cốt Lõi

Continuous Integration (CI)

Developer merge code thường xuyên (nhiều lần/ngày) vào shared branch. Mỗi commit trigger automated build và test.

Continuous Delivery (CD)

Code luôn ở trạng thái sẵn sàng deploy. Release lên production là quyết định kinh doanh, không phải kỹ thuật.

Continuous Deployment

Mọi commit pass CI tự động được deploy lên production, không cần manual approval.

```
Code Commit → Build → Unit Test → Integration Test → Security Scan
     → Staging Deploy → Smoke Test → [Approval] → Production Deploy
```

---

2. Jenkins

2.1 Declarative Pipeline

```
// Jenkinsfile
pipeline {
```

```
    agent {
        kubernetes {
            yaml '''
                apiVersion: v1
                kind: Pod
                spec:
                  containers:
                  - name: maven
                    image: maven:3.9-eclipse-temurin-17
                    command: [sleep, infinity]
                  - name: docker
                    image: docker:24-dind
                    securityContext:
                      privileged: true
            '''
        }
    }

    environment {
        APP_NAME    = 'webapp'
        REGISTRY    = 'registry.company.com'
        IMAGE_TAG   = "${env.GIT_COMMIT[0..7]}"
        KUBECONFIG  = credentials('kubeconfig-production')
    }

    options {
        timeout(time: 30, unit: 'MINUTES')
        disableConcurrentBuilds()
        buildDiscarder(logRotator(numToKeepStr: '20'))
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
                script {
                    env.VERSION = sh(script: 'cat VERSION', returnStdout: true).trim()
                }
            }
        }

        stage('Build & Test') {
            parallel {
                stage('Unit Tests') {
                    steps {
                        container('maven') {
                            sh 'mvn test -pl webapp-core'
                        }
```

```
                }
                post {
                    always {
                        junit 'webapp-core/target/surefire-reports/*.xml'
                        jacoco(execPattern: '**/target/jacoco.exec',
                               minimumLineCoverage: '80')
                    }
                }
            }
            stage('Lint & SAST') {
                steps {
                    container('maven') {
                        sh 'mvn checkstyle:check pmd:check'
                        sh 'mvn spotbugs:check'
                    }
                }
            }
        }
    }

    stage('Build Docker Image') {
        steps {
            container('docker') {
                sh """
                    docker build \
                      --build-arg VERSION=${env.VERSION} \
                      --build-arg GIT_COMMIT=${env.IMAGE_TAG} \
                      --cache-from ${REGISTRY}/${APP_NAME}:latest \
                      -t ${REGISTRY}/${APP_NAME}:${IMAGE_TAG} \
                      -t ${REGISTRY}/${APP_NAME}:latest .
                """
                withCredentials([usernamePassword(
                    credentialsId: 'registry-credentials',
                    usernameVariable: 'USER',
                    passwordVariable: 'PASS'
                )]) {
                    sh "docker login ${REGISTRY} -u ${USER} -p ${PASS}"
                    sh "docker push ${REGISTRY}/${APP_NAME}:${IMAGE_TAG}"
                    sh "docker push ${REGISTRY}/${APP_NAME}:latest"
                }
            }
        }
    }

    stage('Deploy to Staging') {
        steps {
            sh """
                kubectl set image deployment/webapp \
```

```
                    webapp=${REGISTRY}/${APP_NAME}:${IMAGE_TAG} \
                    -n staging
                kubectl rollout status deployment/webapp -n staging --timeout=5m
            """
        }
    }

    stage('Integration Tests') {
        steps {
            sh 'mvn verify -pl integration-tests -Dbase.url=https://staging.example.com'
        }
    }

    stage('Deploy to Production') {
        when {
            branch 'main'
        }
        input {
            message "Deploy ${env.VERSION} to production?"
            ok "Deploy"
            submitter "senior-dev,tech-lead"
        }
        steps {
            sh """
                kubectl set image deployment/webapp \
                  webapp=${REGISTRY}/${APP_NAME}:${IMAGE_TAG} \
                  -n production
                kubectl rollout status deployment/webapp -n production --timeout=10m
            """
        }
    }
}

post {
    failure {
        slackSend(
            channel: '#deployments',
            color: 'danger',
            message: "Pipeline FAILED: ${env.JOB_NAME} #${env.BUILD_NUMBER}\n${env.BUILD_URL}"
        )
    }
    success {
        slackSend(
            channel: '#deployments',
            color: 'good',
            message: "Deployed ${APP_NAME}:${env.VERSION} to production"
        )
    }
```

```
    }
}
```

2.2 Shared Libraries

```
// vars/deployToK8s.groovy (trong shared library repo)
def call(Map config = [:]) {
    def namespace   = config.namespace   ?: 'default'
    def deployment  = config.deployment  ?: error('deployment required')
    def image       = config.image       ?: error('image required')
    def timeout     = config.timeout     ?: '5m'

    sh """
        kubectl set image deployment/${deployment} \
          ${deployment}=${image} -n ${namespace}
        kubectl rollout status deployment/${deployment} \
          -n ${namespace} --timeout=${timeout}
    """
}

// Jenkinsfile dùng shared library
@Library('jenkins-shared-libs@main') _

pipeline {
    stages {
        stage('Deploy') {
            steps {
                deployToK8s(
                    namespace:  'production',
                    deployment: 'webapp',
                    image:      "registry.company.com/webapp:${env.IMAGE_TAG}"
                )
            }
        }
    }
}
```

---

3. GitHub Actions

3.1 Complete Workflow

```
# .github/workflows/ci-cd.yml
name: CI/CD Pipeline
```

```yaml
on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]
  workflow_dispatch:
    inputs:
      environment:
        description: 'Target environment'
        required: true
        type: choice
        options: [staging, production]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [18, 20, 22]    # Matrix build
    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js ${{ matrix.node-version }}
        uses: actions/setup-node@v4
        with:
          node-version: ${{ matrix.node-version }}
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Lint
        run: npm run lint

      - name: Test with coverage
        run: npm test -- --coverage

      - name: Upload coverage
        uses: codecov/codecov-action@v4
        with:
          token: ${{ secrets.CODECOV_TOKEN }}
          fail_ci_if_error: true

  security:
```

```yaml
    runs-on: ubuntu-latest
    permissions:
      security-events: write
    steps:
      - uses: actions/checkout@v4

      - name: Run Trivy vulnerability scanner
        uses: aquasecurity/trivy-action@master
        with:
          scan-type: 'fs'
          scan-ref: '.'
          format: 'sarif'
          output: 'trivy-results.sarif'

      - name: Upload Trivy results to GitHub Security
        uses: github/codeql-action/upload-sarif@v3
        with:
          sarif_file: 'trivy-results.sarif'

      - name: Dependency audit
        run: npm audit --audit-level=high

  build:
    needs: [test, security]
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
      id-token: write    # For OIDC signing
    outputs:
      image-digest: ${{ steps.build.outputs.digest }}
      image-tag: ${{ steps.meta.outputs.tags }}
    steps:
      - uses: actions/checkout@v4

      - name: Docker meta
        id: meta
        uses: docker/metadata-action@v5
        with:
          images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
          tags: |
            type=ref,event=branch
            type=ref,event=pr
            type=sha,prefix=sha-
            type=semver,pattern={{version}}
            type=raw,value=latest,enable=${{ github.ref == 'refs/heads/main' }}

      - name: Setup QEMU (multi-arch)
```

```yaml
      uses: docker/setup-qemu-action@v3

    - name: Setup Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Login to registry
      uses: docker/login-action@v3
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Build and push
      id: build
      uses: docker/build-push-action@v5
      with:
        context: .
        platforms: linux/amd64,linux/arm64
        push: true
        tags: ${{ steps.meta.outputs.tags }}
        labels: ${{ steps.meta.outputs.labels }}
        cache-from: type=gha
        cache-to: type=gha,mode=max
        provenance: true    # SLSA provenance
        sbom: true          # Software Bill of Materials

    - name: Sign image with Cosign (OIDC)
      uses: sigstore/cosign-installer@v3
      run: |
        cosign sign --yes \
          ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}@${{ steps.build.outputs.digest }}

deploy-staging:
  needs: build
  runs-on: ubuntu-latest
  environment:
    name: staging
    url: https://staging.example.com
  steps:
    - uses: actions/checkout@v4

    - name: Configure AWS credentials (OIDC)
      uses: aws-actions/configure-aws-credentials@v4
      with:
        role-to-assume: arn:aws:iam::123456789:role/github-actions-deploy
        aws-region: us-east-1

    - name: Update kubeconfig
```

```
        run: aws eks update-kubeconfig --name my-cluster --region us-east-1

      - name: Deploy to staging
        run: |
          IMAGE="${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}@${{ needs.build.outputs.image-digest }}"
          helm upgrade --install webapp ./charts/webapp \
            --namespace staging \
            --set image.repository="${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}" \
            --set image.digest="${{ needs.build.outputs.image-digest }}" \
            --values charts/webapp/values-staging.yaml \
            --wait --timeout 10m

  deploy-production:
    needs: deploy-staging
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    environment:
      name: production    # Requires manual approval via GitHub Environments
      url: https://example.com
    steps:
      - uses: actions/checkout@v4
      - name: Deploy to production
        run: |
          helm upgrade --install webapp ./charts/webapp \
            --namespace production \
            --set image.digest="${{ needs.build.outputs.image-digest }}" \
            --values charts/webapp/values-production.yaml \
            --wait --timeout 15m
```

3.2 Reusable Workflows

```
# .github/workflows/reusable-deploy.yml
on:
  workflow_call:
    inputs:
      environment:
        required: true
        type: string
      image-digest:
        required: true
        type: string
    secrets:
      DEPLOY_ROLE_ARN:
        required: true

jobs:
  deploy:
```

```
    runs-on: ubuntu-latest
    environment: ${{ inputs.environment }}
    steps:
      - name: Deploy
        run: echo "Deploying ${{ inputs.image-digest }} to ${{ inputs.environment }}"
```

---

4. GitLab CI

```
# .gitlab-ci.yml
stages:
  - validate
  - test
  - build
  - scan
  - staging
  - production

variables:
  DOCKER_DRIVER: overlay2
  IMAGE_TAG: $CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA

# Reusable snippets
.docker-login: &docker-login
  before_script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY

lint:
  stage: validate
  image: node:20-alpine
  script:
    - npm ci --cache .npm
    - npm run lint
  cache:
    key: $CI_COMMIT_REF_SLUG
    paths: [.npm/]

test:
  stage: test
  image: node:20-alpine
  script:
    - npm ci
    - npm test -- --coverage --reporters=default --reporters=jest-junit
  coverage: '/Statements\s*:\s*(\d+\.?\d*)%/'
  artifacts:
    when: always
```

```
    reports:
      junit: junit.xml
      coverage_report:
        coverage_format: cobertura
        path: coverage/cobertura-coverage.xml
    expire_in: 1 week

build:
  stage: build
  image: docker:24
  services:
    - docker:24-dind
  <<: *docker-login
  script:
    - |
      docker build \
        --cache-from $CI_REGISTRY_IMAGE:latest \
        --tag $IMAGE_TAG \
        --tag $CI_REGISTRY_IMAGE:latest \
        --build-arg BUILDKIT_INLINE_CACHE=1 .
    - docker push $IMAGE_TAG
    - docker push $CI_REGISTRY_IMAGE:latest
  only:
    - main
    - merge_requests

trivy-scan:
  stage: scan
  image:
    name: aquasec/trivy:latest
    entrypoint: [""]
  script:
    - trivy image --exit-code 1 --severity HIGH,CRITICAL $IMAGE_TAG
  allow_failure: false
  needs: [build]

deploy-staging:
  stage: staging
  image: bitnami/kubectl:latest
  environment:
    name: staging
    url: https://staging.example.com
  script:
    - kubectl set image deployment/webapp webapp=$IMAGE_TAG -n staging
    - kubectl rollout status deployment/webapp -n staging --timeout=5m
  only:
    - main
```

```yaml
deploy-production:
  stage: production
  image: bitnami/kubectl:latest
  environment:
    name: production
    url: https://example.com
  when: manual    # Require manual trigger
  script:
    - kubectl set image deployment/webapp webapp=$IMAGE_TAG -n production
    - kubectl rollout status deployment/webapp -n production --timeout=10m
  only:
    - main
```

---

5. ArgoCD (GitOps)

```yaml
# argocd/application.yml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: webapp
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: production

  source:
    repoURL: https://github.com/company/k8s-manifests
    targetRevision: main
    path: apps/webapp/production
    helm:
      releaseName: webapp
      valueFiles:
        - values.yaml
        - values-production.yaml

  destination:
    server: https://kubernetes.default.svc
    namespace: production

  syncPolicy:
    automated:
      prune: true       # Xóa resources không còn trong Git
      selfHeal: true    # Tự sửa drift
      allowEmpty: false
```

```
    syncOptions:
      - CreateNamespace=true
      - PrunePropagationPolicy=foreground
      - ApplyOutOfSyncOnly=true
    retry:
      limit: 3
      backoff:
        duration: 5s
        factor: 2
        maxDuration: 3m

  revisionHistoryLimit: 5
```

```
# ArgoCD CLI commands
argocd login argocd.example.com
argocd app list
argocd app get webapp
argocd app sync webapp
argocd app rollback webapp 3    # Rollback về revision 3
argocd app diff webapp          # Xem diff với live state

# ApplicationSet - Tạo nhiều apps từ template
kubectl apply -f applicationset.yml
```

```
# ApplicationSet - Deploy cùng app lên nhiều clusters
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: webapp-appset
spec:
  generators:
    - list:
        elements:
          - cluster: staging
            url: https://staging-cluster.example.com
          - cluster: production
            url: https://prod-cluster.example.com
  template:
    metadata:
      name: "webapp-{{cluster}}"
    spec:
      source:
        path: "apps/webapp/{{cluster}}"
      destination:
        server: "{{url}}"
        namespace: webapp
```

6. Flux CD

```yaml
# flux-system/webapp-kustomization.yml
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: webapp
  namespace: flux-system
spec:
  interval: 10m
  path: ./apps/webapp
  prune: true
  sourceRef:
    kind: GitRepository
    name: k8s-manifests
  healthChecks:
    - apiVersion: apps/v1
      kind: Deployment
      name: webapp
      namespace: production
  postBuild:
    substituteFrom:
      - kind: ConfigMap
        name: cluster-vars
```

---

7. Pipeline Patterns

Trunk-Based Development

```
main ←── feature/login (short-lived, < 1 ngày)
main ←── fix/bug-123   (merge nhanh, feature flags)
main ──→ release/v2.1  (chỉ bugfix, không feature)
```

GitFlow

```
main ←──────────────────── hotfix/critical-bug
develop ←── feature/user-auth
develop ←── feature/payments
develop ──→ release/v2.0 ──→ main (tag v2.0.0)
```

Lựa chọn thực tế: - Trunk-based + feature flags: Team CI/CD mature, release thường xuyên - GitFlow: Enterprise, release cycle dài, nhiều version concurrent

---

8. Quality Gates

```
# SonarQube Quality Gate trong GitHub Actions
- name: SonarQube Scan
  uses: SonarSource/sonarqube-scan-action@master
  env:
    SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
    SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}

- name: SonarQube Quality Gate check
  uses: sonarsource/sonarqube-quality-gate-action@master
  timeout-minutes: 5
  env:
    SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
```

```
# sonar-project.properties
sonar.projectKey=company_webapp
sonar.qualitygate.wait=true
sonar.coverage.exclusions=**/*.test.ts,**/migrations/**
sonar.cpd.exclusions=**/*.spec.ts
```

Quality Gate thường dùng: - Coverage >= 80% - No new Critical/Blocker issues - Duplicated lines < 3% - Maintainability rating >= A

---

9. Artifact Versioning

```
# Semantic versioning: MAJOR.MINOR.PATCH
# MAJOR: breaking changes
# MINOR: new features, backward compatible
# PATCH: bug fixes

# Docker image tags
webapp:1.2.3           # Semantic version
webapp:1.2             # Minor floating tag
webapp:latest          # Latest (tránh dùng production)
webapp:sha-a1b2c3d     # Git SHA (immutable, recommended)
webapp:20240115-1423   # Timestamp

# Build metadata
webapp:1.2.3+build.42.sha.a1b2c3d

# Convention: Dùng SHA cho production deployments
# Dùng semver tags cho public releases
```

```
# semantic-release config (.releaserc.yml)
branches:
```

```
  - main
  - { name: 'develop', prerelease: true }
plugins:
  - '@semantic-release/commit-analyzer'
  - '@semantic-release/release-notes-generator'
  - '@semantic-release/changelog'
  - '@semantic-release/npm'
  - '@semantic-release/github'
  - - '@semantic-release/git'
    - assets: ['CHANGELOG.md', 'package.json']
      message: 'chore(release): ${nextRelease.version}'
```

---

10. Deployment Strategies

```
# Rolling Update (default K8s)
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 25%        # Tối đa extra pods
    maxUnavailable: 0    # Zero downtime

# Blue/Green với Argo Rollouts
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: webapp
spec:
  strategy:
    blueGreen:
      activeService: webapp-active
      previewService: webapp-preview
      autoPromotionEnabled: false    # Manual promotion
      prePromotionAnalysis:
        templates:
          - templateName: success-rate

# Canary với traffic splitting
spec:
  strategy:
    canary:
      steps:
        - setWeight: 5      # 5% traffic → new version
        - pause: {}          # Chờ manual approve
        - setWeight: 20
        - pause: { duration: 10m }
```

```
        - setWeight: 50
        - pause: { duration: 10m }
        - setWeight: 100
      canaryService: webapp-canary
      stableService: webapp-stable
      trafficRouting:
        nginx:
          stableIngress: webapp-ingress
      analysis:
        templates:
          - templateName: error-rate
        startingStep: 1
        args:
          - name: service-name
            value: webapp-canary
```

---

11. Pipeline Security

OIDC (Không cần long-lived credentials)

```
# GitHub Actions với AWS OIDC
- name: Configure AWS Credentials
  uses: aws-actions/configure-aws-credentials@v4
  with:
    role-to-assume: arn:aws:iam::123456789:role/github-actions
    aws-region: us-east-1
    # Không cần AWS_ACCESS_KEY_ID / AWS_SECRET_ACCESS_KEY!
```

```
# AWS Trust Policy cho GitHub Actions OIDC
data "aws_iam_policy_document" "github_actions" {
  statement {
    actions = ["sts:AssumeRoleWithWebIdentity"]
    principals {
      type        = "Federated"
      identifiers = [aws_iam_openid_connect_provider.github.arn]
    }
    condition {
      test     = "StringEquals"
      variable = "token.actions.githubusercontent.com:aud"
      values   = ["sts.amazonaws.com"]
    }
    condition {
      test     = "StringLike"
      variable = "token.actions.githubusercontent.com:sub"
      values   = ["repo:company/webapp:ref:refs/heads/main"]
```

```
    }
  }
}
```

Supply Chain Security (SLSA)

```yaml
# Tạo SLSA provenance với slsa-github-generator
jobs:
  build:
    outputs:
      hashes: ${{ steps.hash.outputs.hashes }}
    steps:
      - name: Build artifacts
        run: make build

      - name: Generate hashes
        id: hash
        run: |
          sha256sum artifacts/* | base64 -w0
          echo "hashes=$(sha256sum artifacts/* | base64 -w0)" >> $GITHUB_OUTPUT

  provenance:
    needs: [build]
    permissions:
      actions: read
      id-token: write
      contents: write
    uses: slsa-framework/slsa-github-generator/.github/workflows/generator_generic_slsa3.yml@v1.9.0
    with:
      base64-subjects: ${{ needs.build.outputs.hashes }}
```

Secrets Management

```
# Không bao giờ hardcode secrets trong Jenkinsfile/workflow
# Dùng External Secrets Operator với K8s
kubectl apply -f - <<EOF
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: webapp-secrets
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: aws-secretsmanager
    kind: ClusterSecretStore
  target:
```

```
    name: webapp-secrets
  data:
    - secretKey: db-password
      remoteRef:
        key: production/webapp/db
        property: password
EOF
```

---

12. Ví Dụ: Complete CI/CD cho Microservices

Mono-repo với 3 services: user-service, order-service, payment-service

```yaml
# .github/workflows/microservices-cicd.yml
name: Microservices CI/CD

on:
  push:
    branches: [main]
  pull_request:

jobs:
  detect-changes:
    runs-on: ubuntu-latest
    outputs:
      user-service:    ${{ steps.changes.outputs.user-service }}
      order-service:   ${{ steps.changes.outputs.order-service }}
      payment-service: ${{ steps.changes.outputs.payment-service }}
    steps:
      - uses: actions/checkout@v4
      - uses: dorny/paths-filter@v3
        id: changes
        with:
          filters: |
            user-service:
              - 'services/user-service/**'
            order-service:
              - 'services/order-service/**'
            payment-service:
              - 'services/payment-service/**'

  build-user-service:
    needs: detect-changes
    if: needs.detect-changes.outputs.user-service == 'true'
    uses: ./.github/workflows/reusable-service-build.yml
    with:
      service: user-service
```

```
      context: services/user-service
    secrets: inherit

  build-order-service:
    needs: detect-changes
    if: needs.detect-changes.outputs.order-service == 'true'
    uses: ./.github/workflows/reusable-service-build.yml
    with:
      service: order-service
      context: services/order-service
    secrets: inherit

  deploy-staging:
    needs: [build-user-service, build-order-service]
    if: always() && !failure() && github.ref == 'refs/heads/main'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Update image tags in Helm values
        run: |
          # ArgoCD Image Updater hoặc update values file
          for service in user-service order-service payment-service; do
            if [ -n "${{ needs.build-${service}.outputs.digest }}" ]; then
              yq -i ".services.${service}.image.digest = \"${{ needs.build-${service}.outputs.digest
↪  }}\"" \
                 argocd/values-staging.yaml
            fi
          done
      - name: Commit updated values
        run: |
          git config user.email "ci@company.com"
          git config user.name "CI Bot"
          git add argocd/values-staging.yaml
          git commit -m "chore: update staging image digests [skip ci]"
          git push
      # ArgoCD sẽ tự detect và sync
```

```
# ArgoCD Image Updater - Tự động update image tags
kubectl apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-image-updater-config
  namespace: argocd
data:
  registries.conf: |
    registries:
      - name: GitHub Container Registry
```

```
        prefix: ghcr.io
        api_url: https://ghcr.io
        credentials: secret:argocd/ghcr-credentials#token
EOF

# Annotation trên ArgoCD Application
metadata:
  annotations:
    argocd-image-updater.argoproj.io/image-list: "webapp=ghcr.io/company/webapp"
    argocd-image-updater.argoproj.io/webapp.update-strategy: digest
    argocd-image-updater.argoproj.io/webapp.allow-tags: "regexp:^sha-"
```

---

13. Monitoring Pipeline Health

```
# Datadog pipeline monitoring
- name: Send deployment metric
  run: |
    curl -X POST "https://api.datadoghq.com/api/v1/events" \
      -H "DD-API-KEY: ${{ secrets.DATADOG_API_KEY }}" \
      -d '{
        "title": "Deployment: webapp v${{ env.VERSION }}",
        "text": "Deployed to production",
        "tags": ["env:production", "service:webapp"],
        "alert_type": "info"
      }'

# DORA metrics tracking
# Lead time, deployment frequency, MTTR, change failure rate
```

---

14. Best Practices Tổng Hợp

1. **Fail fast**: Chạy lint và unit tests trước, integration tests sau
2. **Parallel jobs**: Tăng tốc pipeline bằng parallel execution
3. **Cache aggressively**: Cache dependencies, Docker layers, build artifacts
4. **Immutable artifacts**: Build một lần, deploy nhiều lần (không rebuild)
5. **Environment parity**: Dev = Staging = Production (chỉ khác config)
6. **Zero-trust secrets**: Dùng OIDC, không dùng long-lived credentials
7. **Sign artifacts**: Cosign cho container images, SLSA provenance
8. **Rollback plan**: Luôn có rollback strategy trước khi deploy
9. **Observability**: Correlate deployments với metrics/errors (Datadog, Honeycomb)
10. **Pipeline as code**: Jenkinsfile/workflow trong Git, reviewed như application code

Chapter 17: Infrastructure Monitoring

Infrastructure monitoring là nền tảng của observability - khả năng hiểu trạng thái hệ thống từ output của nó. Không có monitoring tốt, vận hành production là đi trong bóng tối.

---

1. Monitoring Fundamentals

1.1 Metric Types

Counter - chỉ tăng, không giảm (trừ khi reset): - Số requests, errors, bytes transferred - Dùng `rate()` hoặc `increase()` để tính tốc độ thay đổi

Gauge - giá trị có thể tăng/giảm tùy thời điểm: - CPU usage, memory, queue size, active connections - Đọc giá trị tức thời là có ý nghĩa

Histogram - phân phối giá trị vào các bucket: - Request latency, response size - Tính được percentile (p50, p95, p99) - Tốn storage hơn nhưng linh hoạt hơn

Summary - pre-computed percentiles phía client: - Chính xác hơn histogram nhưng không aggregate được cross-instance - Dùng khi chỉ cần percentile tại một node

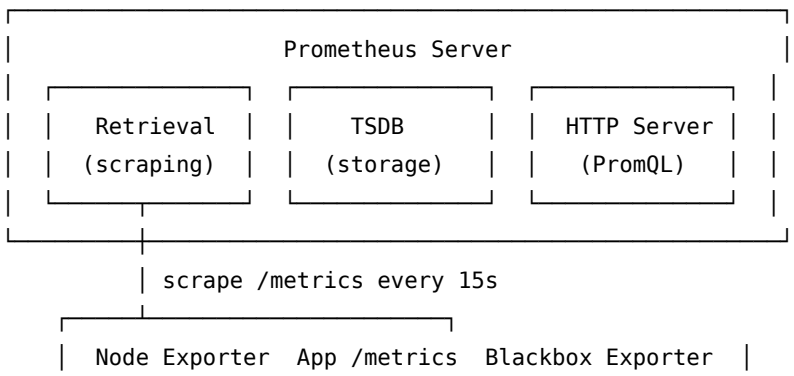
1.2 The Four Golden Signals (Google SRE)

```
1. Latency    - Thời gian xử lý request (phân biệt success vs error latency)
2. Traffic    - Số lượng demand (RPS, queries/sec)
3. Errors     - Tỷ lệ requests thất bại (explicit 5xx, implicit wrong data)
4. Saturation - Mức độ "đầy" của resource (CPU, memory, disk queue)
```

---

2. Prometheus

2.1 Architecture



Prometheus theo mô hình **pull** - server chủ động scrape targets. Điều này giúp kiểm soát load và phát hiện target down.

2.2 Cấu hình cơ bản

```yaml
# prometheus.yml
global:
  scrape_interval: 15s
  evaluation_interval: 15s
  external_labels:
    cluster: production
    region: ap-southeast-1

rule_files:
  - /etc/prometheus/rules/*.yml

alerting:
  alertmanagers:
    - static_configs:
        - targets: ["alertmanager:9093"]

scrape_configs:
  - job_name: node-exporter
    static_configs:
      - targets:
          - node-01:9100
          - node-02:9100
    relabel_configs:
      - source_labels: [__address__]
        target_label: instance

  - job_name: kubernetes-pods
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: "true"
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)
```

2.3 PromQL - Prometheus Query Language

```
# Rate of HTTP requests per second (5m window)
rate(http_requests_total[5m])

# Error rate
sum(rate(http_requests_total{status=~"5.."}[5m])) /
```

```
sum(rate(http_requests_total[5m]))

# P99 latency từ histogram
histogram_quantile(0.99, sum(rate(http_request_duration_seconds_bucket[5m])) by (le, service))

# Memory usage %
(1 - node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) * 100

# CPU usage (aggregate across cores)
100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)

# Disk usage %
(node_filesystem_size_bytes - node_filesystem_free_bytes) / node_filesystem_size_bytes * 100

# Network traffic in/out Mbps
rate(node_network_receive_bytes_total{device!="lo"}[5m]) * 8 / 1024 / 1024
rate(node_network_transmit_bytes_total{device!="lo"}[5m]) * 8 / 1024 / 1024
```

2.4 Service Discovery

```
# Kubernetes service discovery - scrape tất cả services có annotation
- job_name: kubernetes-services
  kubernetes_sd_configs:
    - role: endpoints
      namespaces:
        names: [default, production, staging]
  relabel_configs:
    - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scrape]
      action: keep
      regex: "true"
    - source_labels: [__meta_kubernetes_namespace]
      target_label: namespace
    - source_labels: [__meta_kubernetes_service_name]
      target_label: service
```

---

3. Alerting với Alertmanager

3.1 Alerting Rules

```
# /etc/prometheus/rules/infrastructure.yml
groups:
  - name: infrastructure
    interval: 1m
    rules:
```

```
    - alert: HighCPUUsage
      expr: |
        100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100) > 85
      for: 5m
      labels:
        severity: warning
        team: platform
      annotations:
        summary: "High CPU on {{ $labels.instance }}"
        description: "CPU usage is {{ $value | humanize }}% for 5+ minutes"
        runbook: "https://runbooks.company.com/high-cpu"

    - alert: CriticalCPUUsage
      expr: |
        100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100) > 95
      for: 2m
      labels:
        severity: critical
        team: platform
      annotations:
        summary: "CRITICAL: CPU near 100% on {{ $labels.instance }}"
        runbook: "https://runbooks.company.com/critical-cpu"

    - alert: MemoryPressure
      expr: |
        (1 - node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) * 100 > 90
      for: 5m
      labels:
        severity: critical
      annotations:
        summary: "Memory critical on {{ $labels.instance }}"

    - alert: DiskSpaceLow
      expr: |
        (node_filesystem_size_bytes - node_filesystem_free_bytes) /
        node_filesystem_size_bytes * 100 > 80
      for: 10m
      labels:
        severity: warning
      annotations:
        summary: "Disk {{ $labels.mountpoint }} at {{ $value | humanize }}% on {{ $labels.instance }}"

    # SLO-based alert: error budget burn rate
    - alert: ErrorBudgetBurnRate
      expr: |
        (
          sum(rate(http_requests_total{status=~"5.."}[1h])) /
          sum(rate(http_requests_total[1h]))
```

```
          ) > (1 - 0.999) * 14.4
        for: 5m
        labels:
          severity: critical
          slo: availability
        annotations:
          summary: "Error budget burning fast - SLO at risk"
```

3.2 Alertmanager Configuration

```
# alertmanager.yml
global:
  resolve_timeout: 5m
  slack_api_url: "https://hooks.slack.com/services/xxx"
  pagerduty_url: "https://events.pagerduty.com/v2/enqueue"

route:
  receiver: default
  group_by: [alertname, cluster, namespace]
  group_wait: 30s       # chờ nhóm các alert liên quan
  group_interval: 5m    # gửi tiếp theo sau bao lâu
  repeat_interval: 4h   # re-notify nếu chưa resolve

  routes:
    - match:
        severity: critical
      receiver: pagerduty-critical
      continue: true    # tiếp tục match routes khác

    - match:
        severity: critical
      receiver: slack-critical

    - match:
        severity: warning
      receiver: slack-warning

    - match:
        team: database
      receiver: dba-team-slack

receivers:
  - name: default
    slack_configs:
      - channel: "#alerts-default"
        title: "{{ .GroupLabels.alertname }}"
        text: "{{ range .Alerts }}{{ .Annotations.summary }}{{ end }}"
```

```
  - name: pagerduty-critical
    pagerduty_configs:
      - service_key: "{{ env \"PAGERDUTY_KEY\" }}"
        description: "{{ .GroupLabels.alertname }}: {{ .CommonAnnotations.summary }}"
        severity: critical

  - name: slack-critical
    slack_configs:
      - channel: "#alerts-critical"
        color: danger
        title: "CRITICAL: {{ .GroupLabels.alertname }}"

inhibit_rules:
  # Nếu có critical alert, suppress warning cùng instance
  - source_match:
      severity: critical
    target_match:
      severity: warning
    equal: [instance, alertname]
```

---

4. Node Exporter

4.1 Cài đặt và cấu hình

```
# Cài Node Exporter
wget https://github.com/prometheus/node_exporter/releases/download/v1.7.0/node_exporter-1.7.0.linux-
↪  amd64.tar.gz
tar xvf node_exporter-*.tar.gz
sudo mv node_exporter-*/node_exporter /usr/local/bin/

# systemd service
cat > /etc/systemd/system/node_exporter.service << 'EOF'
[Unit]
Description=Prometheus Node Exporter
After=network.target

[Service]
User=prometheus
ExecStart=/usr/local/bin/node_exporter \
  --collector.systemd \
  --collector.processes \
  --collector.diskstats.ignored-devices="^(ram|loop|fd|(h|s|v|xv)d[a-z]|nvme\\d+n\\d+p)\\d+$" \
  --web.listen-address=:9100
Restart=always
```

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

systemctl daemon-reload
systemctl enable --now node_exporter
```

4.2 Custom Textfile Collector

```
#!/bin/bash
# /usr/local/bin/collect-custom-metrics.sh
# Chạy môĩ phút qua cron, output vào textfile collector

TEXTFILE_DIR=/var/lib/prometheus/textfile_collector

# Sô´backup files còn lại trong 24h
backup_count=$(find /backup -name "*.tar.gz" -mtime -1 | wc -l)
echo "backup_files_last_24h ${backup_count}" > ${TEXTFILE_DIR}/backup.prom

# Chứng chỉ SSL còn hạn bao nhiêu ngày
cert_expiry=$(openssl x509 -enddate -noout -in /etc/ssl/certs/app.crt | \
  date -d "$(awk -F= '{print $2}')" +%s)
echo "ssl_certificate_expiry_seconds ${cert_expiry}" >> ${TEXTFILE_DIR}/ssl.prom
```

---

5. Blackbox Exporter

Probe endpoints từ bên ngoài - HTTP, TCP, ICMP, DNS.

```
# blackbox.yml
modules:
  http_2xx:
    prober: http
    timeout: 5s
    http:
      valid_http_versions: ["HTTP/1.1", "HTTP/2.0"]
      valid_status_codes: [200, 201, 204]
      follow_redirects: true
      tls_config:
        insecure_skip_verify: false

  http_post_json:
    prober: http
    http:
      method: POST
```

```
      headers:
        Content-Type: application/json
      body: '{"healthcheck": true}'

  tcp_connect:
    prober: tcp
    timeout: 5s

  icmp_ping:
    prober: icmp
    timeout: 5s

# prometheus.yml - scrape blackbox
- job_name: blackbox-http
  metrics_path: /probe
  params:
    module: [http_2xx]
  static_configs:
    - targets:
        - https://api.company.com/health
        - https://app.company.com
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: blackbox-exporter:9115
```

---

6. Thanos - Long-term Storage và Multi-cluster

```
# Thanos Sidecar chạy cạnh Prometheus
# Upload blocks lên object storage (S3/GCS) sau mỗi 2h

thanos sidecar \
  --tsdb.path /prometheus \
  --prometheus.url http://localhost:9090 \
  --objstore.config-file /etc/thanos/objstore.yml \
  --http-address 0.0.0.0:19191 \
  --grpc-address 0.0.0.0:19090

# objstore.yml
type: S3
config:
  bucket: company-prometheus-long-term
```

```
  endpoint: s3.ap-southeast-1.amazonaws.com
  region: ap-southeast-1

# Thanos Query - query across multiple Prometheus
thanos query \
  --store thanos-sidecar-cluster-a:19090 \
  --store thanos-sidecar-cluster-b:19090 \
  --store thanos-store:19090 \
  --query.replica-label replica
```

---

7. Grafana Dashboards

7.1 Dashboard as Code (Grafonnet/JSON)

```
{
  "title": "Infrastructure Overview",
  "refresh": "30s",
  "templating": {
    "list": [
      {
        "name": "instance",
        "type": "query",
        "query": "label_values(node_uname_info, instance)",
        "multi": true,
        "includeAll": true
      },
      {
        "name": "interval",
        "type": "interval",
        "options": ["1m", "5m", "15m", "1h"]
      }
    ]
  },
  "panels": [
    {
      "title": "CPU Usage %",
      "type": "timeseries",
      "targets": [
        {
          "expr": "100 - (avg by (instance) (rate(node_cpu_seconds_total{mode='idle',
          ↪  instance=~'$instance'}[$interval])) * 100)",
          "legendFormat": "{{instance}}"
        }
      ],
      "fieldConfig": {
```

```
        "defaults": {
          "unit": "percent",
          "thresholds": {
            "steps": [
              {"color": "green", "value": 0},
              {"color": "yellow", "value": 70},
              {"color": "red", "value": 90}
            ]
          }
        }
      }
    }
  ]
}
```

7.2 Annotations - đánh dấu deployments

```
# Thêm annotation khi deploy
curl -X POST http://grafana:3000/api/annotations \
  -H "Authorization: Bearer ${GRAFANA_API_KEY}" \
  -H "Content-Type: application/json" \
  -d '{
    "text": "Deployed v2.3.1 to production",
    "tags": ["deployment", "production"],
    "time": '$(date +%s%3N)'
  }'
```

---

8. Alerting Best Practices

```
Nguyên tắć alert tốt:
1. Alert on symptoms, not causes       - Alert "users cannot login", không phải "DB CPU high"
2. Alert must be actionable            - Mỗĩ alert cần runbook
3. Eliminate noisy alerts              - Alert fatigue = on-call bỏ qua alerts
4. SLO-based alerting                  - Tập trung vào user impact
5. Severity levels rõ ràng:
   - P1/Critical: wake up on-call, user impact NOW
   - P2/Warning:  address trong business hours
   - P3/Info:     log, không notify
```

8.1 SLO-based Alerting

```
# SLO: 99.9% availability (error budget = 43.2 min/month)
# Multi-window, multi-burn-rate alert
```

```
# Fast burn: 14.4x burn rate trong 1h → alert ngay
(
  sum(rate(http_requests_total{status=~"5.."}[1h])) /
  sum(rate(http_requests_total[1h]))
) > 14.4 * (1 - 0.999)

# Slow burn: 3x burn rate trong 6h → warning
(
  sum(rate(http_requests_total{status=~"5.."}[6h])) /
  sum(rate(http_requests_total[6h]))
) > 3 * (1 - 0.999)
```

---

9. On-call Integration

9.1 PagerDuty

```
# alertmanager.yml - PagerDuty routing
receivers:
  - name: pagerduty-p1
    pagerduty_configs:
      - routing_key: R1234ABCD
        severity: critical
        client: "Alertmanager"
        client_url: "https://alertmanager.company.com"
        description: "{{ .GroupLabels.alertname }}"
        details:
          runbook: "{{ .CommonAnnotations.runbook }}"
          cluster: "{{ .GroupLabels.cluster }}"
```

```
# Test PagerDuty integration
amtool alert add \
  alertname=TestAlert \
  severity=critical \
  --annotation=summary="Test alert from amtool" \
  --alertmanager.url=http://alertmanager:9093
```

---

10. Ví dụ Production Stack

```
# docker-compose.yml - monitoring stack hoàn chỉnh
version: "3.8"
services:
  prometheus:
    image: prom/prometheus:v2.48.0
```

```
  volumes:
    - ./prometheus:/etc/prometheus
    - prometheus_data:/prometheus
  command:
    - --config.file=/etc/prometheus/prometheus.yml
    - --storage.tsdb.retention.time=15d
    - --storage.tsdb.path=/prometheus
    - --web.enable-lifecycle
  ports:
    - "9090:9090"

alertmanager:
  image: prom/alertmanager:v0.26.0
  volumes:
    - ./alertmanager:/etc/alertmanager
  command:
    - --config.file=/etc/alertmanager/alertmanager.yml
    - --storage.path=/alertmanager
    - --cluster.listen-address=""
  ports:
    - "9093:9093"

node-exporter:
  image: prom/node-exporter:v1.7.0
  network_mode: host
  pid: host
  volumes:
    - /proc:/host/proc:ro
    - /sys:/host/sys:ro
    - /:/rootfs:ro
  command:
    - --path.procfs=/host/proc
    - --path.sysfs=/host/sys
    - --path.rootfs=/rootfs
    - --collector.filesystem.mount-points-exclude=^/(sys|proc|dev|host|etc)($$|/)

grafana:
  image: grafana/grafana:10.2.2
  volumes:
    - grafana_data:/var/lib/grafana
    - ./grafana/provisioning:/etc/grafana/provisioning
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=secure_password_here
    - GF_INSTALL_PLUGINS=grafana-piechart-panel
  ports:
    - "3000:3000"

blackbox-exporter:
```

```
    image: prom/blackbox-exporter:v0.24.0
    volumes:
      - ./blackbox:/etc/blackbox_exporter
    ports:
      - "9115:9115"

volumes:
  prometheus_data:
  grafana_data:
```

```
# Kiểm tra Prometheus targets
curl -s http://localhost:9090/api/v1/targets | jq '.data.activeTargets[] | {job: .labels.job, health:
↪  .health}'

# Reload config không restart
curl -X POST http://localhost:9090/-/reload

# Query PromQL từ CLI
curl -s 'http://localhost:9090/api/v1/query?query=up' | jq '.data.result'

# Check alertmanager status
amtool --alertmanager.url=http://localhost:9093 alert query
amtool --alertmanager.url=http://localhost:9093 silence query
```

Tóm tắt

Component	Vai trò	Port
Prometheus	Scrape, store, query metrics	9090
Alertmanager	Route, deduplicate, silence alerts	9093
Node Exporter	System metrics (CPU, mem, disk)	9100
Blackbox Exporter	Probe endpoints externally	9115
Grafana	Visualize, dashboard, alerting UI	3000
Thanos Sidecar	Long-term storage, remote write	19090

Key takeaway: Monitoring hiệu quả = đúng metrics + đúng alerts + runbooks rõ ràng. Tránh alert fatigue bằng cách chỉ alert những gì thực sự cần action.

Chapter 18: Application Performance Monitoring (APM)

APM đi sâu hơn infrastructure monitoring - thay vì hỏi "server có OK không?", APM hỏi "ứng dụng có hoạt động đúng không? User có đang gặp vấn đề gì không?". Distributed tracing, metrics, và profiling là ba trụ cột.

---

1. Distributed Tracing Concepts

1.1 Traces, Spans, và Context Propagation

```
Một request qua microservices:

[Browser] ──► [API Gateway] ──► [Auth Service] ──► [User Service] ──► [DB]
                                        │
                                        └──► [Cache (Redis)]

Trace ID: abc-123 (xuyên suốt toàn bộ request)
├── Span: api-gateway        duration=150ms
│   ├── Span: auth-service   duration=30ms
│   │   └── Span: cache      duration=2ms (cache hit)
│   └── Span: user-service   duration=80ms
│       └── Span: postgres   duration=15ms (query)
```

Trace: toàn bộ hành trình của một request **Span**: một đơn vị công việc trong trace (có start time, duration, tags, logs) **Context Propagation**: truyền trace/span IDs qua HTTP headers hoặc message queues

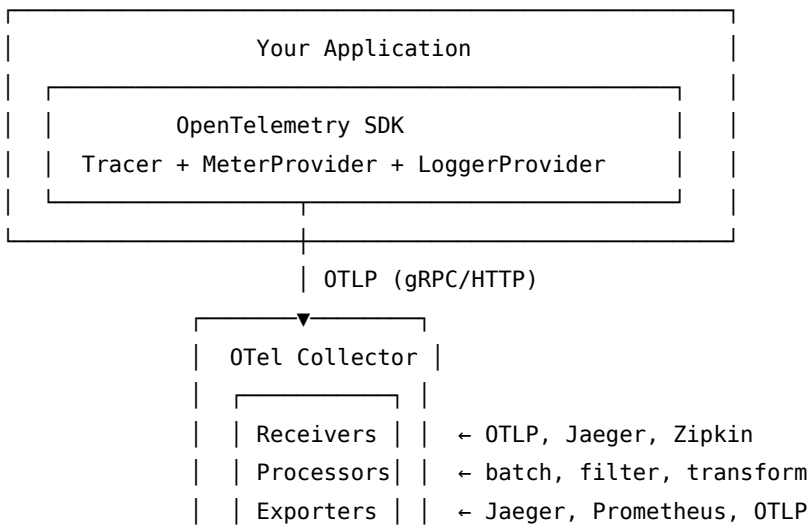
```
HTTP Headers (W3C Trace Context standard):
traceparent: 00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01
             version  traceId(16 bytes)         spanId(8 bytes)  flags
tracestate:   vendor-specific key-value pairs
```

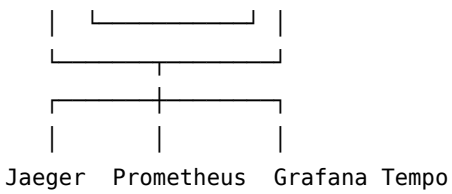
---

2. OpenTelemetry

OpenTelemetry (OTel) là standard mở cho instrumentation - một SDK, nhiều backends.

2.1 Architecture





2.2 Auto-instrumentation (Python)

```
pip install opentelemetry-distro opentelemetry-exporter-otlp
opentelemetry-bootstrap -a install

# Chạy app với auto-instrumentation
OTEL_SERVICE_NAME=user-service \
OTEL_EXPORTER_OTLP_ENDPOINT=http://otel-collector:4317 \
OTEL_TRACES_SAMPLER=parentbased_traceidratio \
OTEL_TRACES_SAMPLER_ARG=0.1 \
opentelemetry-instrument python app.py
```

2.3 Manual Instrumentation (Python)

```
from opentelemetry import trace, metrics
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import OTLPSpanExporter
from opentelemetry.sdk.metrics import MeterProvider
from opentelemetry.sdk.metrics.export import PeriodicExportingMetricReader
from opentelemetry.exporter.otlp.proto.grpc.metric_exporter import OTLPMetricExporter

# Setup tracing
tracer_provider = TracerProvider()
tracer_provider.add_span_processor(
    BatchSpanProcessor(OTLPSpanExporter(endpoint="http://otel-collector:4317"))
)
trace.set_tracer_provider(tracer_provider)

# Setup metrics
metric_reader = PeriodicExportingMetricReader(
    OTLPMetricExporter(endpoint="http://otel-collector:4317"),
    export_interval_millis=30000
)
meter_provider = MeterProvider(metric_readers=[metric_reader])
metrics.set_meter_provider(meter_provider)

# Sử dụng trong code
tracer = trace.get_tracer("user-service")
meter = metrics.get_meter("user-service")
```

```python
# Counter cho business metrics
order_counter = meter.create_counter(
    name="orders.created",
    description="Number of orders created",
    unit="1"
)

# Histogram cho latency
db_query_duration = meter.create_histogram(
    name="db.query.duration",
    description="Database query duration",
    unit="ms"
)

def get_user(user_id: str):
    with tracer.start_as_current_span("get_user") as span:
        span.set_attribute("user.id", user_id)
        span.set_attribute("db.system", "postgresql")

        start = time.time()
        try:
            user = db.query("SELECT * FROM users WHERE id = %s", user_id)
            span.set_attribute("db.rows_returned", 1 if user else 0)
            db_query_duration.record(
                (time.time() - start) * 1000,
                attributes={"query": "get_user", "status": "success"}
            )
            return user
        except Exception as e:
            span.record_exception(e)
            span.set_status(trace.Status(trace.StatusCode.ERROR, str(e)))
            db_query_duration.record(
                (time.time() - start) * 1000,
                attributes={"query": "get_user", "status": "error"}
            )
            raise
```

2.4 OTel Collector Configuration

```yaml
# otel-collector-config.yml
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318
```

```yaml
  prometheus:
    config:
      scrape_configs:
        - job_name: otel-collector
          static_configs:
            - targets: [localhost:8888]

processors:
  batch:
    timeout: 1s
    send_batch_size: 1024
  memory_limiter:
    limit_mib: 512
    spike_limit_mib: 128
  resource:
    attributes:
      - key: environment
        value: production
        action: insert
  filter/exclude_health:
    traces:
      span:
        - 'attributes["http.target"] == "/health"'

exporters:
  jaeger:
    endpoint: jaeger:14250
    tls:
      insecure: true
  prometheus:
    endpoint: 0.0.0.0:8889
  otlp/tempo:
    endpoint: tempo:4317
    tls:
      insecure: true

service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [memory_limiter, filter/exclude_health, batch]
      exporters: [jaeger, otlp/tempo]
    metrics:
      receivers: [otlp, prometheus]
      processors: [memory_limiter, batch]
      exporters: [prometheus]
```

3. Jaeger

3.1 Architecture

```
Sampling Strategies:
- Constant (always/never)     - dev/test
- Probabilistic (ratio)       - production low traffic
- Rate Limiting               - production high traffic
- Remote (adaptive)           - dynamic, recommended
```

```yaml
# jaeger docker-compose
services:
  jaeger:
    image: jaegertracing/all-in-one:1.52
    environment:
      - COLLECTOR_OTLP_ENABLED=true
      - SPAN_STORAGE_TYPE=elasticsearch
      - ES_SERVER_URLS=http://elasticsearch:9200
    ports:
      - "16686:16686"  # UI
      - "14250:14250"  # gRPC collector
      - "4317:4317"    # OTLP gRPC
      - "4318:4318"    # OTLP HTTP
```

3.2 Sampling Strategy

```json
{
  "service_strategies": [
    {
      "service": "payment-service",
      "type": "probabilistic",
      "param": 1.0
    },
    {
      "service": "api-gateway",
      "type": "ratelimiting",
      "param": 100
    }
  ],
  "default_strategy": {
    "type": "probabilistic",
    "param": 0.1
  }
}
```

---

4. RED Method và USE Method

4.1 RED Method (cho services)

```
R - Rate:     requests/second service đang xử lý
E - Errors:   tỷ lệ request thất bại
D - Duration: latency phân phối (p50, p95, p99)
```

```
# Rate
sum(rate(http_server_requests_total{service="order-service"}[5m]))

# Error rate
sum(rate(http_server_requests_total{service="order-service", status=~"5.."}[5m])) /
sum(rate(http_server_requests_total{service="order-service"}[5m]))

# P99 Duration
histogram_quantile(0.99,
  sum(rate(http_server_request_duration_seconds_bucket{service="order-service"}[5m])) by (le)
)
```

4.2 USE Method (cho resources)

```
U - Utilization:  % thời gian resource đang bận
S - Saturation:   lượng công việc đang chờ (queue depth)
E - Errors:       lỗi liên quan resource

Áp dụng cho: CPU, Memory, Disk I/O, Network, Thread Pool
```

---

5. Custom Business Metrics

```
# Metrics có ý nghĩa business
meter = metrics.get_meter("ecommerce")

# Business KPIs
revenue_counter = meter.create_counter("revenue.total", unit="USD")
orders_histogram = meter.create_histogram("order.value", unit="USD")
cart_abandonment = meter.create_counter("cart.abandoned")
checkout_duration = meter.create_histogram("checkout.duration", unit="s")

# SLI metrics
login_success = meter.create_counter("auth.login",
    description="Login attempts by outcome")

def process_order(order):
    with tracer.start_as_current_span("process_order"):
        start = time.time()
        result = payment_gateway.charge(order)
```

```
        orders_histogram.record(
            order.total,
            attributes={"currency": order.currency, "method": order.payment_method}
        )
        revenue_counter.add(
            order.total,
            attributes={"region": order.region}
        )
        checkout_duration.record(
            time.time() - start,
            attributes={"status": "success" if result.ok else "failed"}
        )
```

---

6. APM Tools Thương mại

6.1 Datadog APM

```
# ddtrace - Datadog Python agent
from ddtrace import tracer, patch_all
patch_all()  # auto-patch supported libraries

@tracer.wrap(service="user-service", resource="get_user")
def get_user(user_id):
    return db.get(user_id)

# Custom spans
with tracer.trace("custom.operation", service="billing") as span:
    span.set_tag("invoice.id", invoice_id)
    span.set_tag("amount", amount)
    process_invoice(invoice_id)
```

```
# Chạy với Datadog agent
DD_AGENT_HOST=datadog-agent \
DD_TRACE_SAMPLE_RATE=0.5 \
DD_ENV=production \
DD_VERSION=2.3.1 \
python app.py
```

6.2 Elastic APM

```
from elasticapm.contrib.flask import ElasticAPM

app = Flask(__name__)
```

```
app.config['ELASTIC_APM'] = {
    'SERVICE_NAME': 'user-service',
    'SECRET_TOKEN': os.environ['ELASTIC_APM_TOKEN'],
    'SERVER_URL': 'http://apm-server:8200',
    'ENVIRONMENT': 'production',
    'TRANSACTION_SAMPLE_RATE': 0.1,
}
apm = ElasticAPM(app)
```

---

7. Continuous Profiling

Profiling trong production giúp tìm CPU hotspots và memory leaks mà không cần reproduce locally.

```
# Pyroscope - continuous profiling
import pyroscope

pyroscope.configure(
    application_name="user-service",
    server_address="http://pyroscope:4040",
    auth_token=os.environ.get("PYROSCOPE_TOKEN"),
    tags={"region": "ap-southeast-1", "version": "2.3.1"}
)

# Hoặc dùng pprof cho Go
import _ "net/http/pprof"
go func() {
    log.Println(http.ListenAndServe(":6060", nil))
}()
```

```
# Go pprof - lấy CPU profile 30 giây
go tool pprof http://localhost:6060/debug/pprof/profile?seconds=30

# Memory profile
go tool pprof http://localhost:6060/debug/pprof/heap

# Visualize flame graph
go tool pprof -http=:8080 profile.out
```

---

8. Error Tracking với Sentry

```
import sentry_sdk
from sentry_sdk.integrations.flask import FlaskIntegration
```

```python
from sentry_sdk.integrations.sqlalchemy import SqlalchemyIntegration

sentry_sdk.init(
    dsn=os.environ["SENTRY_DSN"],
    integrations=[FlaskIntegration(), SqlalchemyIntegration()],
    environment="production",
    release="user-service@2.3.1",
    traces_sample_rate=0.1,
    profiles_sample_rate=0.1,
    # Lọc sensitive data
    before_send=lambda event, hint: scrub_pii(event),
)

# Gắń user context
with sentry_sdk.push_scope() as scope:
    scope.user = {"id": user_id, "email": user_email}
    scope.set_tag("payment_method", "credit_card")
    scope.set_context("order", {"id": order_id, "total": total})

    try:
        process_payment(order)
    except PaymentException as e:
        sentry_sdk.capture_exception(e)
        raise
```

---

9. Correlating Logs + Metrics + Traces

```python
import logging
import json
from opentelemetry import trace

class OtelJsonFormatter(logging.Formatter):
    """Inject trace context vào log records."""

    def format(self, record):
        log_data = {
            "timestamp": self.formatTime(record),
            "level": record.levelname,
            "logger": record.name,
            "message": record.getMessage(),
            "service": os.environ.get("OTEL_SERVICE_NAME", "unknown"),
        }

        # Inject trace context
        span = trace.get_current_span()
```

```
        if span.is_recording():
            ctx = span.get_span_context()
            log_data["trace_id"] = format(ctx.trace_id, "032x")
            log_data["span_id"] = format(ctx.span_id, "016x")

        return json.dumps(log_data)

# Với trace_id trong log, có thể jump từ log → trace trong Grafana/Jaeger
```

```
# Grafana Explore: correlate logs và traces
# Loki datasource - derived field từ trace_id
derivedFields:
  - name: TraceID
    matcherRegex: '"trace_id":"(\w+)"'
    url: "http://jaeger:16686/trace/$${__value.raw}"
    urlDisplayLabel: "View Trace in Jaeger"
```

---

10. Ví dụ: Instrument Microservices với OpenTelemetry

```
# shared/telemetry.py - module dùng chung cho tất cả services
import os
from opentelemetry import trace, metrics
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.sdk.metrics import MeterProvider
from opentelemetry.sdk.metrics.export import PeriodicExportingMetricReader
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import OTLPSpanExporter
from opentelemetry.exporter.otlp.proto.grpc.metric_exporter import OTLPMetricExporter
from opentelemetry.sdk.resources import Resource
from opentelemetry.instrumentation.requests import RequestsInstrumentor
from opentelemetry.instrumentation.sqlalchemy import SQLAlchemyInstrumentor

def setup_telemetry(service_name: str, service_version: str = "unknown"):
    resource = Resource.create({
        "service.name": service_name,
        "service.version": service_version,
        "deployment.environment": os.environ.get("ENVIRONMENT", "development"),
    })

    otel_endpoint = os.environ.get("OTEL_EXPORTER_OTLP_ENDPOINT", "http://localhost:4317")

    # Tracing
    tracer_provider = TracerProvider(resource=resource)
    tracer_provider.add_span_processor(
        BatchSpanProcessor(OTLPSpanExporter(endpoint=otel_endpoint, insecure=True))
```

```
    )
    trace.set_tracer_provider(tracer_provider)

    # Metrics
    metric_reader = PeriodicExportingMetricReader(
        OTLPMetricExporter(endpoint=otel_endpoint, insecure=True),
        export_interval_millis=30000
    )
    metrics.set_meter_provider(MeterProvider(resource=resource, metric_readers=[metric_reader]))

    # Auto-instrument libraries
    RequestsInstrumentor().instrument()
    SQLAlchemyInstrumentor().instrument()

    return trace.get_tracer(service_name), metrics.get_meter(service_name)

# order-service/main.py
from shared.telemetry import setup_telemetry

tracer, meter = setup_telemetry("order-service", "1.5.2")

orders_created = meter.create_counter("orders.created")
order_total = meter.create_histogram("order.total.usd")

@app.route("/orders", methods=["POST"])
def create_order():
    with tracer.start_as_current_span("create_order") as span:
        data = request.json
        span.set_attribute("order.items_count", len(data["items"]))

        order = Order.create(data)

        orders_created.add(1, {"region": data.get("region", "unknown")})
        order_total.record(order.total, {"currency": order.currency})

        return jsonify(order.to_dict()), 201
```

```
# Docker Compose để test locally
services:
  otel-collector:
    image: otel/opentelemetry-collector-contrib:0.90.1
    volumes:
      - ./otel-config.yml:/etc/otelcol/config.yaml
    ports:
      - "4317:4317"
      - "8889:8889"

  jaeger:
```

```
    image: jaegertracing/all-in-one:1.52
    ports:
      - "16686:16686"

  prometheus:
    image: prom/prometheus:v2.48.0
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml

  grafana:
    image: grafana/grafana:10.2.2
    environment:
      - GF_FEATURE_TOGGLES_ENABLE=traceqlEditor
    ports:
      - "3000:3000"
```

Tóm tắt

Tool	Mục đích	When to use
OpenTelemetry SDK	Instrumentation standard	Mọi service mới
OTel Collector	Gateway, processing pipeline	Bắt buộc trong production
Jaeger/Tempo	Distributed tracing backend	Tìm latency bottlenecks
Prometheus	Metrics storage	RED/USE metrics
Sentry	Error tracking	Catch & alert exceptions
Pyroscope	Continuous profiling	Optimize CPU/memory
Datadog/New Relic	All-in-one APM	Khi cần managed solution

Key takeaway: Bắt đầu với RED metrics + basic tracing. Thêm business metrics và profiling khi cần optimize. Correlation giữa logs/metrics/traces là yếu tố quan trọng nhất để debug nhanh.

Chapter 19: Logs Management

Logs là nguồn thông tin chi tiết nhất về những gì xảy ra bên trong hệ thống. Quản lý logs hiệu quả giúp debug nhanh, audit compliance, và phát hiện security incidents. Thách thức: volume lớn, nhiều format, cần query real-time.

---

1. Logging Fundamentals

1.1 Structured Logging

Tránh unstructured logs - khó parse, khó query:

```
# BAD - unstructured
logger.info(f"User {user_id} placed order {order_id} for ${amount}")

# GOOD - structured JSON
logger.info("order.created", extra={
    "user_id": user_id,
    "order_id": order_id,
    "amount": amount,
    "currency": "USD",
    "payment_method": "credit_card",
    "region": "ap-southeast-1",
})
```

Output JSON:

```
{
  "timestamp": "2024-01-15T10:23:45.123Z",
  "level": "INFO",
  "logger": "order-service",
  "message": "order.created",
  "user_id": "usr_abc123",
  "order_id": "ord_xyz789",
  "amount": 149.99,
  "currency": "USD",
  "payment_method": "credit_card",
  "region": "ap-southeast-1",
  "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736",
  "span_id": "00f067aa0ba902b7",
  "service": "order-service",
  "version": "2.3.1",
  "environment": "production"
}
```

1.2 Log Levels

```
FATAL/CRITICAL  - App không thể tiếp tục, cần restart ngay
ERROR           - Lỗi cần investigate, nhưng app vẫn chạy
WARN            - Vấn đề tiềm ẩn, không phải lỗi hiện tại
INFO            - Events bình thường quan trọng (request served, order created)
DEBUG           - Chi tiết cho development/troubleshooting
TRACE           - Rất chi tiết, không dùng production
```

```
# Python logging với structlog
import structlog
```

```
log = structlog.get_logger()

structlog.configure(
    processors=[
        structlog.contextvars.merge_contextvars,
        structlog.processors.add_log_level,
        structlog.processors.TimeStamper(fmt="iso"),
        structlog.dev.ConsoleRenderer() if DEBUG else structlog.processors.JSONRenderer()
    ]
)

# Gắn correlation ID vào toàn bộ request context
def handle_request(request):
    correlation_id = request.headers.get("X-Correlation-ID", str(uuid.uuid4()))
    structlog.contextvars.bind_contextvars(
        correlation_id=correlation_id,
        user_id=request.user.id,
        request_id=str(uuid.uuid4()),
    )
    log.info("request.started", path=request.path, method=request.method)
```

1.3 Correlation IDs

```
# Middleware truyền correlation ID qua các services
import httpx

class CorrelationMiddleware:
    def __init__(self, app):
        self.app = app

    def __call__(self, environ, start_response):
        correlation_id = environ.get("HTTP_X_CORRELATION_ID", str(uuid.uuid4()))
        # Lưu vào thread-local/context var để dùng khi gọi downstream
        request_context.correlation_id = correlation_id
        return self.app(environ, start_response)

# Khi gọi service khác, luôn forward correlation ID
def call_payment_service(order_id: str) -> dict:
    return httpx.post(
        "http://payment-service/charge",
        json={"order_id": order_id},
        headers={"X-Correlation-ID": request_context.correlation_id}
    ).json()
```

---

2. ELK Stack

2.1 Architecture

```
Applications → Filebeat → Logstash → Elasticsearch → Kibana
                                ↑
                          (filter/parse/enrich)
```

2.2 Elasticsearch - Indices và ILM

```
// Index template cho logs
PUT _index_template/app-logs
{
  "index_patterns": ["app-logs-*"],
  "template": {
    "settings": {
      "number_of_shards": 3,
      "number_of_replicas": 1,
      "index.lifecycle.name": "app-logs-policy",
      "index.lifecycle.rollover_alias": "app-logs"
    },
    "mappings": {
      "properties": {
        "@timestamp":    { "type": "date" },
        "level":         { "type": "keyword" },
        "service":       { "type": "keyword" },
        "trace_id":      { "type": "keyword" },
        "user_id":       { "type": "keyword" },
        "message":       { "type": "text", "fields": { "keyword": { "type": "keyword" } } },
        "duration_ms":   { "type": "float" },
        "status_code":   { "type": "integer" }
      }
    }
  }
}
```

```
// Index Lifecycle Management (ILM) policy
PUT _ilm/policy/app-logs-policy
{
  "policy": {
    "phases": {
      "hot": {
        "actions": {
          "rollover": { "max_size": "50GB", "max_age": "1d" },
          "set_priority": { "priority": 100 }
        }
      },
      "warm": {
```

```
        "min_age": "7d",
        "actions": {
          "forcemerge": { "max_num_segments": 1 },
          "shrink": { "number_of_shards": 1 },
          "set_priority": { "priority": 50 }
        }
      },
      "cold": {
        "min_age": "30d",
        "actions": {
          "freeze": {},
          "set_priority": { "priority": 0 }
        }
      },
      "delete": {
        "min_age": "90d",
        "actions": { "delete": {} }
      }
    }
  }
}
```

2.3 Logstash Pipeline

```
# /etc/logstash/conf.d/app-logs.conf

input {
  beats {
    port => 5044
    ssl => true
    ssl_certificate => "/etc/logstash/ssl/logstash.crt"
    ssl_key => "/etc/logstash/ssl/logstash.key"
  }
  kafka {
    bootstrap_servers => "kafka:9092"
    topics => ["app-logs"]
    group_id => "logstash-consumer"
    codec => "json"
  }
}

filter {
  # Parse JSON logs
  if [message] =~ /^\{/ {
    json { source => "message" }
  }
```

```
  # Grok pattern cho nginx access logs
  if [fields][log_type] == "nginx" {
    grok {
      match => {
        "message" => '%{IPORHOST:client_ip} - %{USER:auth} \[%{HTTPDATE:timestamp}\] "%{WORD:method}
        ↪  %{URIPATHPARAM:request} HTTP/%{NUMBER:http_version}" %{NUMBER:status_code:int}
        ↪  %{NUMBER:bytes:int} "%{DATA:referrer}" "%{DATA:user_agent}" %{NUMBER:response_time:float}'
      }
    }
    date { match => ["timestamp", "dd/MMM/yyyy:HH:mm:ss Z"] }
  }

  # Enrich với GeoIP
  if [client_ip] {
    geoip {
      source => "client_ip"
      target => "geoip"
      fields => ["country_name", "city_name", "location"]
    }
  }

  # Loại bỏ health check logs
  if [request] =~ "/health" or [request] =~ "/readyz" {
    drop {}
  }

  # Mask sensitive data - KHÔNG log PII
  mutate {
    gsub => [
      "message", '"password"\s*:\s*"[^"]*"', '"password": "[REDACTED]"',
      "message", '"credit_card"\s*:\s*"[^"]*"', '"credit_card": "[REDACTED]"'
    ]
  }

  # Add metadata
  mutate {
    add_field => {
      "[@metadata][index]" => "app-logs-%{+YYYY.MM.dd}"
    }
    remove_field => ["host", "agent", "ecs", "input"]
  }
}

output {
  elasticsearch {
    hosts => ["https://elasticsearch:9200"]
    user => "${ES_USER}"
    password => "${ES_PASSWORD}"
```

```
    ssl => true
    cacert => "/etc/logstash/ssl/ca.crt"
    index => "%{[@metadata][index]}"
    ilm_enabled => true
    ilm_rollover_alias => "app-logs"
    ilm_policy => "app-logs-policy"
  }

  # Dead letter queue cho failed events
  if "_grokparsefailure" in [tags] {
    file { path => "/var/log/logstash/failed-events.log" }
  }
}
```

2.4 Filebeat Configuration

```
# filebeat.yml
filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - /var/log/apps/*.log
      - /var/log/apps/**/*.log
    json.keys_under_root: true
    json.add_error_key: true
    fields:
      log_type: app
      environment: production
    fields_under_root: true
    multiline.pattern: '^\d{4}-\d{2}-\d{2}'
    multiline.negate: true
    multiline.match: after

  - type: container
    paths:
      - /var/lib/docker/containers/*/*.log
    processors:
      - add_docker_metadata:
          host: "unix:///var/run/docker.sock"

processors:
  - add_host_metadata: ~
  - add_cloud_metadata: ~
  - drop_fields:
      fields: ["log.offset", "agent.ephemeral_id"]

output.logstash:
```

```
  hosts: ["logstash:5044"]
  ssl.certificate_authorities: ["/etc/filebeat/ssl/ca.crt"]
  bulk_max_size: 2048

monitoring.enabled: true
monitoring.elasticsearch:
  hosts: ["https://elasticsearch:9200"]
```

---

3. Grafana Loki

Loki được thiết kế theo kiểu Prometheus cho logs: chỉ index labels, không index content. Rẻ hơn Elasticsearch, tích hợp tốt với Grafana.

3.1 Architecture

```
Promtail/Fluentd → Loki Distributor → Ingester → Object Storage (S3/GCS)
                                                          ↑
Grafana Explore ←── Loki Query Frontend ←── Querier ──────┘
```

3.2 Loki Configuration

```
# loki-config.yml
auth_enabled: false

server:
  http_listen_port: 3100
  grpc_listen_port: 9096

ingester:
  chunk_idle_period: 5m
  chunk_retain_period: 30s
  max_chunk_age: 1h
  lifecycler:
    ring:
      kvstore:
        store: consul
      replication_factor: 3

schema_config:
  configs:
    - from: 2023-01-01
      store: boltdb-shipper
      object_store: s3
      schema: v11
      index:
```

```
        prefix: loki_index_
        period: 24h

storage_config:
  boltdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
  aws:
    s3: s3://ap-southeast-1/company-loki-logs
    region: ap-southeast-1

limits_config:
  ingestion_rate_mb: 10
  ingestion_burst_size_mb: 20
  max_query_parallelism: 32
  retention_period: 744h  # 31 days

compactor:
  working_directory: /loki/compactor
  shared_store: s3
  retention_enabled: true
  retention_delete_delay: 2h
```

3.3 LogQL - Loki Query Language

```
# Log stream selector (chọn logs theo labels)
{service="order-service", environment="production"}

# Filter bằng regex
{service="order-service"} |= "ERROR"
{service="order-service"} != "health"
{service="order-service"} |~ "timeout|connection refused"

# Parse JSON logs
{service="order-service"}
  | json
  | level="ERROR"
  | line_format "{{.message}} (trace: {{.trace_id}})"

# Metric queries từ logs
# Error rate theo service
sum by (service) (
  rate({environment="production"} | json | level="ERROR" [5m])
)

# P99 latency từ logs (nếú log duration_ms)
quantile_over_time(0.99,
```

```
  {service="api-gateway"} | json | unwrap duration_ms [5m]
) by (endpoint)

# Top 10 slowest endpoints
topk(10,
  avg_over_time(
    {service="api-gateway"} | json | unwrap duration_ms [1h]
  ) by (path)
)
```

3.4 Promtail - Log Collector cho Loki

```
# promtail-config.yml
server:
  http_listen_port: 9080

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://loki:3100/loki/api/v1/push
    batchwait: 1s
    batchsize: 1048576
    timeout: 10s

scrape_configs:
  - job_name: kubernetes-pods
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_label_app]
        target_label: service
      - source_labels: [__meta_kubernetes_namespace]
        target_label: namespace
      - source_labels: [__meta_kubernetes_pod_name]
        target_label: pod
    pipeline_stages:
      - json:
          expressions:
            level: level
            trace_id: trace_id
            message: message
      - labels:
          level:
          trace_id:
      - timestamp:
          source: timestamp
```

```
          format: RFC3339Nano
      - output:
          source: message

  - job_name: system-logs
    static_configs:
      - targets: [localhost]
        labels:
          job: system
          __path__: /var/log/{syslog,messages,auth.log}
    pipeline_stages:
      - regex:
          expression: '^(?P<timestamp>\w+ \d+ \d+:\d+:\d+) (?P<host>\S+) (?P<service>\S+):
↪  (?P<message>.*)'
      - labels:
          service:
          host:
```

---

4. Fluentd và Fluent Bit

4.1 Fluent Bit - Lightweight Collector (DaemonSet trên K8s)

```
# fluent-bit-config.yml
[SERVICE]
    Flush         5
    Daemon        Off
    Log_Level     info
    Parsers_File  parsers.conf

[INPUT]
    Name              tail
    Path              /var/log/containers/*.log
    Parser            docker
    Tag               kube.*
    Refresh_Interval  5
    Mem_Buf_Limit     50MB
    Skip_Long_Lines   On

[FILTER]
    Name                kubernetes
    Match               kube.*
    Kube_URL            https://kubernetes.default.svc:443
    Kube_CA_File        /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
    Kube_Token_File     /var/run/secrets/kubernetes.io/serviceaccount/token
    Merge_Log           On
```

```
    Labels              On
    Annotations         Off

[FILTER]
    Name    grep
    Match   kube.*
    Exclude log /health|/readyz|/livez

[OUTPUT]
    Name            loki
    Match           *
    Host            loki
    Port            3100
    Labels          job=fluentbit, namespace=$kubernetes['namespace_name'],
↪   service=$kubernetes['labels']['app']
    Auto_Kubernetes_Labels On

[OUTPUT]
    Name  es
    Match kube.*
    Host  elasticsearch
    Port  9200
    HTTP_User ${ES_USER}
    HTTP_Passwd ${ES_PASSWORD}
    TLS   On
    Index app-logs
    Type  _doc
    Suppress_Type_Name On
```

---

5. Log Aggregation Patterns trên Kubernetes

5.1 DaemonSet Pattern (Recommended)

```
# Fluent Bit chạy trên mỗi node
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluent-bit
  namespace: logging
spec:
  selector:
    matchLabels:
      app: fluent-bit
  template:
    spec:
```

```
      serviceAccountName: fluent-bit
      tolerations:
        - key: node-role.kubernetes.io/master
          effect: NoSchedule
      containers:
        - name: fluent-bit
          image: fluent/fluent-bit:2.2.0
          resources:
            limits:
              memory: 200Mi
              cpu: 200m
            requests:
              memory: 100Mi
              cpu: 100m
          volumeMounts:
            - name: varlog
              mountPath: /var/log
            - name: varlibdockercontainers
              mountPath: /var/lib/docker/containers
              readOnly: true
            - name: config
              mountPath: /fluent-bit/etc/
      volumes:
        - name: varlog
          hostPath:
            path: /var/log
        - name: varlibdockercontainers
          hostPath:
            path: /var/lib/docker/containers
        - name: config
          configMap:
            name: fluent-bit-config
```

5.2 Sidecar Pattern (khi cần log format khác nhau)

```
# Pod với sidecar log shipper
spec:
  containers:
    - name: app
      image: my-app:1.0
      volumeMounts:
        - name: log-volume
          mountPath: /var/log/app

    - name: log-shipper
      image: fluent/fluent-bit:2.2.0
      volumeMounts:
```

```
        - name: log-volume
          mountPath: /var/log/app
          readOnly: true
        - name: shipper-config
          mountPath: /fluent-bit/etc/

  volumes:
    - name: log-volume
      emptyDir: {}
    - name: shipper-config
      configMap:
        name: sidecar-fluent-bit-config
```

---

6. Log Retention và Compliance

```
# Retention strategy theo loại data
Production application logs:  30 ngày hot, 90 ngày warm, 1 năm cold
Security/audit logs:          7 năm (PCI-DSS, SOC2 requirement)
Debug logs:                   7 ngày maximum
Access logs:                  90 ngày (GDPR compliance)

# S3 Lifecycle cho archived logs
aws s3api put-bucket-lifecycle-configuration \
  --bucket company-logs-archive \
  --lifecycle-configuration '{
    "Rules": [{
      "ID": "logs-tiering",
      "Status": "Enabled",
      "Filter": {"Prefix": ""},
      "Transitions": [
        {"Days": 30, "StorageClass": "STANDARD_IA"},
        {"Days": 90, "StorageClass": "GLACIER"},
        {"Days": 365, "StorageClass": "DEEP_ARCHIVE"}
      ],
      "Expiration": {"Days": 2555}
    }]
  }'
```

---

7. Best Practices - Security và PII

```
# Danh sách fields cần REDACT trước khi log
SENSITIVE_FIELDS = {
```

```
    "password", "secret", "token", "api_key", "credit_card",
    "ssn", "date_of_birth", "phone_number", "email"
}

def sanitize_log_data(data: dict) -> dict:
    """Đệ quy xóa sensitive fields khỏi log data."""
    if not isinstance(data, dict):
        return data
    return {
        k: "[REDACTED]" if k.lower() in SENSITIVE_FIELDS else sanitize_log_data(v)
        for k, v in data.items()
    }

# GDPR: hash user IDs thay vì log raw
import hashlib
def anonymize_user_id(user_id: str, salt: str) -> str:
    return hashlib.sha256(f"{salt}{user_id}".encode()).hexdigest()[:16]

# Log request/response - KHÔNG log request body nếú có auth data
@app.middleware("http")
async def log_requests(request: Request, call_next):
    log.info("http.request", **{
        "method": request.method,
        "path": request.url.path,
        # KHÔNG log: request.body(), request.headers["Authorization"]
        "user_agent": request.headers.get("user-agent"),
        "correlation_id": request.headers.get("x-correlation-id"),
    })
    response = await call_next(request)
    log.info("http.response", **{
        "status_code": response.status_code,
        "path": request.url.path,
    })
    return response
```

---

8. Ví dụ: ELK Stack và Loki Stack Deployment

```
# docker-compose.yml - ELK + Loki hybrid stack
version: "3.8"
services:
  # ELK Stack
  elasticsearch:
    image: elasticsearch:8.11.1
    environment:
      - discovery.type=single-node
```

```
    - xpack.security.enabled=true
    - ELASTIC_PASSWORD=changeme
    - "ES_JAVA_OPTS=-Xms2g -Xmx2g"
  volumes:
    - es_data:/usr/share/elasticsearch/data
  ports:
    - "9200:9200"
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:9200/_cluster/health"]
    interval: 30s

logstash:
  image: logstash:8.11.1
  volumes:
    - ./logstash/pipeline:/usr/share/logstash/pipeline
    - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml
  environment:
    - "LS_JAVA_OPTS=-Xmx1g -Xms1g"
    - ES_USER=elastic
    - ES_PASSWORD=changeme
  depends_on:
    elasticsearch:
      condition: service_healthy

kibana:
  image: kibana:8.11.1
  environment:
    - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    - ELASTICSEARCH_USERNAME=kibana_system
    - ELASTICSEARCH_PASSWORD=changeme
  ports:
    - "5601:5601"

# Loki Stack
loki:
  image: grafana/loki:2.9.3
  volumes:
    - ./loki/config.yml:/etc/loki/config.yml
    - loki_data:/loki
  command: -config.file=/etc/loki/config.yml
  ports:
    - "3100:3100"

promtail:
  image: grafana/promtail:2.9.3
  volumes:
    - ./promtail/config.yml:/etc/promtail/config.yml
    - /var/log:/var/log:ro
```

```
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
    command: -config.file=/etc/promtail/config.yml
    depends_on:
      - loki

  grafana:
    image: grafana/grafana:10.2.2
    volumes:
      - grafana_data:/var/lib/grafana
      - ./grafana/provisioning:/etc/grafana/provisioning
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    ports:
      - "3000:3000"

volumes:
  es_data:
  loki_data:
  grafana_data:
```

```
# Kiểm tra Elasticsearch health
curl -u elastic:changeme http://localhost:9200/_cluster/health?pretty

# Xem indices
curl -u elastic:changeme http://localhost:9200/_cat/indices?v

# Query Loki từ CLI
logcli --addr=http://localhost:3100 \
  query '{service="order-service"}' \
  --since=1h \
  --limit=100

# Xem labels
logcli --addr=http://localhost:3100 labels

# Tail logs real-time
logcli --addr=http://localhost:3100 \
  query '{service="order-service"} |= "ERROR"' \
  --tail
```

Tóm tắt

Công nghệ	Khi nào dùng	Trade-off
ELK Stack	Cần full-text search mạnh, complex queries	Chi phí cao, ops phức tạp

Công nghệ	Khi nào dùng	Trade-off
Grafana Loki	Tích hợp Prometheus/Grafana, cost-sensitive	Query kém linh hoạt hơn ES
Fluentd	Nhiều input/output, cần transform phức tạp	Tốn memory hơn Fluent Bit
Fluent Bit	K8s DaemonSet, resource-constrained	Ít plugins hơn Fluentd
Filebeat	Đơn giản, chỉ ship to ELK	Ít tính năng xử lý

Key takeaway: Structured JSON logs + correlation IDs là nền tảng. ELK cho compliance và full-text search; Loki cho cost-effective logs khi đã có Grafana stack. Luôn redact PII trước khi gửi log đến bất kỳ storage nào.

Chapter 20: Cloud Providers

Cloud providers cung cấp infrastructure theo mô hình pay-as-you-go, loại bỏ chi phí capex và cho phép scale nhanh. Hiểu rõ core services của từng provider, khi nào dùng managed vs self-managed, và cách tối ưu chi phí là kỹ năng thiết yếu.

---

1. AWS - Amazon Web Services

1.1 Core Compute và Storage

EC2 - Elastic Compute Cloud

```
# Launch instance
aws ec2 run-instances \
  --image-id ami-0abcdef1234567890 \
  --instance-type t3.medium \
  --key-name my-keypair \
  --security-group-ids sg-12345678 \
  --subnet-id subnet-12345678 \
  --iam-instance-profile Name=my-ec2-role \
  --tag-specifications
  ↪  'ResourceType=instance,Tags=[{Key=Name,Value=app-server},{Key=Environment,Value=production}]' \
  --user-data file://bootstrap.sh

# List instances với filter
aws ec2 describe-instances \
  --filters "Name=tag:Environment,Values=production" "Name=instance-state-name,Values=running" \
  --query 'Reserva-
  ↪  tions[].Instances[].[InstanceId,InstanceType,PrivateIpAddress,Tags[?Key==`Name`].Value|[0]]'
  ↪  \
  --output table
```

S3 - Simple Storage Service

```
# Tạo bucket với versioning và encryption
aws s3api create-bucket \
  --bucket company-app-assets \
  --region ap-southeast-1 \
  --create-bucket-configuration LocationConstraint=ap-southeast-1

aws s3api put-bucket-versioning \
  --bucket company-app-assets \
  --versioning-configuration Status=Enabled

aws s3api put-bucket-encryption \
  --bucket company-app-assets \
  --server-side-encryption-configuration '{
    "Rules": [{"ApplyServerSideEncryptionByDefault": {"SSEAlgorithm": "aws:kms"}}]
  }'

# Sync thư mục lên S3
aws s3 sync ./dist s3://company-app-assets/frontend/ \
  --delete \
  --cache-control "max-age=31536000" \
  --exclude "*.html" \
  --include "*.js" --include "*.css"

# Presigned URL cho private objects
aws s3 presign s3://company-private/report.pdf --expires-in 3600
```

RDS - Relational Database Service

```
# Tạo RDS PostgreSQL instance
aws rds create-db-instance \
  --db-instance-identifier prod-postgres \
  --db-instance-class db.r6g.xlarge \
  --engine postgres \
  --engine-version 15.4 \
  --master-username admin \
  --master-user-password "${DB_PASSWORD}" \
  --allocated-storage 100 \
  --storage-type gp3 \
  --storage-encrypted \
  --multi-az \
  --backup-retention-period 7 \
  --deletion-protection \
  --db-subnet-group-name prod-db-subnet-group \
  --vpc-security-group-ids sg-database

# Tạo read replica
aws rds create-db-instance-read-replica \
```

```
  --db-instance-identifier prod-postgres-replica \
  --source-db-instance-identifier prod-postgres \
  --db-instance-class db.r6g.large
```

1.2 AWS Networking

VPC Design - 3-tier architecture

```
VPC: 10.0.0.0/16
├── Public Subnets (ALB, NAT Gateway, Bastion)
│   ├── 10.0.1.0/24  (AZ-a)
│   └── 10.0.2.0/24  (AZ-b)
├── Private Subnets (App Servers, EKS Nodes)
│   ├── 10.0.10.0/24 (AZ-a)
│   └── 10.0.11.0/24 (AZ-b)
└── Database Subnets (RDS, ElastiCache)
    ├── 10.0.20.0/24 (AZ-a)
    └── 10.0.21.0/24 (AZ-b)
```

```
# VPC Peering - kết nối 2 VPC
aws ec2 create-vpc-peering-connection \
  --vpc-id vpc-prod \
  --peer-vpc-id vpc-staging \
  --peer-region ap-southeast-1

# Transit Gateway - hub cho nhiều VPCs
aws ec2 create-transit-gateway \
  --description "Central hub for all VPCs" \
  --options AmazonSideAsn=64512,AutoAcceptSharedAttachments=enable

# PrivateLink - expose service privately
aws ec2 create-vpc-endpoint-service-configuration \
  --network-load-balancer-arns arn:aws:elasticloadbalancing:...:loadbalancer/net/internal-nlb/xxx
```

1.3 EKS - Elastic Kubernetes Service

```
# Tạo EKS cluster với eksctl
cat > cluster.yaml << 'EOF'
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: production
  region: ap-southeast-1
  version: "1.28"

vpc:
  id: vpc-12345678
```

```
  subnets:
    private:
      ap-southeast-1a: {id: subnet-private-a}
      ap-southeast-1b: {id: subnet-private-b}

managedNodeGroups:
  - name: general
    instanceType: m6i.xlarge
    minSize: 3
    maxSize: 20
    desiredCapacity: 5
    privateNetworking: true
    volumeSize: 100
    volumeType: gp3
    iam:
      withAddonPolicies:
        autoScaler: true
        albIngress: true
        cloudWatch: true
    tags:
      Environment: production

  - name: spot
    instanceTypes: [m6i.xlarge, m6a.xlarge, m5.xlarge]
    spot: true
    minSize: 0
    maxSize: 50
    desiredCapacity: 0

addons:
  - name: vpc-cni
    version: latest
  - name: coredns
    version: latest
  - name: kube-proxy
    version: latest
  - name: aws-ebs-csi-driver
    version: latest
    wellKnownPolicies:
      ebsCSIController: true
EOF

eksctl create cluster -f cluster.yaml
```

1.4 IAM Best Practices

```
// Least privilege policy - chỉ cho phép đọc S3 bucket cụ thể
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadAppAssets",
      "Effect": "Allow",
      "Action": ["s3:GetObject", "s3:ListBucket"],
      "Resource": [
        "arn:aws:s3:::company-app-assets",
        "arn:aws:s3:::company-app-assets/*"
      ],
      "Condition": {
        "StringEquals": {"aws:RequestedRegion": "ap-southeast-1"}
      }
    }
  ]
}
```

```
# IAM Role cho EC2 (Instance Profile)
aws iam create-role \
  --role-name AppServerRole \
  --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [{
      "Effect": "Allow",
      "Principal": {"Service": "ec2.amazonaws.com"},
      "Action": "sts:AssumeRole"
    }]
  }'

# OIDC federation cho Kubernetes ServiceAccount
eksctl create iamserviceaccount \
  --cluster=production \
  --namespace=default \
  --name=app-service-account \
  --attach-policy-arn=arn:aws:iam::123456789:policy/AppPolicy \
  --override-existing-serviceaccounts \
  --approve
```

---

2. GCP - Google Cloud Platform

2.1 Core Services

```
# GKE - Google Kubernetes Engine
gcloud container clusters create production \
  --region asia-southeast1 \
  --node-locations asia-southeast1-a,asia-southeast1-b \
  --machine-type n2-standard-4 \
  --num-nodes 3 \
  --min-nodes 3 \
  --max-nodes 20 \
  --enable-autoscaling \
  --enable-autorepair \
  --enable-autoupgrade \
  --workload-pool=$(gcloud config get-value project).svc.id.goog \
  --network vpc-production \
  --subnetwork subnet-gke \
  --enable-private-nodes \
  --master-ipv4-cidr 172.16.0.0/28

# Cloud Storage
gsutil mb -l asia-southeast1 -c STANDARD gs://company-production-assets
gsutil iam ch serviceAccount:app@project.iam.gserviceaccount.com:roles/storage.objectViewer
 ↪  gs://company-production-assets

# Cloud Run - serverless containers
gcloud run deploy order-service \
  --image gcr.io/project/order-service:v1.2.3 \
  --region asia-southeast1 \
  --platform managed \
  --min-instances 1 \
  --max-instances 100 \
  --concurrency 80 \
  --cpu 2 \
  --memory 1Gi \
  --service-account order-service@project.iam.gserviceaccount.com \
  --set-env-vars DB_HOST=10.0.1.5 \
  --no-allow-unauthenticated

# BigQuery - analytics
bq query --use_legacy_sql=false \
  'SELECT DATE(timestamp) as date, COUNT(*) as orders, SUM(amount) as revenue
   FROM `project.dataset.orders`
   WHERE timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 30 DAY)
   GROUP BY date
   ORDER BY date DESC'
```

2.2 GCP IAM và Workload Identity

```
# Workload Identity - pod dùng Google Service Account
gcloud iam service-accounts create app-service-account \
  --display-name "App Service Account"

gcloud projects add-iam-policy-binding PROJECT_ID \
  --member "serviceAccount:app-service-account@PROJECT_ID.iam.gserviceaccount.com" \
  --role "roles/cloudsql.client"

# Bind KSA → GSA
gcloud iam service-accounts add-iam-policy-binding \
  app-service-account@PROJECT_ID.iam.gserviceaccount.com \
  --role roles/iam.workloadIdentityUser \
  --member "serviceAccount:PROJECT_ID.svc.id.goog[namespace/ksa-name]"
```

---

3. Azure

```
# AKS - Azure Kubernetes Service
az aks create \
  --resource-group production-rg \
  --name production-aks \
  --location southeastasia \
  --node-count 3 \
  --node-vm-size Standard_D4s_v3 \
  --enable-cluster-autoscaler \
  --min-count 3 \
  --max-count 20 \
  --network-plugin azure \
  --network-policy azure \
  --vnet-subnet-id /subscriptions/.../subnets/aks-subnet \
  --enable-managed-identity \
  --enable-addons monitoring \
  --workspace-resource-id /subscriptions/.../workspaces/aks-logs

# Azure Blob Storage
az storage account create \
  --name companyprodstorage \
  --resource-group production-rg \
  --location southeastasia \
  --sku Standard_LRS \
  --kind StorageV2 \
  --min-tls-version TLS1_2 \
  --https-only true
```

```
# Azure Functions
az functionapp create \
  --resource-group production-rg \
  --consumption-plan-location southeastasia \
  --runtime python \
  --runtime-version 3.11 \
  --functions-version 4 \
  --name company-processing-fn \
  --storage-account companyprodstorage
```

---

4. Multi-cloud Strategy

4.1 Pros và Cons

```
PROS multi-cloud:
+ Tránh vendor lock-in
+ Best-of-breed: GCP BigQuery + AWS EKS + Cloudflare CDN
+ Redundancy và disaster recovery
+ Geo-specific compliance (data residency)
+ Bargaining power với vendors

CONS multi-cloud:
- Tăng complexity vận hành đáng kể
- Chi phí training team cao
- Networking giữa clouds: latency + egress cost
- Security perimeter khó kiểm soát hơn
- Tooling bị phân mảnh
```

4.2 Abstraction Layers

```
# Terraform - Infrastructure as Code đa cloud
# modules/kubernetes-cluster/main.tf

variable "provider" { type = string }  # aws, gcp, azure
variable "cluster_name" { type = string }
variable "node_count" { type = number }

module "eks" {
  count  = var.provider == "aws" ? 1 : 0
  source = "./providers/aws-eks"
  name   = var.cluster_name
  nodes  = var.node_count
}

module "gke" {
```

```
  count  = var.provider == "gcp" ? 1 : 0
  source = "./providers/gcp-gke"
  name   = var.cluster_name
  nodes  = var.node_count
}

# Crossplane - K8s-native multi-cloud provisioning
apiVersion: eks.aws.crossplane.io/v1beta1
kind: Cluster
metadata:
  name: production-east
spec:
  forProvider:
    region: us-east-1
    version: "1.28"
  providerConfigRef:
    name: aws-provider
```

---

5. Cost Optimization

5.1 Reserved Instances và Savings Plans

```
# AWS - mua Reserved Instance 1 năm (save 30-40%)
aws ec2 purchase-reserved-instances-offering \
  --reserved-instances-offering-id <offering-id> \
  --instance-count 10

# Savings Plans - flexible hơn RI
aws savingsplans purchase-savings-plan \
  --savings-plan-type ComputeSavingsPlans \
  --commitment 100.00 \
  --purchase-time $(date -u +%Y-%m-%dT%H:%M:%SZ) \
  --term-duration-in-years 1 \
  --payment-option PARTIAL_UPFRONT
```

5.2 Spot/Preemptible Instances

```
# Kubernetes - dùng Spot cho batch workloads
apiVersion: apps/v1
kind: Deployment
metadata:
  name: batch-processor
spec:
  template:
```

```
    spec:
      nodeSelector:
        eks.amazonaws.com/capacityType: SPOT
      tolerations:
        - key: "eks.amazonaws.com/capacityType"
          operator: Equal
          value: SPOT
          effect: NoSchedule
      # Graceful shutdown khi spot bị reclaim (2 min notice)
      terminationGracePeriodSeconds: 120
      containers:
        - name: processor
          lifecycle:
            preStop:
              exec:
                command: ["/bin/sh", "-c", "sleep 10 && checkpoint-state.sh"]
```

5.3 Right-sizing và Cost Monitoring

```
# AWS Cost Explorer - tìm overprovisioned instances
aws ce get-rightsizing-recommendation \
  --service EC2 \
  --configuration RightsizingType=TERMINATE,TerminateRecommendationDetail={EstimatedMonthlyBenefit=50}

# AWS Trusted Advisor
aws support describe-trusted-advisor-checks --language en

# Kubecost - K8s cost allocation
helm repo add kubecost https://kubecost.github.io/cost-analyzer/
helm install kubecost kubecost/cost-analyzer \
  --namespace kubecost \
  --create-namespace \
  --set kubecostToken="${KUBECOST_TOKEN}"

# Tag enforcement - tất cả resources phải có tags
aws config put-config-rule --config-rule '{
  "ConfigRuleName": "required-tags",
  "Source": {
    "Owner": "AWS",
    "SourceIdentifier": "REQUIRED_TAGS"
  },
  "InputParameters": "{\"tag1Key\":\"Environment\",\"tag2Key\":\"Team\",\"tag3Key\":\"CostCenter\"}"
}'
```

---

6. Landing Zone - Account/Project Structure

```
AWS Organizations:
├── Root
│   ├── Management Account (billing, org policies)
│   ├── Security OU
│   │   ├── Security Tooling Account (GuardDuty, Security Hub)
│   │   └── Log Archive Account (CloudTrail, Config)
│   ├── Infrastructure OU
│   │   ├── Shared Services Account (DNS, Monitoring)
│   │   └── Network Account (Transit Gateway, Direct Connect)
│   └── Workloads OU
│       ├── Production OU
│       │   ├── prod-web-account
│       │   └── prod-data-account
│       └── Non-Production OU
│           ├── staging-account
│           └── dev-account
```

```
# AWS Control Tower - guardrails tự động
# Sau khi setup, tạo account mới tự động apply baseline policies

# Service Control Policy (SCP) - chặn regions không dùng
aws organizations create-policy \
  --name DenyUnusedRegions \
  --type SERVICE_CONTROL_POLICY \
  --content '{
    "Version": "2012-10-17",
    "Statement": [{
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:RequestedRegion": ["ap-southeast-1", "us-east-1"]
        }
      }
    }]
  }'
```

---

7. CLI Tools

```
# AWS CLI
aws configure --profile production
export AWS_PROFILE=production
```

```
# Assume role cross-account
aws sts assume-role \
  --role-arn arn:aws:iam::123456789:role/DeployRole \
  --role-session-name deploy-session \
  | jq -r '.Credentials | "export AWS_ACCESS_KEY_ID=\(.AccessKeyId)\nexport
  ↪  AWS_SECRET_ACCESS_KEY=\(.SecretAccessKey)\nexport AWS_SESSION_TOKEN=\(.SessionToken)"'

# GCP CLI
gcloud auth application-default login
gcloud config set project my-project
gcloud config configurations create production
gcloud config set account user@company.com

# Azure CLI
az login --service-principal \
  --username $ARM_CLIENT_ID \
  --password $ARM_CLIENT_SECRET \
  --tenant $ARM_TENANT_ID

az account set --subscription "Production Subscription"
az account list --output table

# aws-vault - quản lý credentials an toàn
brew install aws-vault
aws-vault add production
aws-vault exec production -- aws s3 ls
```

---

8. Ví dụ: AWS Architecture cho Web Application

```
Internet → Route53 (DNS + Health Check)
         → CloudFront (CDN, WAF, DDoS protection)
         → S3 (static assets)
         → ALB (Application Load Balancer)
         → EKS (app containers, private subnet)
         → RDS Aurora PostgreSQL (multi-AZ, private subnet)
         → ElastiCache Redis (session, cache, private subnet)
         → SQS (async job queue)
         → Lambda (event processors)
         → S3 (uploads, backups)

Supporting services:
- ACM: SSL certificates
- Secrets Manager: DB passwords, API keys
- Parameter Store: app config
- KMS: encryption keys
- CloudWatch: logs, metrics, alarms
```

- X-Ray: distributed tracing
- GuardDuty: threat detection
- Config: compliance monitoring
- CloudTrail: audit logs

```
# Triển khai full stack với Terraform
cat > main.tf << 'EOF'
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "5.4.0"

  name = "production"
  cidr = "10.0.0.0/16"

  azs              = ["ap-southeast-1a", "ap-southeast-1b", "ap-southeast-1c"]
  private_subnets  = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets   = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]
  database_subnets = ["10.0.201.0/24", "10.0.202.0/24", "10.0.203.0/24"]

  enable_nat_gateway = true
  single_nat_gateway = false  # HA: NAT gateway mỗi AZ

  enable_flow_log                      = true
  flow_log_cloudwatch_log_group_name   = "/aws/vpc/production"
  create_flow_log_cloudwatch_log_group = true
  create_flow_log_cloudwatch_iam_role  = true
}

module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "20.2.1"

  cluster_name    = "production"
  cluster_version = "1.28"

  vpc_id                   = module.vpc.vpc_id
  subnet_ids               = module.vpc.private_subnets
  control_plane_subnet_ids = module.vpc.private_subnets

  cluster_endpoint_private_access = true
  cluster_endpoint_public_access  = false  # access qua VPN/bastion

  eks_managed_node_groups = {
    general = {
      instance_types = ["m6i.xlarge"]
      min_size       = 3
      max_size       = 20
      desired_size   = 5
```

```
      block_device_mappings = {
        xvda = {
          device_name = "/dev/xvda"
          ebs = { volume_size = 100, volume_type = "gp3" }
        }
      }
    }
  }
}

module "rds" {
  source  = "terraform-aws-modules/rds/aws"
  version = "6.3.1"

  identifier = "production-postgres"
  engine     = "postgres"
  engine_version = "15.4"
  instance_class = "db.r6g.xlarge"

  allocated_storage     = 100
  max_allocated_storage = 1000  # autoscaling storage
  storage_encrypted     = true

  multi_az               = true
  db_subnet_group_name   = module.vpc.database_subnet_group
  vpc_security_group_ids = [aws_security_group.rds.id]

  backup_retention_period = 7
  deletion_protection     = true
  skip_final_snapshot     = false
}
EOF

terraform init
terraform plan -out=tfplan
terraform apply tfplan
```

```
# Health check toàn bộ stack
# Kiểm tra EKS nodes
kubectl get nodes -o wide
kubectl top nodes

# Kiểm tra RDS
aws rds describe-db-instances \
  --db-instance-identifier production-postgres \
  --query 'DBInstances[0].{Status:DBInstanceStatus,MultiAZ:MultiAZ,Endpoint:Endpoint.Address}'

# Kiểm tra ALB target health
```

```
aws elbv2 describe-target-health \
  --target-group-arn arn:aws:elasticloadbalancing:...:targetgroup/app-tg/xxx \
  --query 'TargetHealthDescriptions[].{Target:Target.Id,Status:TargetHealth.State}'

# Cost report tháng này
aws ce get-cost-and-usage \
  --time-period Start=$(date -d "$(date +%Y-%m-01)" +%Y-%m-%d),End=$(date +%Y-%m-%d) \
  --granularity MONTHLY \
  --metrics BlendedCost \
  --group-by Type=DIMENSION,Key=SERVICE \
  --query
  ↪  'ResultsByTime[0].Groups[?Metrics.BlendedCost.Amount>`100`].[Keys[0],Metrics.BlendedCost.Amount]'
  ↪  \
  --output table
```

Tóm tắt

Provider	Thế mạnh	Best for
AWS	Ecosystem rộng nhất, mature services	General purpose, enterprise
GCP	Data/ML (BigQuery, Vertex AI), Kubernetes (GKE tốt nhất)	Analytics, AI/ML workloads
Azure	Tích hợp Microsoft ecosystem (AD, Office365)	Enterprise với Windows workloads
Multi-cloud	Flexibility, avoid lock-in	Large orgs, compliance requirements

Cost optimization priority: 1. Right-size instances trước (dễ nhất, impactful nhất) 2. Reserved/Savings Plans cho baseline load 3. Spot/Preemptible cho non-critical workloads 4. Review và xóa unused resources hàng tuần 5. Set budgets và alerts từ đầu

Key takeaway: Chọn một cloud chính, master nó. Chỉ multi-cloud khi có nhu cầu cụ thể. IAM least privilege và tagging strategy phải làm từ đầu - rất khó retrofit sau.

Chương 21: Cloud Design Patterns

Cloud design patterns là tập hợp các giải pháp kiến trúc đã được kiểm chứng cho các bài toán phổ biến khi xây dựng hệ thống trên cloud. Hiểu và áp dụng đúng patterns giúp xây dựng hệ thống reliable, scalable, và cost-effective.

---

1. High Availability Patterns

1.1 Multi-AZ Deployment

Triển khai ứng dụng trên nhiều Availability Zone (AZ) trong cùng một Region để chịu lỗi ở cấp datacenter.

```
Region: ap-southeast-1
├── AZ-1a: App servers + DB primary
├── AZ-1b: App servers + DB replica
└── AZ-1c: App servers + DB replica

Load Balancer phân phối traffic đến cả 3 AZ
Nếu AZ-1a fail → traffic tự động chuyển sang 1b, 1c
```

AWS example:

```
# Terraform - Multi-AZ RDS
resource "aws_db_instance" "main" {
  identifier           = "prod-db"
  engine               = "mysql"
  instance_class       = "db.r6g.large"
  multi_az             = true  # Tự động tạo standby ở AZ khác
  allocated_storage    = 100
  storage_encrypted    = true
}
```

1.2 Multi-Region Architecture

Triển khai trên nhiều Region để chịu lỗi ở cấp region (thiên tai, sự cố lớn).

```
Active-Active Multi-Region:

┌──────────────────┐     ┌──────────────────┐
│  Region US-East  │ ←→  │  Region EU-West  │
│  (50% traffic)   │     │  (50% traffic)   │
└──────────────────┘     └──────────────────┘
         ↑                        ↑
         └──── Global LB (Route53 / Cloudflare) ────┘

Database: DynamoDB Global Tables / CockroachDB / Spanner
```

1.3 Active-Active vs Active-Passive

Pattern	Mô tả	RTO	Chi phí
Active-Active	Cả 2 site đều xử lý traffic	~0	Cao nhất
Active-Passive	1 site chính, 1 site dự phòng (warm)	5-30 phút	Trung bình

Pilot Light	Site dự phòng ở trạng thái tối thiểu	30-60 phút	Thấp

---

2. Disaster Recovery Patterns

2.1 Các khái niệm cốt lõi

- **RPO (Recovery Point Objective)**: Dữ liệu mất tối đa bao nhiêu trước điểm failover
- **RTO (Recovery Time Objective)**: Thời gian khôi phục hệ thống tối đa sau sự cố

```
Thảm họa xảy ra
       |
       ▼
[Last backup] ←——— RPO ———→ [Disaster] ←— RTO —→ [Recovery]
   (RPO = data loss window)        (RTO = downtime window)
```

2.2 DR Strategies

Backup & Restore (RPO: hours, RTO: hours)

```
# Automated S3 backup với lifecycle policy
aws s3 cp /data/backup.tar.gz s3://dr-bucket/backups/$(date +%Y%m%d)/
# Cost: chỉ trả tiền storage
```

Pilot Light (RPO: minutes, RTO: 30-60 phút)

```
Luôn chạy: Database replication
Tắt khi không cần: App servers (chỉ AMI/snapshot sẵn sàng)
Khi DR: Boot app servers từ AMI → update DNS
```

Warm Standby (RPO: minutes, RTO: 5-15 phút)

```
Luôn chạy ở capacity thấp:
- DR site: 2 app servers (prod: 10 servers)
- Database: replica đang sync liên tục
Khi DR: Scale up DR site → failover DNS
```

Hot Standby / Active-Active (RPO: ~0, RTO: ~0)

```
Cả 2 site full capacity, traffic split 50/50
Database: sync replication (chấp nhận latency cao hơn)
```

2.3 Backup Strategies

```
# Backup với versioning và retention
aws s3api put-bucket-versioning \
  --bucket my-backup-bucket \
  --versioning-configuration Status=Enabled
```

```
# S3 Lifecycle: move to Glacier sau 30 ngày
aws s3api put-bucket-lifecycle-configuration \
  --bucket my-backup-bucket \
  --lifecycle-configuration file://lifecycle.json

# Test restore định kỳ (critical!)
aws rds restore-db-instance-from-db-snapshot \
  --db-instance-identifier test-restore \
  --db-snapshot-identifier prod-snapshot-20260101
```

---

3. Resilience Patterns

3.1 Circuit Breaker

Ngắt kết nối đến service bị lỗi để tránh cascade failure.

```
States:
CLOSED → requests pass through normally
OPEN   → requests fail immediately (không gọi service lỗi)
HALF-OPEN → thử một số requests để test recovery

CLOSED ──(failure threshold exceeded)──→ OPEN
  ↑                                        │
  └──(success)── HALF-OPEN ←──(timeout)────┘
```

```
# Python với pybreaker
import pybreaker

db_breaker = pybreaker.CircuitBreaker(
    fail_max=5,          # 5 lỗi liên tiếp → mở circuit
    reset_timeout=60,    # 60 giây → thử lại
)

@db_breaker
def query_database(sql):
    return db.execute(sql)
```

3.2 Bulkhead Pattern

Cô lập resources để lỗi một phần không ảnh hưởng toàn bộ.

```
Thay vì: 1 thread pool cho tất cả services
         → Service A chậm → chiếm hết thread → Service B cũng chết

Bulkhead: Thread pool riêng cho mỗi service
├── Pool A: 20 threads (Service A)
├── Pool B: 20 threads (Service B)
```

```
└── Pool C: 10 threads (Service C)
```

3.3 Retry with Exponential Backoff

```python
import time
import random

def retry_with_backoff(func, max_retries=5, base_delay=1.0):
    for attempt in range(max_retries):
        try:
            return func()
        except Exception as e:
            if attempt == max_retries - 1:
                raise
            # Exponential backoff + jitter
            delay = base_delay * (2 ** attempt) + random.uniform(0, 1)
            print(f"Attempt {attempt+1} failed, retrying in {delay:.2f}s")
            time.sleep(delay)

# AWS SDK tự động retry với backoff
import boto3
from botocore.config import Config

config = Config(
    retries={"max_attempts": 5, "mode": "adaptive"}
)
s3 = boto3.client("s3", config=config)
```

---

4. Scalability Patterns

4.1 Horizontal vs Vertical Scaling

	Horizontal (Scale Out)	Vertical (Scale Up)
Cách	Thêm servers	Tăng CPU/RAM
Downtime	Không	Thường có
Giới hạn	Gần như vô hạn	Giới hạn phần cứng
Chi phí	Tuyến tính	Exponential
Stateless	Bắt buộc	Không cần

4.2 Auto-Scaling Patterns

```yaml
# Kubernetes HPA - scale theo CPU
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
```

```yaml
metadata:
  name: webapp-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: webapp
  minReplicas: 2
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: External
    external:
      metric:
        name: sqs_queue_depth  # Scale theo queue length
      target:
        type: AverageValue
        averageValue: "100"
```

---

5. Data Patterns

5.1 CQRS (Command Query Responsibility Segregation)

Tách biệt model đọc và ghi dữ liệu.

```
Write side:                      Read side:
Command → Command Handler  →  Event Store → Projections → Read Model
          (validate + save)     (events)     (materialize)  (optimized views)
```

5.2 Event Sourcing

Lưu trữ tất cả thay đổi dưới dạng events thay vì state hiện tại.

```python
# Thay vì: UPDATE account SET balance = 1500
# Event sourcing:
events = [
    {"type": "AccountCreated", "balance": 0},
    {"type": "MoneyDeposited", "amount": 2000},
    {"type": "MoneyWithdrawn", "amount": 500},
]
# Current state = replay tất cả events
```

```
current_balance = sum(e.get("amount", 0) * (1 if e["type"] == "MoneyDeposited" else -1)
                      for e in events if "amount" in e)  # = 1500
```

5.3 Saga Pattern

Quản lý distributed transactions qua nhiều services.

```
Choreography Saga (Event-driven):
OrderService → emit OrderCreated
  → PaymentService: reserve payment → emit PaymentReserved
  → InventoryService: reserve stock → emit StockReserved
  → ShippingService: create shipment → emit OrderCompleted

Compensating transactions khi lỗi:
StockReserved failed → emit StockReservationFailed
  → PaymentService: release payment (compensation)
  → OrderService: cancel order (compensation)
```

---

6. Deployment Patterns

6.1 Blue/Green Deployment

```
# Nginx blue/green switch
# Blue (current production): port 8080
# Green (new version): port 8081

# Switch traffic đến Green
sed -i 's/proxy_pass http:\/\/localhost:8080/proxy_pass http:\/\/localhost:8081/' /etc/nginx/nginx.conf
nginx -s reload

# Nếu có vấn đề: rollback ngay lập tức
sed -i 's/proxy_pass http:\/\/localhost:8081/proxy_pass http:\/\/localhost:8080/' /etc/nginx/nginx.conf
nginx -s reload
```

6.2 Canary Deployment

```
# Istio VirtualService: 5% traffic đến canary
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: webapp
spec:
  http:
  - route:
    - destination:
```

```
        host: webapp
        subset: stable
      weight: 95
    - destination:
        host: webapp
        subset: canary
      weight: 5  # Tăng dần: 5 → 20 → 50 → 100
```

6.3 Feature Flags

```python
# LaunchDarkly / Unleash / homemade flags
import os

def is_feature_enabled(feature_name: str, user_id: str = None) -> bool:
    # Check environment variable
    if os.environ.get(f"FEATURE_{feature_name.upper()}") == "true":
        return True
    # Check percentage rollout
    if user_id:
        return hash(user_id) % 100 < get_rollout_percentage(feature_name)
    return False

# Usage
if is_feature_enabled("new_checkout", user.id):
    return new_checkout_flow()
else:
    return legacy_checkout_flow()
```

---

7. Twelve-Factor App

Factor	Mô tả	Thực tế
Codebase	1 repo, nhiều deployments	Git monorepo hoặc polyrepo
Dependencies	Khai báo tường minh	requirements.txt, package.json
Config	Lưu trong environment	.env, Kubernetes ConfigMap
Backing services	Treat như attached resources	DATABASE_URL env var
Build/Release/Run	Tách biệt 3 giai đoạn	CI/CD pipeline
Processes	Stateless, share-nothing	Sessions trong Redis
Port binding	Export service qua port	Listen trên PORT env var
Concurrency	Scale theo process model	Horizontal scaling
Disposability	Fast startup/graceful shutdown	SIGTERM handling

Factor	Mô tả	Thực tế
Dev/prod parity	Dev giống prod nhất có thể	Docker Compose
Logs	Treat logs như event streams	stdout → log aggregator
Admin processes	One-off tasks	`kubectl exec` / job

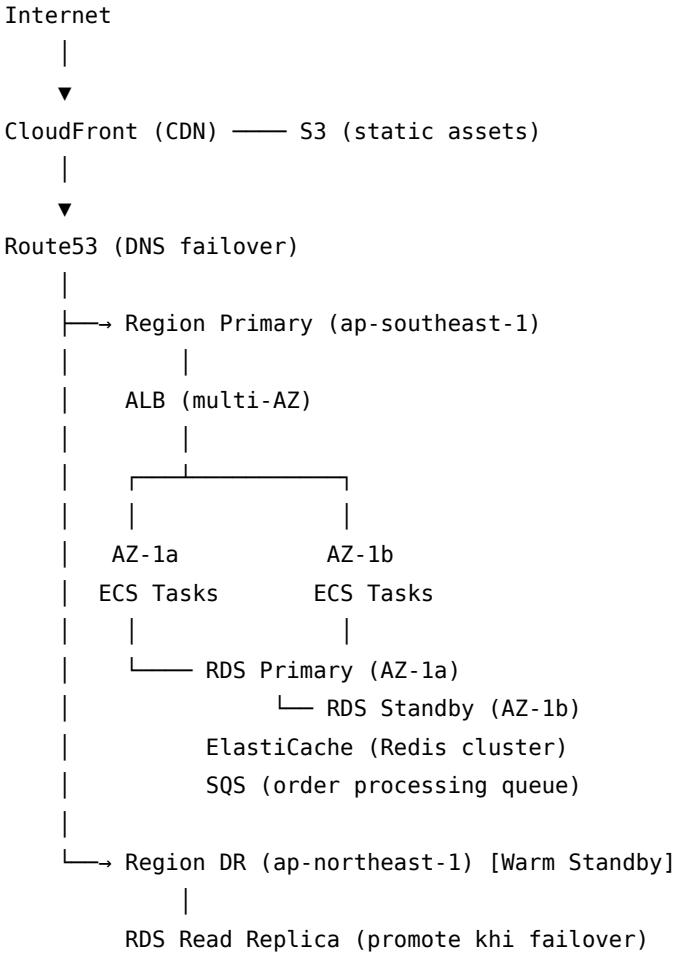
---

8. AWS Well-Architected Framework: 6 Pillars

1. **Operational Excellence**: Automate everything, learn from failure
2. **Security**: Defense in depth, least privilege, encryption everywhere
3. **Reliability**: Automatic recovery, scale horizontally, stop guessing capacity
4. **Performance Efficiency**: Use managed services, serverless, go global in minutes
5. **Cost Optimization**: Pay for what you use, right-sizing, reserved instances
6. **Sustainability**: Maximize utilization, use efficient hardware, minimize footprint

---

9. Ví dụ: HA Architecture cho E-Commerce Platform



```
        ECS Tasks (scale up khi DR activated)

Monitoring:
- CloudWatch Alarms → SNS → PagerDuty
- Health checks mỗi 30 giây
- Auto failover khi primary unhealthy > 2 phút
```

RPO: 1 phút (RDS automated backup + binlog) **RTO: 5 phút** (Route53 health check TTL 60s + warm standby boot)

Chương 22: Artifact Management

Artifact management là việc lưu trữ, versioning, và phân phối các build artifacts (container images, packages, binaries, Helm charts). Một hệ thống artifact management tốt đảm bảo traceability, security, và reliability của software supply chain.

---

1. Artifact Types

Loại	Ví dụ	Format	Registry phổ biến
Container images	Docker, OCI	tar layers	Docker Hub, Harbor, ECR
Java packages	JAR, WAR, EAR	Maven/Gradle	Maven Central, Nexus
Node packages	npm modules	tarball	npm registry, Verdaccio
Python packages	wheel, sdist	PEP 517/518	PyPI, Nexus, Artifactory
Helm charts	Kubernetes apps	tar.gz	Artifact Hub, Harbor
Binary artifacts	CLI tools, scripts	platform-specific	GitHub Releases, Nexus
OS packages	DEB, RPM	apt/yum	APT repo, YUM repo

---

2. Nexus Repository Manager

Nexus hỗ trợ proxy (cache từ upstream), hosted (lưu internal artifacts), và group (aggregate nhiều repos).

2.1 Cài đặt Nexus với Docker

```
docker run -d \
  --name nexus \
  -p 8081:8081 \
  -v nexus-data:/nexus-data \
  sonatype/nexus3:latest

# Lấy initial admin password
docker exec nexus cat /nexus-data/admin.password
```

2.2 Cấu hình Maven Repository

```
<!-- ~/.m2/settings.xml -->
<settings>
  <mirrors>
    <mirror>
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>http://nexus.internal:8081/repository/maven-group/</url>
    </mirror>
  </mirrors>
  <servers>
    <server>
      <id>nexus</id>
      <username>deployer</username>
      <password>${env.NEXUS_PASSWORD}</password>
    </server>
  </servers>
</settings>
```

2.3 Cấu hình npm Registry

```
# Trỏ npm về Nexus proxy
npm config set registry http://nexus.internal:8081/repository/npm-group/

# Login để publish
npm login --registry=http://nexus.internal:8081/repository/npm-hosted/

# Publish package
npm publish --registry=http://nexus.internal:8081/repository/npm-hosted/
```

2.4 Docker Registry qua Nexus

```
# Cấu hình Docker daemon để dùng insecure registry (internal only)
# /etc/docker/daemon.json
```

```
{
  "insecure-registries": ["nexus.internal:8082"]
}

# Push image
docker tag myapp:1.0 nexus.internal:8082/myapp:1.0
docker push nexus.internal:8082/myapp:1.0

# Pull qua proxy (cache Docker Hub)
docker pull nexus.internal:8083/library/nginx:alpine
```

---

3. Harbor: Enterprise Container Registry

Harbor là CNCF graduated project, cung cấp container registry với vulnerability scanning, RBAC, replication, và signing.

3.1 Cài đặt Harbor với Helm

```
helm repo add harbor https://helm.goharbor.io
helm repo update

helm install harbor harbor/harbor \
  --namespace harbor \
  --create-namespace \
  --set expose.type=ingress \
  --set expose.ingress.hosts.core=harbor.example.com \
  --set externalURL=https://harbor.example.com \
  --set harborAdminPassword=SecurePassword123 \
  --set persistence.persistentVolumeClaim.registry.size=100Gi
```

3.2 Vulnerability Scanning với Trivy

Harbor tích hợp Trivy để scan container images tự động khi push.

```
# Câú hình Harbor: Admin → Interrogation Services → Add Trivy scanner

# Scan image thủ công
trivy image harbor.example.com/myproject/myapp:1.0

# CI/CD: Fail build nêú có CRITICAL vulnerabilities
trivy image \
  --severity CRITICAL,HIGH \
  --exit-code 1 \
  harbor.example.com/myproject/myapp:1.0
```

3.3 Harbor Replication

```
# Replicate từ Harbor → ECR (cho DR hoặc multi-region)
# Harbor UI: Administration → Replications → New Replication Rule
name: "replicate-to-ecr"
source_registry: harbor.example.com
destination_registry: 123456789.dkr.ecr.ap-southeast-1.amazonaws.com
trigger: "On Push"  # Tự động replicate khi push
filter:
  name: "myproject/**"  # Chỉ replicate project này
```

---

4. Container Image Best Practices

4.1 Tagging Strategy

```
# BAD: chỉ dùng :latest
docker push myapp:latest  # Không traceble, không rollback được

# GOOD: semantic versioning + git SHA
VERSION="1.2.3"
GIT_SHA=$(git rev-parse --short HEAD)
BUILD_DATE=$(date -u +%Y%m%dT%H%M%SZ)

docker tag myapp:build \
  harbor.example.com/prod/myapp:${VERSION}
docker tag myapp:build \
  harbor.example.com/prod/myapp:${VERSION}-${GIT_SHA}
docker tag myapp:build \
  harbor.example.com/prod/myapp:latest

docker push harbor.example.com/prod/myapp:${VERSION}
docker push harbor.example.com/prod/myapp:${VERSION}-${GIT_SHA}
docker push harbor.example.com/prod/myapp:latest
```

4.2 Image Signing với Cosign (Sigstore)

```
# Cài cosign
brew install cosign  # hoặc download binary

# Generate key pair
cosign generate-key-pair

# Sign image sau khi push
cosign sign --key cosign.key harbor.example.com/prod/myapp:1.2.3
```

```
# Verify signature trước khi deploy
cosign verify \
  --key cosign.pub \
  harbor.example.com/prod/myapp:1.2.3

# Keyless signing với OIDC (GitHub Actions)
cosign sign \
  --identity-token=$(cat $ACTIONS_ID_TOKEN_REQUEST_TOKEN) \
  harbor.example.com/prod/myapp:1.2.3
```

---

5. SBOM (Software Bill of Materials)

SBOM là danh sách đầy đủ tất cả components, dependencies, và metadata của một software artifact.

5.1 Generate SBOM với Syft

```
# Cài syft
curl -sSfL https://raw.githubusercontent.com/anchore/syft/main/install.sh | sh

# Generate SBOM cho container image (SPDX format)
syft harbor.example.com/prod/myapp:1.2.3 -o spdx-json > sbom.spdx.json

# Generate SBOM cho directory (CycloneDX format)
syft dir:./src -o cyclonedx-json > sbom.cyclonedx.json

# Attach SBOM vào OCI image
cosign attach sbom \
  --sbom sbom.spdx.json \
  --type spdx \
  harbor.example.com/prod/myapp:1.2.3

# Verify SBOM
cosign download sbom harbor.example.com/prod/myapp:1.2.3
```

5.2 SLSA Framework (Supply-chain Levels for Software Artifacts)

Level	Yêu cầu	Bảo vệ khỏi
SLSA 1	Build process documented	Không có documentation
SLSA 2	Version control + generated provenance	Manual errors
SLSA 3	Hardened build environment	Build compromise
SLSA 4	Two-party review + hermetic builds	Insider threats

```yaml
# GitHub Actions: Generate SLSA provenance
- name: Generate SLSA provenance
  uses: slsa-framework/slsa-github-generator/.github/workflows/generator_container_slsa3.yml@v1
  with:
    image: harbor.example.com/prod/myapp
    digest: ${{ steps.build.outputs.digest }}
```

---

6. Artifact Promotion Pipeline

```
Developer push code
        |
        ▼
  CI Build & Test
        |
        ▼
  Push to DEV registry
  harbor.example.com/dev/myapp:feature-xyz-abc1234
        |
   (Integration tests pass)
        |
        ▼
  Promote to STAGING registry
  harbor.example.com/staging/myapp:1.2.3-rc1
  (Copy image, không rebuild!)
        |
   (UAT/QA approval)
        |
        ▼
  Promote to PROD registry
  harbor.example.com/prod/myapp:1.2.3
  (Sign với cosign, attach SBOM)
```

```bash
# Promote image: copy từ staging → prod (không rebuild)
# Dùng crane (Google container tool)
crane copy \
  harbor.example.com/staging/myapp:1.2.3-rc1 \
  harbor.example.com/prod/myapp:1.2.3

# Hoặc skopeo
skopeo copy \
  docker://harbor.example.com/staging/myapp:1.2.3-rc1 \
  docker://harbor.example.com/prod/myapp:1.2.3
```

---

7. Cleanup Policies & Retention Rules

7.1 Harbor Cleanup Policy

```
# Harbor UI: Projects → myproject → Policy → Tag Retention
# Giữ lại:
# - Tất cả tags matching "v[0-9]+.[0-9]+.[0-9]+" (semantic versions)
# - 10 tags mới nhất cho branch develop
# - Xóa tags cũ hơn 30 ngày không có pattern semver

# Garbage Collection (xóa unreferenced layers)
# Harbor UI: Administration → Garbage Collection → Schedule
# Chạy hàng tuần vào chủ nhật 2:00 AM
```

7.2 Nexus Cleanup Policies

```
// Nexus REST API: tạo cleanup policy
POST /service/rest/v1/cleanup-policies
{
  "name": "cleanup-snapshots",
  "format": "maven2",
  "notes": "Remove SNAPSHOT artifacts older than 7 days",
  "criteria": {
    "lastBlobUpdated": 7,      // Ngày
    "lastDownloaded": 30,      // Ngày không được download
    "preRelease": "PRERELEASES" // Chỉ áp dụng cho pre-release
  }
}
```

---

8. Ví dụ: Setup Harbor với Vulnerability Scanning trong CI/CD

```
# .github/workflows/build-and-push.yml
name: Build and Push

on:
  push:
    branches: [main]
    tags: ["v*"]

env:
  REGISTRY: harbor.example.com
  IMAGE_NAME: myproject/myapp

jobs:
  build:
```

```yaml
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Login to Harbor
        uses: docker/login-action@v3
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ secrets.HARBOR_USERNAME }}
          password: ${{ secrets.HARBOR_PASSWORD }}

      - name: Build and Push
        id: build
        uses: docker/build-push-action@v5
        with:
          push: true
          tags: |
            ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }}
            ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:latest

      - name: Scan với Trivy
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: "${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }}"
          format: "sarif"
          output: "trivy-results.sarif"
          severity: "CRITICAL,HIGH"
          exit-code: "1"  # Fail nếu có lỗ hổng nghiêm trọng

      - name: Sign image với Cosign
        env:
          COSIGN_PRIVATE_KEY: ${{ secrets.COSIGN_PRIVATE_KEY }}
        run: |
          cosign sign --key env://COSIGN_PRIVATE_KEY \
            ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }}

      - name: Generate và attach SBOM
        run: |
          syft ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }} \
            -o spdx-json > sbom.json
          cosign attach sbom \
            --sbom sbom.json \
            ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.sha }}
```

Chương 23: Service Mesh

Service mesh là infrastructure layer xử lý service-to-service communication trong microservices, cung cấp traffic management, security (mTLS), và observability mà không cần thay đổi application code.

---

1. Khái niệm cơ bản

1.1 Vấn đề Service Mesh giải quyết

```
Không có service mesh:
ServiceA → HTTP → ServiceB
(Phải tự handle: retry, timeout, circuit breaker, mTLS, tracing)

Với service mesh:
ServiceA → [Sidecar Proxy] → [Sidecar Proxy] → ServiceB
            (Envoy)              (Envoy)
(Infrastructure layer tự động handle tất cả)
```

1.2 Data Plane vs Control Plane

```
Control Plane (Istiod):
├── Pilot: Service discovery, traffic routing rules
├── Citadel: Certificate management (mTLS)
└── Galley: Configuration validation

Data Plane (Envoy sidecars):
├── Intercept tất cả inbound/outbound traffic của pod
├── Apply routing rules từ Control Plane
├── Report metrics/traces lên observability tools
└── Enforce mTLS encryption
```

---

2. Istio

2.1 Cài đặt Istio

```bash
# Download istioctl
curl -L https://istio.io/downloadIstio | sh -
export PATH="$HOME/istio-1.20.0/bin:$PATH"

# Cài Istio với default profile
istioctl install --set profile=default -y

# Verify installation
istioctl verify-install
```

```
# Enable sidecar injection cho namespace
kubectl label namespace production istio-injection=enabled

# Verify sidecar được inject (pod có 2 containers)
kubectl get pods -n production
# NAME                      READY   STATUS    RESTARTS
# webapp-7d6b8f9d4-xk2p9   2/2     Running   0  ← 2/2 = app + envoy sidecar
```

2.2 VirtualService - Traffic Routing

```
# Routing dựa trên header (A/B testing)
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: webapp
spec:
  hosts:
  - webapp
  http:
  # Route users với header "x-version: v2" đến v2
  - match:
    - headers:
        x-version:
          exact: "v2"
    route:
    - destination:
        host: webapp
        subset: v2
  # Route còn lại đến v1
  - route:
    - destination:
        host: webapp
        subset: v1
      weight: 90
    - destination:
        host: webapp
        subset: v2
      weight: 10
```

2.3 DestinationRule - Load Balancing & Circuit Breaker

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: webapp
```

```yaml
spec:
  host: webapp
  trafficPolicy:
    connectionPool:
      http:
        http1MaxPendingRequests: 100
        http2MaxRequests: 1000
    outlierDetection:
      # Circuit breaker: eject host sau 5 lỗi trong 1 phút
      consecutiveGatewayErrors: 5
      interval: 1m
      baseEjectionTime: 30s
      maxEjectionPercent: 50
    loadBalancer:
      simple: LEAST_CONN  # ROUND_ROBIN, RANDOM, PASSTHROUGH
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

2.4 Retry & Timeout

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: payment-service
spec:
  hosts:
  - payment-service
  http:
  - timeout: 5s          # Global timeout 5 giây
    retries:
      attempts: 3        # Retry 3 lần
      perTryTimeout: 2s  # Mỗi lần retry timeout 2 giây
      retryOn: "gateway-error,connect-failure,retriable-4xx"
    route:
    - destination:
        host: payment-service
```

2.5 Fault Injection (Chaos Testing)

```yaml
# Inject 50% delay 5 giây vào calls đến recommendation-service
apiVersion: networking.istio.io/v1alpha3
```

```yaml
kind: VirtualService
metadata:
  name: recommendation-service
spec:
  hosts:
  - recommendation-service
  http:
  - fault:
      delay:
        percentage:
          value: 50.0
        fixedDelay: 5s
      abort:
        percentage:
          value: 10.0
        httpStatus: 503  # 10% requests trả về`503
    route:
    - destination:
        host: recommendation-service
```

---

3. Istio Security: mTLS & Authorization

3.1 PeerAuthentication - Enable mTLS

```yaml
# Enable STRICT mTLS cho toàn namespace
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: production
spec:
  mtls:
    mode: STRICT  # PERMISSIVE (mixed), STRICT (mTLS only), DISABLE

# Kiểm tra mTLS đang hoạt động
istioctl x describe pod webapp-7d6b8f9d4-xk2p9 -n production
```

3.2 AuthorizationPolicy - Zero Trust

```yaml
# Chỉ cho phép frontend gọi backend, không cho phép others
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: backend-policy
```

```
  namespace: production
spec:
  selector:
    matchLabels:
      app: backend
  rules:
  - from:
    - source:
        principals:
        - "cluster.local/ns/production/sa/frontend"  # Service Account
    to:
    - operation:
        methods: ["GET", "POST"]
        paths: ["/api/*"]
  # Deny everything else (default deny)
```

---

4. Istio Observability

4.1 Kiali - Service Graph

```
# Cài Kiali cùng với addons
kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.20/samples/addons/kiali.yaml
kubectl apply -f
↪  https://raw.githubusercontent.com/istio/istio/release-1.20/samples/addons/prometheus.yaml
kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.20/samples/addons/jaeger.yaml
kubectl apply -f https://raw.githubusercontent.com/istio/istio/release-1.20/samples/addons/grafana.yaml

# Mở Kiali dashboard
istioctl dashboard kiali
# Xem: service graph, traffic flow, error rates, latency heatmap
```

4.2 Distributed Tracing với Jaeger

```
# Istio tự động inject trace headers (x-b3-traceid, etc.)
# Chỉ cần forward headers trong application code

# Python Flask example
from flask import Flask, request, Response
import requests

app = Flask(__name__)

TRACE_HEADERS = [
    "x-request-id", "x-b3-traceid", "x-b3-spanid",
```

```
    "x-b3-parentspanid", "x-b3-sampled", "b3"
]

@app.route("/order")
def create_order():
    headers = {h: request.headers.get(h) for h in TRACE_HEADERS if request.headers.get(h)}
    # Forward headers khi gọi downstream services
    payment_response = requests.post("http://payment-service/charge", headers=headers)
    return {"status": "ok"}
```

---

5. Linkerd: Lightweight Alternative

5.1 So sánh với Istio

Feature	Istio	Linkerd
Proxy	Envoy (C++)	Linkerd2-proxy (Rust)
Memory overhead	~200MB/sidecar	~10MB/sidecar
CPU overhead	Cao hơn	Thấp hơn
Configuration	YAML phức tạp	Đơn giản hơn
Feature set	Đầy đủ nhất	Core features
Learning curve	Dốc	Thoải hơn
Best for	Large, complex	Simplicity-first

5.2 Cài đặt Linkerd

```
# Cài linkerd CLI
curl --proto "=https" --tlsv1.2 -sSfL https://run.linkerd.io/install | sh

# Pre-flight check
linkerd check --pre

# Cài Linkerd control plane
linkerd install --crds | kubectl apply -f -
linkerd install | kubectl apply -f -

# Verify
linkerd check

# Inject sidecar vào deployment
kubectl get deployment webapp -o yaml | linkerd inject - | kubectl apply -f -

# Xem metrics ngay lập tức
linkerd viz install | kubectl apply -f -
linkerd viz dashboard
```

---

6. eBPF-based Service Mesh: Cilium

Cilium Service Mesh dùng eBPF thay vì sidecar proxy, giảm overhead đáng kể.

```
Sidecar model (Istio/Linkerd):
[App Container] + [Envoy Sidecar] per pod
→ Overhead: memory, CPU, latency per pod

eBPF model (Cilium):
[App Container] per pod
[Cilium eBPF programs] in kernel (shared, không per-pod)
→ Overhead: thấp hơn nhiều, không có extra container
```

```bash
# Cài Cilium với service mesh features
helm install cilium cilium/cilium \
  --namespace kube-system \
  --set kubeProxyReplacement=true \
  --set ingressController.enabled=true \
  --set envoy.enabled=true  # Dùng Envoy ở node level thay vì per-pod

# Kiểm tra
cilium status
cilium connectivity test
```

---

7. So sánh: Istio vs Linkerd vs Consul Connect vs Cilium

	Istio	Linkerd	Consul Connect	Cilium
Proxy	Envoy	Rust proxy	Envoy	eBPF + Envoy
Multi-cluster	Có	Có (basic)	Có (tốt nhất)	Có
VM support	Có	Không	Có	Partial
Non-K8s	Có	Không	Có	Không
Overhead	Cao	Thấp	Trung bình	Rất thấp
mTLS	Auto	Auto	Manual config	Auto

---

8. Khi nào cần và không cần Service Mesh

NÊN dùng khi: - Có 10+ microservices cần secure communication - Cần mTLS zero-trust networking - Cần traffic management tinh vi (canary, A/B test) - Cần distributed tracing cross-service - Compliance yêu cầu encryption in-transit

KHÔNG NÊN dùng khi: - Monolith hoặc ít hơn 5 services - Team chưa quen Kubernetes - Resource constrained (< 4 CPU, < 8GB RAM) - Latency cực kỳ sensitive (HFT, gaming) - Startup/prototype stage

---

9. Ví dụ: Istio Setup cho Microservices E-Commerce

```
# 1. Enable injection cho namespace
kubectl label namespace ecommerce istio-injection=enabled

# 2. Deploy services (sidecar tự động inject)
kubectl apply -f manifests/ -n ecommerce

# 3. Apply global mTLS policy
kubectl apply -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: ecommerce
spec:
  mtls:
    mode: STRICT
EOF

# 4. Rate limiting cho payment service (dùng EnvoyFilter)
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: payment-ratelimit
  namespace: ecommerce
spec:
  workloadSelector:
    labels:
      app: payment-service
  configPatches:
  - applyTo: HTTP_FILTER
    match:
      context: SIDECAR_INBOUND
    patch:
      operation: INSERT_BEFORE
      value:
        name: envoy.filters.http.local_ratelimit
        typed_config:
          "@type": type.googleapis.com/envoy.extensions.filters.http.local_ratelimit.v3.LocalRateLimit
          stat_prefix: http_local_rate_limiter
```

```
          token_bucket:
            max_tokens: 100
            tokens_per_fill: 100
            fill_interval: 60s
EOF

# 5. Verify traffic flow trong Kiali
istioctl dashboard kiali
# Xem: ecommerce namespace → Graph
# Thấy service dependencies, error rates, request rates
```

Chương 24: Site Reliability Engineering (SRE)

SRE là discipline kết hợp software engineering và systems operations, được Google phát minh để scale reliability một cách bền vững. SRE không phải là title, mà là một culture và practice.

---

1. SRE Principles

1.1 Embracing Risk

```
Không phải mục tiêu là 100% uptime.
100% uptime:
- Không thể đạt được (dependency không 100%)
- Quá đắt để duy trì
- Người dùng không phân biệt được 99.99% vs 100%

Mục tiêu thực tế:
- Đặt SLO phù hợp với business need
- Dùng error budget để cân bằng reliability vs velocity
- "Hope is not a strategy" → measure everything
```

1.2 Toil

Toil là công việc thủ công, lặp đi lặp lại, không mang lại giá trị lâu dài.

```
Đặc điểm của Toil:
✗ Manual (cần người làm)
✗ Repetitive (làm đi làm lại)
✗ Automatable (có thể tự động hóa)
✗ Reactive (chạy theo sự cố)
✗ Không cải thiện service theo thời gian

Rule: SRE không được dành hơn 50% thời gian cho toil
→ 50% còn lại cho engineering work (automation, tooling)
```

---

2. SLI, SLO, SLA

2.1 SLI (Service Level Indicator)

Metric đo lường behavior của service từ góc nhìn người dùng.

```
Availability SLI:
= (successful requests) / (total requests) × 100%
= (requests với HTTP 2xx/3xx) / (tất cả requests)

Latency SLI:
= % requests hoàn thành trong X ms
= P99 latency < 500ms (99% requests xong trong 500ms)

Error Rate SLI:
= (error requests) / (total requests) × 100%

Throughput SLI:
= requests per second (RPS) hệ thống xử lý được
```

2.2 SLO (Service Level Objective)

Target mà team cam kết đạt được, đo trong một rolling window.

```yaml
# Ví dụ SLO cho web service (định nghĩa trong YAML)
slo:
  name: "webapp-availability"
  description: "Homepage và API endpoints phải available"

  sli:
    type: availability
    query: |
      sum(rate(http_requests_total{status=~"2..",job="webapp"}[5m]))
      /
      sum(rate(http_requests_total{job="webapp"}[5m]))

  objectives:
    - target: 0.999       # 99.9% availability
      window: 30d          # Trong 30 ngày rolling window

  # Alerting: burn rate alerts
  alerts:
    - name: "high-burn-rate"
      severity: page
      burnRate: 14.4  # 14.4x burn rate = sẽ hết budget trong 5% window
      window: 1h
```

2.3 Error Budget

```
Error Budget = 1 - SLO target
Ví dụ: SLO = 99.9% → Error Budget = 0.1%

Trong 30 ngày (43,200 phút):
Error Budget = 43,200 × 0.001 = 43.2 phút downtime cho phép

Error Budget Policy:
- Budget > 50%: Team có thể deploy freely
- Budget 10-50%: Review thêm trước deployment
- Budget < 10%: Freeze feature releases, focus reliability
- Budget = 0%: Chỉ fix reliability issues, không release mới
```

```python
# Script tính error budget
def calculate_error_budget(slo_target: float, window_days: int,
                           current_availability: float) -> dict:
    window_minutes = window_days * 24 * 60
    budget_minutes = window_minutes * (1 - slo_target)
    consumed_minutes = window_minutes * (1 - current_availability)
    remaining_minutes = budget_minutes - consumed_minutes
    remaining_percent = (remaining_minutes / budget_minutes) * 100

    return {
        "budget_total_minutes": round(budget_minutes, 1),
        "consumed_minutes": round(consumed_minutes, 1),
        "remaining_minutes": round(remaining_minutes, 1),
        "remaining_percent": round(remaining_percent, 1),
        "status": "OK" if remaining_percent > 10 else "WARNING" if remaining_percent > 0 else "EXHAUSTED"
    }

# Ví dụ
result = calculate_error_budget(0.999, 30, 0.9994)
# {'budget_total_minutes': 43.2, 'consumed_minutes': 25.9, 'remaining_percent': 40.0, 'status': 'OK'}
```

2.4 SLA (Service Level Agreement)

Contract pháp lý với khách hàng, thường lỏng hơn SLO internal.

```
Relationship:
SLA (external commitment) < SLO (internal target) < SLI (actual metric)

Ví dụ:
SLA với khách hàng: 99.5% uptime/month
SLO internal:       99.9% uptime/month  (buffer để không vi phạm SLA)
SLI actual:         99.94% (đo được thực tế)

Hậu quả vi phạm SLA: Service credits, penalties, churn
Hậu quả vi phạm SLO: Error budget consumed, deploy freeze
```

3. Incident Management

3.1 Severity Levels

Severity	Mô tả	Response Time	Ví dụ
SEV-1	Service down, tất cả users bị ảnh hưởng	< 5 phút	Production site down
SEV-2	Degraded performance, nhiều users	< 15 phút	Checkout chậm 10x
SEV-3	Minor issue, ít users	< 1 giờ	Dashboard broken cho 5% users
SEV-4	Cosmetic, không ảnh hưởng functionality	Next sprint	CSS misaligned

3.2 Incident Response Roles

```
Incident Commander (IC):
- Coordinator chính, không fix trực tiếp
- Delegate tasks, track progress
- Communicate với stakeholders
- Declare incident resolved

Technical Lead:
- Lead investigation và fix
- Report to IC

Communications Lead:
- Update status page
- Notify customers
- Internal stakeholder updates

Scribe:
- Ghi lại timeline của incident
- Document actions taken
```

3.3 Incident Communication Template

```
## [SEV-1] Payment Service Degradation - 14:32 UTC
```

```
**Status:** Investigating

**Impact:** ~30% of payment transactions failing with 503 errors.
Estimated 2,000 users/hour affected.

**Timeline:**
- 14:28 - Alert fired: payment-service error rate > 10%
- 14:32 - Incident declared SEV-1, @alice assigned IC
- 14:35 - @bob investigating payment service logs
- 14:41 - Root cause identified: database connection pool exhausted
- 14:48 - Fix deployed: increased connection pool from 10 to 50
- 14:52 - Error rate back to normal, monitoring 10 minutes
- 15:02 - Incident resolved

**Next steps:** Post-mortem scheduled for 2026-03-05
```

---

4. Post-Mortem: Blameless Culture

4.1 Blameless Post-Mortem Template

```
# Post-Mortem: Payment Service Outage (2026-03-03)

## Summary
Payment service bị lỗi 20 phút, ảnh hưởng ~600 transactions.

## Impact
- Duration: 14:32 - 14:52 UTC (20 phút)
- Error rate: peak 34%
- Affected transactions: ~600
- Revenue impact: ước tính $15,000

## Root Cause
DB connection pool (max=10) bị exhausted do traffic spike sau marketing campaign.
Không có alert cho connection pool utilization.

## Contributing Factors
- Connection pool size không được review khi traffic tăng 3x (tháng trước)
- Không có load testing với new traffic levels
- Runbook không đề`cập connection pool troubleshooting

## Timeline
[Chi tiết timeline như trên]

## What Went Well
- Alert fired trong vòng 4 phút
```

```
- Root cause identify nhanh (9 phút)
- Fix đơn giản và không cần deploy

## Action Items
| Action | Owner | Due Date | Priority |
|--------|-------|----------|----------|
| Add connection pool monitoring alert | @alice | 2026-03-10 | P1 |
| Document connection pool config trong runbook | @bob | 2026-03-12 | P2 |
| Quarterly load testing với 2x expected traffic | @team | 2026-04-01 | P2 |
| Auto-scaling cho connection pool | @carol | 2026-03-20 | P1 |

## Lessons Learned
- Infrastructure config cần được review khi traffic patterns thay đổi
- "It worked last month" không đủ → automated regression testing
```

---

5. Chaos Engineering

5.1 Chaos Monkey & Netflix Chaos Principles

```
Chaos Engineering Principles:
1. Define "steady state" (normal behavior)
2. Hypothesize steady state sẽ tiếp tục trong control AND experimental group
3. Introduce variables: server crashes, network latency, disk full
4. Try to disprove hypothesis
```

5.2 Chaos Mesh (Kubernetes)

```yaml
# Inject network latency vào payment-service
apiVersion: chaos-mesh.org/v1alpha1
kind: NetworkChaos
metadata:
  name: payment-latency-test
spec:
  action: delay
  mode: one
  selector:
    namespaces:
    - production
    labelSelectors:
      app: payment-service
  delay:
    latency: "200ms"
    correlation: "25"
    jitter: "50ms"
  duration: "5m"
```

```yaml
---
# Kill random pods trong deployment
apiVersion: chaos-mesh.org/v1alpha1
kind: PodChaos
metadata:
  name: webapp-pod-kill
spec:
  action: pod-kill
  mode: random-max-percent
  value: "30"  # Kill 30% pods ngẫu nhiên
  selector:
    namespaces: [production]
    labelSelectors:
      app: webapp
  duration: "2m"
  scheduler:
    cron: "@every 1h"  # Chạy mỗi giờ trong business hours
```

5.3 Litmus Chaos Experiments

```bash
# Cài Litmus
kubectl apply -f https://litmuschaos.github.io/litmus/litmus-operator-v3.0.0.yaml

# Chạy experiment: pod-cpu-hog
kubectl apply -f - <<EOF
apiVersion: litmuschaos.io/v1alpha1
kind: ChaosEngine
metadata:
  name: webapp-cpu-hog
spec:
  appinfo:
    appns: production
    applabel: app=webapp
  chaosServiceAccount: litmus-admin
  experiments:
  - name: pod-cpu-hog
    spec:
      components:
        env:
        - name: TOTAL_CHAOS_DURATION
          value: "60"       # 60 giây
        - name: CPU_CORES
          value: "1"        # Hog 1 CPU core
        - name: PODS_AFFECTED_PERC
          value: "50"       # 50% pods
EOF
```

6. On-Call Best Practices

6.1 On-Call Rotation

```yaml
# PagerDuty schedule (ví dụ)
schedule:
  name: "Backend On-Call"
  rotation_type: weekly
  handoff_time: "Monday 09:00 UTC"

  teams:
    - name: "Team Alpha"
      members: [alice, bob, carol]
    - name: "Team Beta"
      members: [dave, eve, frank]

  escalation:
    level_1: on-call engineer (response: 5 min)
    level_2: team lead (if no response in 15 min)
    level_3: engineering manager (if no response in 30 min)
```

6.2 Runbook Example

```
# Runbook: High Database CPU Alert

## Alert: `DatabaseCPUHigh` (> 80% for 5 minutes)

## Immediate Actions (< 2 phút)
1. Check current queries: `SELECT * FROM pg_stat_activity WHERE state = 'active' ORDER BY duration DESC
↪  LIMIT 20;`
2. Check slow query log: `tail -f /var/log/postgresql/slow-queries.log`

## Investigation Steps
1. Identify top queries by CPU: `SELECT pid, query, cpu_time FROM pg_stat_statements ORDER BY cpu_time
↪  DESC LIMIT 5;`
2. Check for missing indexes: Xem EXPLAIN ANALYZE cho slow queries
3. Check connection count: `SELECT count(*) FROM pg_stat_activity;`

## Mitigation Options
- Kill long-running query: `SELECT pg_terminate_backend(PID);`
- Add missing index (nếu safe): `CREATE INDEX CONCURRENTLY idx_name ON table(column);`
- Scale up RDS instance (nếu cần): AWS Console → RDS → Modify

## Escalate if
- CPU > 95% sau 10 phút
```

```
- Production down
- Không tìm được root cause sau 30 phút

## Contact
- DBA team: #db-oncall Slack
- Manager: alice@company.com
```

---

7. Capacity Planning

```python
# Demand forecasting script đơn giản
import numpy as np

def forecast_capacity(historical_rps: list, growth_rate_monthly: float,
                      months_ahead: int, safety_factor: float = 1.3) -> dict:
    """
    historical_rps: list of monthly peak RPS (last 6 months)
    growth_rate_monthly: expected growth (e.g., 0.1 = 10% per month)
    safety_factor: headroom multiplier (1.3 = 30% buffer)
    """
    baseline = np.mean(historical_rps[-3:])  # Average of last 3 months
    forecasted = baseline * ((1 + growth_rate_monthly) ** months_ahead)
    required_capacity = forecasted * safety_factor

    return {
        "current_peak_rps": round(baseline),
        "forecasted_rps": round(forecasted),
        "required_capacity_rps": round(required_capacity),
        "months_ahead": months_ahead,
        "recommendation": f"Provision for {round(required_capacity)} RPS by month {months_ahead}"
    }

# Ví dụ: 10% monthly growth, planning 3 months ahead
result = forecast_capacity(
    historical_rps=[1000, 1100, 1150, 1200, 1300, 1400],
    growth_rate_monthly=0.10,
    months_ahead=3
)
# {'current_peak_rps': 1300, 'forecasted_rps': 1731, 'required_capacity_rps': 2250}
```

---

8. Ví dụ: SLO Definition cho Web Service

```yaml
# slo-config.yaml - Dùng với Sloth hoặc OpenSLO
version: "openslo/v1"
kind: SLO
metadata:
  name: webapp-api
  displayName: "Web Application API SLOs"

spec:
  service: webapp
  description: "SLOs cho webapp REST API endpoints"
  timeWindow:
    duration: 30d
    isRolling: true

  budgetingMethod: Occurrences

  objectives:
    # Availability SLO
    - displayName: "API Availability"
      op: gte
      target: 0.999  # 99.9%
      indicator:
        ratio:
          errors:
            metric:
              prometheusMetric: |
                sum(rate(http_requests_total{job="webapp",status=~"5.."}[5m]))
          total:
            metric:
              prometheusMetric: |
                sum(rate(http_requests_total{job="webapp"}[5m]))

    # Latency SLO
    - displayName: "API P99 Latency < 500ms"
      op: gte
      target: 0.995  # 99.5% requests dưới 500ms
      indicator:
        ratio:
          good:
            metric:
              prometheusMetric: |
                sum(rate(http_request_duration_seconds_bucket{job="webapp",le="0.5"}[5m]))
          total:
            metric:
              prometheusMetric: |
                sum(rate(http_request_duration_seconds_count{job="webapp"}[5m]))

  alertPolicies:
```

```yaml
  - name: webapp-burn-rate
    description: "Alert khi error budget burn rate quá cao"
    conditions:
      # Page: 14.4x burn rate trong 1h (hết budget trong 2 ngày)
      - kind: burnrate
        op: gte
        threshold: 14.4
        lookbackWindow: 1h
        alertAfter: 2m
    alertWhen: onBurnRateBreached
    notification:
      slack: "#sre-alerts"
      pagerduty: "webapp-oncall"
```

Giải thích error budget: - 30 ngày × 24h × 60min = 43,200 phút - 99.9% SLO → 43.2 phút downtime cho phép/tháng - Burn rate 14.4x → 43.2 / 14.4 = 3 phút thực tế để hết budget nếu tiếp tục - Alert sớm để team có thời gian phản ứng trước khi budget cạn kiệt