# DevOps Complete Study Notes

## with Microsoft Azure

Linux | Git | Docker | Kubernetes | Jenkins | Terraform | Ansible
Azure DevOps | AKS | Monitoring | Security | SRE | Interview Prep

## How to Use These Notes

This is a 30-day structured DevOps study plan. Each day has a topic, core concepts, key commands/syntax, and a quick recap. Study 1.5 to 2 hours daily. On weekends, revise the week's topics and build small hands-on labs.

| Week | Focus Area |
|---|---|
| Week 1 (Days 1–7) | Linux, Git, Networking, Shell Scripting — the absolute foundation |
| Week 2 (Days 8–14) | Docker, CI/CD, Jenkins, GitHub Actions — build and deploy |
| Week 3 (Days 15–21) | Kubernetes, Helm, Azure AKS — container orchestration |
| Week 4 (Days 22–30) | Azure DevOps, Terraform, Ansible, Monitoring, Security, SRE |

📌 **Note:** Do not skip days. If you miss a day, pick up where you left off — do not try to cover 2 days at once.

💡 **Tip:** After every topic, open a terminal or Azure portal and try the commands yourself. Reading alone will not make you interview-ready.

# WEEK 1 — Foundation: Linux, Git, Networking, Shell

## DAY 1 Linux Operating System Basics (Study Time: 2 hrs)

### What is Linux and Why DevOps Uses It

Linux is an open-source operating system. Almost all servers, cloud VMs, Docker containers, and Kubernetes nodes run on Linux. As a DevOps engineer you will SSH into Linux servers daily, write shell scripts, manage processes, and troubleshoot issues.

Linux follows a philosophy: everything is a file. Even devices, sockets, and processes are represented as files. This makes scripting and automation very powerful.

### Linux File System Structure

| Path | Purpose |
|---|---|
| / | Root directory — top of the entire file system |
| /home | Home directories for all users (e.g., /home/ram) |
| /etc | All configuration files live here (e.g., /etc/nginx/nginx.conf) |
| /var | Variable data — logs (/var/log), databases, email queues |
| /tmp | Temporary files — cleared on reboot |
| /bin | Essential command binaries (ls, cp, mv, cat) |
| /usr/bin | User-installed program binaries |
| /proc | Virtual filesystem — info about running processes (e.g., /proc/cpuinfo) |
| /opt | Optional/third-party software installations |
| /dev | Device files (disk drives, USB, terminals) |

### Essential Commands You Must Know

#### ▶ Navigation

```
pwd                     # show current directory
ls -la                  # list all files with permissions and hidden
files
cd /var/log             # change to /var/log
cd ..                   # go one level up
cd ~                    # go to home directory
```

#### ▶ File Operations

```
touch file.txt          # create empty file
mkdir -p /app/config    # create directory and parents
```

```
cp file.txt /tmp/       # copy file
mv old.txt new.txt      # rename or move file
rm -rf /tmp/test/       # force delete directory recursively
cat file.txt            # print file contents
less file.txt           # scroll through large files
head -20 file.txt       # first 20 lines
tail -f /var/log/syslog  # live follow log file
```

### ▶ Search

```
grep -r 'error' /var/log/    # search all logs for 'error'
find / -name '*.conf' 2>/dev/null    # find all .conf files
grep -i 'failed' app.log     # case-insensitive search
```

## File Permissions (Very Important for DevOps)

Every file has 3 permission groups: Owner, Group, Others. Each can have Read (r=4), Write (w=2), Execute (x=1).

```
ls -la   # output: -rwxr-xr-- 1 ram devs 4096 Jan 1 app.sh
```

Breaking down -rwxr-xr--:  first dash = file type.  rwx = owner can read, write, execute.  r-x = group can read and execute.  r-- = others can only read.

```
chmod 755 script.sh     # rwxr-xr-x  (owner all, group+others read+exec)
chmod 644 config.cfg    # rw-r--r--  (owner read+write, others read only)
chown ram:devs file.sh  # change owner to 'ram' and group to 'devs'
```

📌 **Note:** In DevOps, wrong permissions are a common cause of deployment failures. Always check permissions when a script fails to execute.

## Process Management

```
ps aux                  # list all running processes
top                     # live process monitor (press q to quit)
htop                    # better interactive process monitor
kill -9 1234            # force kill process with PID 1234
pkill nginx             # kill process by name
systemctl start nginx   # start a service
systemctl stop nginx    # stop a service
systemctl status nginx  # check if service is running
systemctl enable nginx  # auto-start service on boot
journalctl -u nginx -f  # live logs for a systemd service
```

### Disk and Memory

```
df -h                   # disk usage per filesystem
du -sh /var/log/        # size of /var/log directory
free -h                 # memory usage (RAM + swap)
lsblk                   # list all disk devices
```

**Tip:** Memorize: ps aux | grep, tail -f, df -h, chmod, systemctl. These are used every single day in DevOps.

# DAY 2 Linux — Users, Networking, SSH & Cron Jobs

(Study Time: 2 hrs)

## User and Group Management

```
whoami                  # current logged-in user
id                      # show user ID and group IDs
sudo su -               # switch to root user
useradd -m devuser      # create new user with home dir
passwd devuser          # set password for user
usermod -aG sudo devuser  # add user to sudo group
groups devuser          # show groups for user
```

## SSH — Secure Shell (Used Every Day in DevOps)

SSH lets you remotely connect to Linux servers securely. In DevOps you use SSH to access cloud VMs, debug servers, deploy code, and run commands remotely.

### ▶ Key Concepts

- **Password SSH** — user authenticates with username and password
- **Key-Based SSH** — safer and standard in DevOps — uses public/private key pair. Server stores your public key. You keep private key.

```
ssh-keygen -t rsa -b 4096   # generate SSH key pair on your machine
# creates: ~/.ssh/id_rsa (private) and ~/.ssh/id_rsa.pub (public)
ssh-copy-id user@server-ip  # copy public key to server
ssh user@192.168.1.100      # connect to server
ssh -i mykey.pem ec2-user@server  # connect using specific key file
scp file.txt user@server:/tmp/    # copy file to remote server
scp user@server:/tmp/file.txt .   # copy file from remote server
```

📌 **Note:** Never share your private key (.pem or id_rsa). Public key can be freely shared. In Azure, you download a .pem file when creating a VM.

## Networking Commands

```
ifconfig / ip addr          # show network interfaces and IP addresses
ping google.com             # test if host is reachable
curl http://localhost:8080  # make HTTP request and see response
wget http://example.com/file.tar.gz  # download file
netstat -tuln               # show all listening ports
ss -tuln                    # modern alternative to netstat
nslookup google.com         # DNS lookup
traceroute google.com       # trace network path to destination
telnet server 3306          # test if port 3306 is open
```

## Cron Jobs — Scheduling Tasks

Cron jobs run commands automatically at scheduled times. DevOps uses cron for: database backups, log cleanup, health checks, report generation.

```
crontab -e    # edit cron jobs for current user
crontab -l    # list current user's cron jobs
```

Cron syntax:  MINUTE  HOUR  DAY  MONTH  WEEKDAY  COMMAND

```
# Cron examples:
0 2 * * *  /scripts/backup.sh     # every day at 2:00 AM
*/5 * * * * /scripts/monitor.sh   # every 5 minutes
0 0 * * 0  /scripts/cleanup.sh   # every Sunday at midnight
30 9 1 * * /scripts/report.sh    # 9:30 AM on 1st of every month
```

💡 **Tip:** Use crontab.guru website concept to visualize cron expressions — great for interviews.

## Environment Variables

```
echo $HOME                 # print HOME variable
export MY_VAR='hello'      # set variable for current session
echo $MY_VAR               # prints: hello
printenv                   # print all environment variables
# To persist variables, add to ~/.bashrc or ~/.profile:
echo 'export JAVA_HOME=/usr/lib/jvm/java-11' >> ~/.bashrc
source ~/.bashrc           # reload bashrc without logout
```

# DAY 3 Shell Scripting (Study Time: 2 hrs)

## What is Shell Scripting

A shell script is a plain text file containing a series of Linux commands. Instead of typing commands one by one, you put them in a script and run it once. In DevOps, shell scripts are used for: deployments, health checks, backups, log rotation, server setup, CI/CD automation.

```
#!/bin/bash                  # first line — tells OS to use bash shell
# This is a comment
echo 'Hello DevOps!'       # print to terminal
```

### Variables

```
NAME='DevOps'
echo "Hello $NAME"     # prints: Hello DevOps
VERSION=$(cat version.txt)     # store command output in variable
echo "Version: $VERSION"
```

### If / Else Conditions

```
#!/bin/bash
DISK=$(df / | tail -1 | awk '{print $5}' | tr -d '%')
```

```
if [ $DISK -gt 80 ]; then
  echo 'ALERT: Disk usage is above 80%'
elif [ $DISK -gt 60 ]; then
  echo 'WARNING: Disk usage above 60%'
else
  echo 'Disk usage is normal: '$DISK'%'
fi
```

## Loops

```
# For loop
for SERVER in web1 web2 web3; do
  echo "Checking server: $SERVER"
  ssh user@$SERVER 'uptime'
done

# While loop
COUNT=0
while [ $COUNT -lt 5 ]; do
  echo "Attempt $COUNT"
  COUNT=$((COUNT + 1))
done
```

## Functions

```
#!/bin/bash
check_service() {
  SERVICE=$1
  if systemctl is-active --quiet $SERVICE; then
    echo "$SERVICE is running"
  else
    echo "$SERVICE is DOWN — restarting..."
    systemctl restart $SERVICE
  fi
}
check_service nginx
check_service docker
```

## Useful Text Processing Commands

```
cat /etc/passwd | grep 'ram'     # search for user 'ram'
cat access.log | awk '{print $1}'  # print first column
cat data.csv | cut -d',' -f2     # print second comma-separated field
echo 'Hello World' | sed 's/World/DevOps/'  # replace text
sort file.txt | uniq -c | sort -rn  # count unique lines, sort by
frequency
wc -l file.txt                   # count number of lines
```

## Exit Codes

Every command returns an exit code. 0 means success. Anything else means failure. DevOps scripts must check exit codes to handle errors.

```
ls /tmp
echo $?    # prints 0 if success
ls /nonexistent
echo $?    # prints 2 (error)
# Best practice — exit script on any error:
set -e     # put this at top of every script
```

## A Real DevOps Script — Deployment Health Check

```
#!/bin/bash
set -e
APP_URL='http://localhost:8080/health'
MAX_RETRY=5
COUNT=0
echo 'Starting health check after deployment...'
while [ $COUNT -lt $MAX_RETRY ]; do
  HTTP_CODE=$(curl -s -o /dev/null -w '%{http_code}' $APP_URL)
  if [ $HTTP_CODE -eq 200 ]; then
```

```
    echo 'App is healthy! Deployment succeeded.'
    exit 0
  fi
  echo "Attempt $((COUNT+1)): App not ready (HTTP $HTTP_CODE).
Waiting..."
  sleep 10
  COUNT=$((COUNT + 1))
done
echo 'DEPLOYMENT FAILED — app did not become healthy'
exit 1
```

## DAY 4 Git & Version Control (Study Time: 2 hrs)

### What is Git and Why It Matters

Git is a distributed version control system. It tracks every change made to code. In DevOps, Git is the starting point of everything — code pushes trigger CI/CD pipelines, infrastructure changes are version-controlled in Git (GitOps), and all team collaboration happens through Git.

### Core Git Concepts

| Term | Meaning |
|---|---|
| Repository (repo) | A folder tracked by Git — contains all code and its full change history |
| Commit | A snapshot of your changes at a specific point in time with a message |
| Branch | An independent line of development — work without affecting main code |
| Merge | Combine changes from one branch into another |
| Pull Request (PR) | A request to merge your branch into the main branch — triggers code review |
| Remote | A copy of the repo hosted on GitHub/GitLab/Bitbucket/Azure Repos |
| Clone | Download a remote repo to your local machine |
| Push | Upload your local commits to the remote repo |
| Pull | Download latest changes from remote to your local machine |
| Staging Area | A middle step — you add files here before committing them |

### Daily Git Workflow

```
git init                          # initialize new repo in current
folder
```

```
git clone https://github.com/org/repo.git  # copy remote repo locally
git status                                  # see what files are changed or staged
git add file.txt                            # stage specific file
git add .                                   # stage all changed files
git commit -m 'Added nginx config'  # save staged changes with message
git push origin main                        # upload commits to remote main branch
git pull origin main                        # get latest changes from remote
```

## Branching Strategy

```
git branch                                  # list all local branches
git checkout -b feature/add-monitoring   # create and switch to new branch
git checkout main                           # switch back to main
git merge feature/add-monitoring  # merge feature branch into current
git branch -d feature/add-monitoring  # delete local branch after merge
```

## Git Log and History

```
git log --oneline                           # compact history view
git log --oneline --graph                   # visual branch graph
git diff                                    # see unstaged changes
git diff HEAD~1                             # compare with last commit
git show abc1234                            # details of a specific commit
```

## Undoing Mistakes

```
git restore file.txt                        # discard unstaged changes to a file
git restore --staged file.txt  # unstage a file
git revert abc1234                          # create new commit that undoes a commit (safe)
git reset --hard HEAD~1                     # undo last commit and discard changes (dangerous!)
```

📌 **Note:** Never use git reset --hard on shared branches. It rewrites history and breaks teammates' repos.

## Common Git Workflows in DevOps Teams

### ▶ GitFlow

main branch is production. develop branch is the integration branch. Developers create feature/fix branches from develop, then raise PR. After review, merge to develop. When ready for release, merge develop to main.

### ▶ Trunk-Based Development

Developers commit to main (trunk) daily. Feature flags hide incomplete features. Short-lived branches (max 1-2 days). This is what fast DevOps teams use — it keeps the pipeline always green.

## Git with Azure Repos

Azure Repos is Microsoft's hosted Git service inside Azure DevOps. It works exactly like GitHub.

```
git remote add origin https://dev.azure.com/org/project/_git/repo
git push -u origin main
```

**Tip:** In Azure DevOps, branch policies can require PR reviews and successful pipeline runs before merging — enforce this to maintain code quality.

# DAY 5 Networking Fundamentals for DevOps (Study Time: 1.5 hrs)

## Why DevOps Engineers Need Networking Knowledge

You will configure load balancers, open firewall ports, set up VPNs, troubleshoot service connectivity, and design VPC/VNet networks in cloud. Without networking basics, you cannot debug why a container cannot talk to a database or why an HTTP request is failing.

## OSI Model — Quick Reference

| Layer | What it Does |
|---|---|
| 7 — Application | HTTP, HTTPS, FTP, DNS, SMTP — what users interact with |
| 6 — Presentation | Encryption, data format (SSL/TLS lives here conceptually) |
| 5 — Session | Session establishment and management |
| 4 — Transport | TCP (reliable) and UDP (fast, no guarantee) — ports live here |
| 3 — Network | IP addressing and routing — packets travel here |
| 2 — Data Link | MAC addresses — within local network |
| 1 — Physical | Actual cables, WiFi signals |

📌 **Note:** For DevOps interviews, you mainly need layers 3 (Network), 4 (Transport), and 7 (Application).

## TCP vs UDP

| Feature | TCP | UDP |
|---|---|---|
| Connection | Connection-oriented (3-way handshake) | Connectionless |
| Reliability | Guaranteed delivery, retransmits lost packets | No guarantee — packets may be lost |
| Speed | Slower (overhead of acknowledgement) | Faster (no acknowledgement) |
| Use Cases | HTTP, HTTPS, SSH, databases | Video streaming, DNS, gaming, VoIP |

## IP Addressing

An IP address identifies a device on a network. IPv4 uses 32-bit addresses like 192.168.1.100. IPv6 uses 128-bit addresses like 2001:0db8:85a3::8a2e:0370:7334.

- **Public IP** — accessible from internet — assigned by cloud/ISP
- **Private IP** — only inside a network — 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
- **CIDR notation** — 192.168.1.0/24 means 256 addresses. /16 means 65536 addresses
- **Subnet** — divides a network into smaller parts — used heavily in Azure VNet/Subnet design

## DNS — Domain Name System

DNS converts human-readable domain names (google.com) to IP addresses. It is the phone book of the internet.

```
nslookup google.com      # query DNS for google.com
dig google.com           # detailed DNS lookup
```

| Record Type | Purpose |
|---|---|
| A Record | Maps domain to IPv4 address (www → 1.2.3.4) |
| CNAME | Maps domain to another domain (www → myapp.azurewebsites.net) |
| MX Record | Mail server for domain |
| TXT Record | Text info — used for SSL verification, SPF email auth |
| NS Record | Name servers authoritative for domain |

## HTTP / HTTPS

HTTP is the protocol for web communication. HTTPS is HTTP encrypted with TLS. Every web app and API uses these.

| Status Code | Meaning |
|---|---|
| 200 OK | Success |
| 201 Created | Resource created (POST success) |
| 301/302 | Redirect — permanent / temporary |
| 400 Bad Request | Client sent invalid data |
| 401 Unauthorized | Authentication required |
| 403 Forbidden | Authenticated but not allowed |

| | |
|---|---|
| 404 Not Found | Resource does not exist |
| 500 Internal Server Error | Server-side error |
| 502 Bad Gateway | Upstream server (app) is down — load balancer cannot reach it |
| 503 Service Unavailable | Server overloaded or in maintenance |

### Ports to Memorize

| Port | Service |
|---|---|
| 22 | SSH |
| 80 | HTTP |
| 443 | HTTPS |
| 3306 | MySQL |
| 5432 | PostgreSQL |
| 6379 | Redis |
| 27017 | MongoDB |
| 8080 | Common app/Tomcat/Jenkins |
| 2379/2380 | etcd (Kubernetes) |
| 6443 | Kubernetes API Server |

# DAY 6 CI/CD — Concepts, Jenkins & GitHub Actions

(Study Time: 2 hrs)

## What is CI/CD

CI stands for Continuous Integration. Every time a developer pushes code, it is automatically built and tested. This catches bugs early before they reach production. CD stands for Continuous Delivery or Continuous Deployment. After successful CI, the code is automatically deployed to staging or production environments.

Without CI/CD: developers work in isolation, code is manually tested before release, releases happen once a month and are very risky. With CI/CD: code is integrated dozens of times daily, automated tests catch bugs instantly, releases are small, frequent, and low-risk.

### CI/CD Pipeline Stages in Detail

| Stage | What Happens |
|---|---|
| Source / Trigger | Developer pushes code → webhook triggers the pipeline |
| Build | Compile code, download dependencies, create artifact (JAR, WAR, Docker image) |
| Unit Test | Run fast tests that test individual functions — must all pass |
| Code Quality | SonarQube scans for code smells, bugs, vulnerabilities, test coverage |
| Security Scan | SAST tools scan source code for security vulnerabilities |
| Package | Bundle artifact — create Docker image, push to registry (ACR, ECR, DockerHub) |
| Deploy to Staging | Deploy to staging environment — identical to production |
| Integration Test | Test how services talk to each other in staging |
| Deploy to Production | Deploy via Blue-Green, Canary, or Rolling strategy |
| Post-Deploy Monitor | Check health endpoint, watch error rate, alert if issues |

## Jenkins

Jenkins is the most widely used open-source CI/CD tool. It runs pipelines, called Jobs, which are defined in a Jenkinsfile stored in the repository.

### Jenkins Architecture

- **Master Node** — controls the Jenkins UI, schedules jobs, manages agents
- **Agent/Slave Node** — runs the actual build steps — can be Docker containers or VMs
- **Pipeline** — the complete build-deploy workflow defined as code in Jenkinsfile
- **Stage** — a logical section of the pipeline (Build, Test, Deploy)
- **Step** — a single command or action inside a stage

### Jenkinsfile — Declarative Pipeline

```
pipeline {
  agent any                      // run on any available agent
  environment {
    IMAGE_NAME = 'myapp'
    REGISTRY   = 'myacr.azurecr.io'
  }
  stages {
    stage('Checkout') {
      steps { checkout scm }     // pull code from repo
    }
    stage('Build') {
      steps { sh 'mvn clean package -DskipTests' }
    }
    stage('Unit Test') {
      steps { sh 'mvn test' }
      post { always { junit 'target/surefire-reports/*.xml' } }
    }
    stage('Docker Build & Push') {
      steps {
        sh 'docker build -t $REGISTRY/$IMAGE_NAME:$BUILD_NUMBER .'
        sh 'docker push $REGISTRY/$IMAGE_NAME:$BUILD_NUMBER'
      }
    }
    stage('Deploy to Staging') {
      steps { sh 'kubectl set image deploy/myapp myapp=$REGISTRY/
$IMAGE_NAME:$BUILD_NUMBER' }
    }
  }
  post {
    success { echo 'Pipeline succeeded!' }
    failure { echo 'Pipeline FAILED — check logs' }
  }
}
```

## GitHub Actions

GitHub Actions is a CI/CD platform built into GitHub. You define workflows in YAML files stored in .github/workflows/ in your repository. Each push or pull request can trigger a workflow.

```
# .github/workflows/ci.yml
name: CI Pipeline
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Java
        uses: actions/setup-java@v3
        with: { java-version: '17' }
      - name: Build
        run: mvn clean package
      - name: Test
        run: mvn test
      - name: Docker Build
        run: docker build -t myapp:${{ github.sha }} .
```

## DAY 7 Week 1 Revision + Hands-On Practice (Study Time: 2 hrs)

### Revision Topics

- **Linux** — File system, permissions (chmod 755), SSH key-based auth, process management, systemctl
- **Shell Scripting** — Variables, if/else, for loop, functions, exit codes, set -e
- **Git** — clone, add, commit, push, pull, branch, merge, PR workflow
- **Networking** — TCP vs UDP, HTTP status codes, DNS record types, common ports
- **CI/CD** — Stages: build, test, scan, package, deploy. Jenkinsfile stages. GitHub Actions YAML

### Hands-On Lab Tasks (Do All of These)

On your own computer or a free Azure VM (B1s free tier):

- Create a bash script that accepts a service name as argument and checks if it is running
- Create a Git repo, make 3 commits, create a branch, and merge it back via pull request
- Set up a cron job that writes the current date and disk usage to a log file every hour
- Write a Jenkinsfile or GitHub Actions YAML for a simple hello-world app pipeline

- SSH into any Linux VM and configure a user with a specific permission setup

# WEEK 2 — Docker, Containers & CI/CD Deep Dive

## DAY 8 Docker — Fundamentals (Study Time: 2 hrs)

### What is Docker

Docker is a containerization platform. It allows you to package an application with all its dependencies (code, runtime, libraries, config) into a single unit called a container. This container runs the same way on any machine — your laptop, staging server, or production cloud VM.

Before Docker, the classic problem was 'it works on my machine but not on the server.' Docker eliminates this because the container includes everything the app needs.

### Key Concepts

| Concept | Meaning |
|---|---|
| Image | Read-only template to create containers. Like a class in OOP. Built from Dockerfile. |
| Container | A running instance of an image. Like an object created from a class. |
| Dockerfile | A text script of instructions to build a Docker image. |
| Registry | A storage server for Docker images. DockerHub is public. Azure Container Registry (ACR) is private. |
| Volume | Persistent storage for containers. Data in a container is lost when it stops — volumes survive. |
| Network | Containers can communicate with each other via Docker networks. |
| Docker Compose | A YAML file to define and run multi-container apps together. |
| Layer | Each instruction in Dockerfile creates a read-only layer. Layers are cached and reused. |

### Docker vs Virtual Machine

| Feature | Docker Container | Virtual Machine |
|---|---|---|
| What it virtualizes | Application + OS libraries | Full operating system + hardware |
| Size | Megabytes (lightweight) | Gigabytes (heavy) |
| Startup time | Seconds | Minutes |
| Resource usage | Low — shares host OS kernel | High — each VM has its own OS |
| Isolation | Process-level isolation | Full hardware-level isolation |
| Use case | Microservices, CI/CD, scalable apps | Legacy apps, strong security isolation |

### Essential Docker Commands

#### ▶ Working with Images

```
docker images                    # list all downloaded images
docker pull nginx:latest        # download image from DockerHub
docker build -t myapp:v1 .      # build image from Dockerfile in
current dir
docker tag myapp:v1 myacr.azurecr.io/myapp:v1  # tag for registry
docker push myacr.azurecr.io/myapp:v1          # push to registry
docker rmi myapp:v1             # remove image
```

▶ **Working with Containers**

```
docker run nginx                # run container (foreground)
docker run -d nginx             # run detached (background)
docker run -d -p 8080:80 nginx  # map host port 8080 to container port
80
docker run -d -p 8080:80 --name webserver nginx
docker ps                       # list running containers
docker ps -a                    # list all containers including
stopped
docker stop webserver           # stop container
docker start webserver          # start stopped container
docker rm webserver             # remove container
docker logs webserver           # view container logs
docker logs -f webserver        # follow live logs
docker exec -it webserver bash  # get shell inside running container
docker inspect webserver        # detailed container info (IP,
volumes, etc)
```

## Dockerfile

A Dockerfile defines how to build a Docker image step by step.

```
# Dockerfile for a Java Spring Boot app
FROM eclipse-temurin:17-jre-alpine   # base image (small Alpine Linux
with Java 17)
WORKDIR /app                          # set working directory inside
container
COPY target/myapp.jar app.jar        # copy JAR from build output
EXPOSE 8080                           # document that app listens on
8080
```

```
ENV SPRING_PROFILES_ACTIVE=prod      # environment variable
RUN adduser --disabled-password appuser && chown appuser /app
USER appuser                          # run as non-root user (security
best practice)
ENTRYPOINT ["java", "-jar", "app.jar"]
```

```
# Dockerfile for Node.js app
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./                # copy package files first (layer
cache optimization)
RUN npm ci --only=production         # install dependencies
COPY . .                             # copy rest of source code
EXPOSE 3000
CMD ["node", "server.js"]
```

📌 **Note:** Copy package.json before copying source code. Dependencies change less often than code, so they get cached as a separate layer, making subsequent builds much faster.

# DAY 9 Docker — Advanced: Volumes, Networks, Compose (Study Time: 2 hrs)

## Docker Volumes — Persistent Storage

By default, all data inside a container is lost when the container stops or is removed. Volumes solve this — they store data outside the container on the host filesystem, so it persists.

```
docker volume create mydata                  # create a named volume
docker volume ls                             # list volumes
docker run -d -v mydata:/app/data nginx     # mount volume into
container
docker run -d -v /host/path:/container/path nginx  # bind mount (host
folder)
docker volume inspect mydata                # see where data is stored
docker volume rm mydata                    # delete volume
```

📌 **Note:** Use named volumes for databases. Use bind mounts for development (hot reload code changes).

## Docker Networking

By default each container gets its own network namespace. Containers on the same Docker network can communicate with each other by container name.

```
docker network create mynet                 # create custom bridge network
docker network ls                           # list networks
docker run -d --network mynet --name db postgres
docker run -d --network mynet --name app myapp
```

Now 'app' container can reach 'db' container using the hostname 'db'. No need to use IP addresses.

## Docker Compose — Multi-Container Apps

Docker Compose lets you define and run multiple containers together using a single YAML file. One command starts your entire application stack.

```
# docker-compose.yml
version: '3.8'
services:
  app:
    build: .
    ports:
      - '8080:8080'
    environment:
      - DB_HOST=db
      - DB_PORT=5432
    depends_on:
      - db
    networks:
      - appnet
  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: myapp
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: secret
    volumes:
      - pgdata:/var/lib/postgresql/data
    networks:
      - appnet

volumes:
  pgdata:

networks:
  appnet:
```

```
docker-compose up -d      # start all services in background
docker-compose down       # stop and remove containers
docker-compose logs -f    # follow logs from all services
docker-compose ps         # see status of all services
docker-compose exec app bash  # shell inside app container
```

## Azure Container Registry (ACR)

ACR is Azure's private Docker image registry. Instead of pushing to DockerHub (public), you push your private images to ACR. ACR integrates natively with AKS and Azure DevOps pipelines.

```
# Create ACR (one time setup)
az acr create --name mycompanyacr --resource-group myRG --sku Basic

# Log in to ACR
az acr login --name mycompanyacr
```

```
# Build and push to ACR
docker build -t mycompanyacr.azurecr.io/myapp:v1 .
docker push mycompanyacr.azurecr.io/myapp:v1

# List images in ACR
az acr repository list --name mycompanyacr
```

# DAY 10 Kubernetes — Core Concepts (Study Time: 2 hrs)

## What is Kubernetes (K8s)

Kubernetes is a container orchestration platform. When you have hundreds or thousands of containers running across multiple servers, you need something to manage them automatically. Kubernetes does that — it decides where containers run, restarts them if they crash, scales them up and down based on traffic, and manages networking between them.

### Kubernetes Architecture

#### ▶ Control Plane (Master) Components

- **API Server** — the front door to Kubernetes — all commands (kubectl) go through here
- **etcd** — distributed key-value store that holds all cluster state and configuration
- **Scheduler** — decides which worker node should run a new pod based on resources available
- **Controller Manager** — watches the cluster and reconciles actual state with desired state

#### ▶ Worker Node Components

- **kubelet** — agent running on each node — receives pod specs from API server and ensures containers run
- **kube-proxy** — handles networking — maintains network rules for pod communication
- **Container Runtime** — actually runs containers — Docker or containerd

## Core Kubernetes Objects

| Object | What it does |
|---|---|
| Pod | Smallest deployable unit — one or more containers running together, sharing network and storage |
| Deployment | Manages a set of identical pod replicas — handles scaling and rolling updates |
| Service | Stable network endpoint to reach pods — pods can die and be recreated, Service IP stays same |
| ConfigMap | Store non-sensitive configuration data (env vars, config files) separately from containers |
| Secret | Store sensitive data (passwords, API keys, certs) encoded in base64 |
| Namespace | Logical partition of the cluster — separate dev, staging, prod environments |
| Ingress | Routes external HTTP traffic to internal services based on rules (like a reverse proxy) |
| PersistentVolume (PV) | A piece of storage provisioned by an admin — independent of pod lifecycle |
| PersistentVolumeClaim (PVC) | A request for storage by a pod — binds to a matching PV |
| DaemonSet | Ensures a pod runs on every node — used for log collectors, monitoring agents |
| StatefulSet | For stateful apps like databases — maintains stable pod names and storage |
| Job / CronJob | Runs a task to completion — CronJob runs on a schedule like cron |

## kubectl — The Kubernetes CLI

### ▶ Get / Describe

```
kubectl get pods                           # list pods in default
namespace

kubectl get pods -n kube-system           # list pods in kube-system
namespace

kubectl get pods -A                       # list pods in ALL
namespaces

kubectl get deployments                   # list deployments

kubectl get services                      # list services

kubectl get all                           # list everything in current
namespace

kubectl describe pod mypod               # detailed info about a pod

kubectl describe node mynode             # node info including
resource usage
```

### ▶ Logs and Debug

```
kubectl logs mypod                        # logs from pod

kubectl logs mypod -f                    # follow live logs
kubectl logs mypod -c mycontainer        # logs from specific
container in pod
kubectl exec -it mypod -- bash           # shell into running pod
kubectl exec mypod -- cat /etc/config     # run single command in pod
```

▶ **Create / Delete**

```
kubectl apply -f deployment.yaml         # create or update from YAML
file
kubectl delete -f deployment.yaml        # delete resources in YAML
file
kubectl delete pod mypod                 # delete a specific pod
kubectl scale deploy/myapp --replicas=5  # scale to 5 replicas
```

# DAY 11 Kubernetes — Deployments, Services & Ingress (Study Time: 2 hrs)

## Writing Kubernetes YAML Manifests

### Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  namespace: production
  labels:
    app: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myacr.azurecr.io/myapp:v1
        ports:
        - containerPort: 8080
        env:
        - name: DB_HOST
          valueFrom:
            configMapKeyRef:
              name: myapp-config
              key: db_host
        resources:
          requests:
            memory: '128Mi'
            cpu: '100m'
          limits:
            memory: '512Mi'
            cpu: '500m'
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 10
```

## Service Types

| Type | Purpose |
|---|---|

| | |
|---|---|
| ClusterIP | Default. Only accessible inside the cluster. Used for internal service-to-service communication. |
| NodePort | Exposes service on a port of each node (30000–32767). Used for testing, not production. |
| LoadBalancer | Provisions a cloud load balancer. External users access app via a public IP. Used in AKS, EKS. |
| ExternalName | Maps service to an external DNS name. Used to alias external services inside the cluster. |

```yaml
# ClusterIP Service
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  type: ClusterIP
```

## Ingress — Routing External Traffic

An Ingress is an API object that manages external HTTP/S traffic routing to internal Services. It acts like an nginx reverse proxy inside your cluster. You need an Ingress Controller (like nginx-ingress or Azure Application Gateway Ingress Controller) installed for Ingress to work.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myapp-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
```

```
  - host: myapp.company.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
```

## Rolling Updates and Rollbacks

```
# Update deployment to new image
kubectl set image deploy/myapp myapp=myacr.azurecr.io/myapp:v2

# Watch rollout progress
kubectl rollout status deploy/myapp

# View rollout history
kubectl rollout history deploy/myapp

# Rollback to previous version
kubectl rollout undo deploy/myapp
```

```
# Rollback to specific revision
kubectl rollout undo deploy/myapp --to-revision=2
```

# DAY 12 Kubernetes — AKS (Azure Kubernetes Service)

(Study Time: 2 hrs)

## What is AKS

Azure Kubernetes Service is Microsoft's managed Kubernetes service. Azure manages the control plane (API server, etcd, scheduler) for free. You only pay for the worker nodes (VMs). AKS integrates natively with Azure Active Directory, Azure Container Registry, Azure Monitor, and Azure DevOps.

### Create AKS Cluster

```
# Install Azure CLI
az login
az account set --subscription 'My Subscription'

# Create resource group
az group create --name myRG --location eastus

# Create AKS cluster
az aks create \
  --resource-group myRG \
  --name myAKSCluster \
  --node-count 3 \
  --node-vm-size Standard_DS2_v2 \
  --enable-addons monitoring \
  --generate-ssh-keys \
  --attach-acr mycompanyacr
```

```
# Connect kubectl to AKS
az aks get-credentials --resource-group myRG --name myAKSCluster
kubectl get nodes    # verify connection
```

## AKS Node Pools

AKS uses node pools — groups of VMs with the same configuration. You can have multiple node pools: a system pool for Kubernetes system pods, and user pools for your application workloads.

```
# Add a new node pool
az aks nodepool add \
  --resource-group myRG \
  --cluster-name myAKSCluster \
  --name gpupool \
  --node-count 2 \
  --node-vm-size Standard_NC6

# Scale a node pool
az aks nodepool scale \
  --resource-group myRG \
  --cluster-name myAKSCluster \
  --name nodepool1 \
  --node-count 5
```

## AKS Autoscaling

### ▶ Horizontal Pod Autoscaler (HPA)

HPA automatically scales the number of pod replicas based on CPU or memory usage.

```
kubectl autoscale deployment myapp --cpu-percent=70 --min=2 --max=10
kubectl get hpa                        # see HPA status
```

### ▶ Cluster Autoscaler

Cluster Autoscaler automatically adds or removes nodes from the node pool based on whether pods are pending (need more nodes) or nodes are underutilized (can be removed).

```
az aks update \
  --resource-group myRG \
  --name myAKSCluster \
  --enable-cluster-autoscaler \
  --min-count 2 \
  --max-count 10
```

### AKS Integration with ACR

```
# Allow AKS to pull images from ACR
az aks update --name myAKSCluster --resource-group myRG --attach-acr mycompanyacr
```

### AKS Networking

- **kubenet** — simple networking — each node gets a subnet, pods get IPs from a separate CIDR
- **Azure CNI** — pods get IPs from your Azure VNet subnet — recommended for production. Enables direct VNet integration
- **Azure Application Gateway Ingress Controller (AGIC)** — uses Azure Application Gateway as Ingress — provides WAF, SSL termination, autoscaling

# DAY 13 Helm — Kubernetes Package Manager (Study Time: 1.5 hrs)

## What is Helm

Helm is the package manager for Kubernetes. Instead of managing dozens of YAML files for an application, Helm bundles them into a 'chart' — a reusable, versioned, configurable package. Think of Helm like apt/yum for Kubernetes applications.

### Key Helm Concepts

| Term | Meaning |
|---|---|
| Chart | A package of Kubernetes YAML templates — defines how an app is deployed |
| Release | An installed instance of a chart in a cluster. Same chart can have multiple releases. |
| Repository | A collection of charts hosted at a URL (like ArtifactHub, Bitnami) |
| Values | Configuration variables that customize a chart. Passed via values.yaml |
| Template | YAML files with Go template syntax — variables make them reusable |

### Helm Commands

```
helm version                              # check helm is installed
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update                          # refresh repo cache
helm search repo nginx                    # search for nginx charts

# Install a chart (creates a release)
helm install mywebserver bitnami/nginx

# Install with custom values
helm install myapp ./mychart -f custom-values.yaml
helm install myapp ./mychart --set image.tag=v2 --set replicas=3

# List releases
helm list
helm list -A               # all namespaces

# Upgrade a release
helm upgrade myapp ./mychart --set image.tag=v3

# Rollback a release
helm rollback myapp 1     # rollback to revision 1

# Uninstall a release
helm uninstall myapp

# See rendered YAML without installing
helm template myapp ./mychart -f values.yaml
```

## Chart Structure

```
mychart/
  Chart.yaml          # chart metadata (name, version, description)
  values.yaml         # default configuration values
  templates/          # Kubernetes YAML templates
    deployment.yaml
    service.yaml
    ingress.yaml
    _helpers.tpl      # reusable template helpers
```

### values.yaml Example

```yaml
replicaCount: 2
image:
  repository: myacr.azurecr.io/myapp
  tag: 'latest'
  pullPolicy: IfNotPresent
service:
  type: LoadBalancer
  port: 80
resources:
  limits:
    cpu: 500m
    memory: 512Mi
```

## DAY 14 Week 2 Revision + Practice (Study Time: 2 hrs)

### Revision Summary

- **Docker** — Image vs container, Dockerfile layers, volumes, networks, Docker Compose, ACR push
- **Kubernetes** — Pod, Deployment, Service, Ingress, ConfigMap, Secret, HPA, rolling update, rollback
- **AKS** — Cluster creation, node pools, autoscaling, ACR integration, CNI networking
- **Helm** — Chart structure, values.yaml, install, upgrade, rollback commands

## Must-Practice Lab Tasks

- Write a Dockerfile for any app and build it locally
- Run a docker-compose stack with an app + PostgreSQL database
- Push an image to ACR using az acr login and docker push
- Deploy a Deployment + Service to AKS using kubectl apply
- Scale the deployment to 5 replicas and watch pods come up
- Trigger a rolling update to a new image tag and then rollback
- Install nginx using Helm and access it via LoadBalancer IP

# WEEK 3 — Azure DevOps & Infrastructure as Code

## DAY 15 Azure DevOps — Full Overview (Study Time: 2 hrs)

### What is Azure DevOps

Azure DevOps is Microsoft's end-to-end DevOps platform. It includes everything a software team needs to plan, build, test, and deploy software. It is available as a cloud service (dev.azure.com) or on-premises (Azure DevOps Server).

#### Azure DevOps Services

| Service | What it Does |
|---|---|
| Azure Boards | Agile planning — backlogs, sprint boards, user stories, bugs, epics. Like JIRA. |
| Azure Repos | Git-based source control. Supports pull requests, branch policies, code review. |
| Azure Pipelines | CI/CD pipelines — build, test, and deploy code. Supports YAML and classic UI. |
| Azure Artifacts | Package management — host NuGet, npm, Maven, Python packages privately. |
| Azure Test Plans | Manual and exploratory testing management. Relevant for your QA background! |

### Azure Pipelines — Deep Dive

Azure Pipelines is the CI/CD engine. Pipelines are defined in YAML files stored in your Azure Repos or GitHub repository.

#### Key Pipeline Concepts

| Concept | Explanation |
|---|---|
| Trigger | What starts the pipeline — code push, PR, schedule, or manual trigger |
| Agent | The machine that runs your pipeline steps. Can be Microsoft-hosted (cloud) or self-hosted (your VM) |
| Stage | A logical group of jobs (e.g., Build stage, Test stage, Deploy stage) |
| Job | A set of steps that run sequentially on the same agent |
| Step | A single action — a script, a task, or a command |
| Task | Prebuilt pipeline actions from Azure Marketplace (e.g., DotNetCoreCLI, Docker, AzureWebApp) |
| Artifact | Files produced by a pipeline (JAR, Docker image, ZIP) — passed between stages |
| Environment | A deployment target in Azure Pipelines — can require manual approvals before deploy |
| Variable Group | Store shared variables and secrets (Azure Key Vault linked) for use across pipelines |

## Azure Pipeline YAML — Complete Example

```yaml
# azure-pipelines.yml
trigger:
  branches:
    include:
      - main
      - release/*

variables:
  acrName: 'mycompanyacr'
  imageName: 'myapp'
  aksCluster: 'myAKSCluster'
  resourceGroup: 'myRG'

stages:
- stage: Build
  displayName: 'Build and Push Docker Image'
  jobs:
  - job: BuildJob
    pool:
      vmImage: 'ubuntu-latest'
    steps:
    - task: Docker@2
      displayName: 'Build Docker Image'
      inputs:
        command: buildAndPush
        containerRegistry: 'MyACRServiceConnection'
        repository: '$(imageName)'
        tags: '$(Build.BuildId)'
```

```
- stage: DeployStaging
  displayName: 'Deploy to Staging'
  dependsOn: Build
  condition: succeeded()
  jobs:
  - deployment: DeployToAKS
    environment: 'staging'
    strategy:
      runOnce:
        deploy:
          steps:
          - task: KubernetesManifest@0
            inputs:
              action: deploy
              kubernetesServiceConnection: 'AKS-Staging'
              manifests: k8s/staging/*.yaml
              containers: '$(acrName).azurecr.io/$(imageName):
$(Build.BuildId)'

- stage: DeployProd
  displayName: 'Deploy to Production'
  dependsOn: DeployStaging
  jobs:
  - deployment: ProdDeploy
    environment: 'production'   # requires manual approval in Azure
DevOps
    strategy:
      runOnce:
        deploy:
          steps:
```

```
          - task: KubernetesManifest@0
            inputs:
              action: deploy
              kubernetesServiceConnection: 'AKS-Production'
              manifests: k8s/prod/*.yaml
              containers: '$(acrName).azurecr.io/$(imageName):
$(Build.BuildId)'
```

### Service Connections

Service Connections are how Azure Pipelines authenticate to external services (AKS, ACR, Azure subscription, GitHub). Set them up under Project Settings → Service Connections in Azure DevOps portal.

- **Azure Resource Manager** — connects pipeline to your Azure subscription — for deploying to AKS, App Service, etc.
- **Docker Registry** — connects to ACR or DockerHub for pushing/pulling images
- **Kubernetes** — connects directly to a Kubernetes cluster for kubectl/manifest deployments
- **GitHub** — connects to GitHub for source code and triggering pipelines on push

# DAY 16 Azure DevOps — Boards, Repos, Artifacts & Test Plans (Study Time: 1.5 hrs)

## Azure Boards

Azure Boards is an Agile project management tool. Used to plan and track work across sprints.

| Work Item | Description |
|---|---|
| Epic | Large feature or initiative spanning multiple sprints — e.g., 'Migrate to Kubernetes' |
| Feature | A meaningful piece of work within an epic — e.g., 'Set up AKS Cluster' |
| User Story | A specific requirement from user perspective — e.g., 'As a developer, I can deploy via pipeline' |
| Task | Smallest unit of work — assigned to a developer, estimated in hours |
| Bug | Defect tracked and resolved in a sprint |

In DevOps roles, you will use Boards to: link pipeline runs to work items, create work items for failed deployments, and track infrastructure changes as tasks/stories.

## Azure Repos — Branch Policies

Branch policies protect important branches from direct pushes and enforce code quality rules. Always configure these on your main/master branch.

- **Require PR reviewers** — at least 1 or 2 approvals required before merge
- **Require linked work items** — every PR must link to a Board work item for traceability
- **Require passing builds** — CI pipeline must pass before PR can be merged
- **Require comment resolution** — all PR review comments must be resolved before merge
- **Limit merge types** — only allow squash or rebase merges to keep history clean

## Azure Artifacts

Azure Artifacts is a private package registry. Used when your organization creates internal libraries or tools that other projects depend on. Supports npm, NuGet, Maven, Python, and Universal Packages.

```
# Add Artifacts feed to your npm project
npm config set registry https://pkgs.dev.azure.com/myorg/
_packaging/myfeed/npm/registry/


# Publish package to Artifacts
npm publish
```

## Azure Test Plans (Relevant for QA-to-DevOps Transition)

Azure Test Plans manages manual and automated test cases. As someone from QA, this will feel familiar.

- **Test Plan** — a collection of test suites for a sprint or release
- **Test Suite** — grouped test cases — static, requirement-based, or query-based
- **Test Case** — individual test with steps, expected results, and parameters
- **Test Run** — executing a test plan and recording pass/fail results

In DevOps, automated tests from Pipelines can report results directly into Test Plans, giving a unified view of all testing — manual and automated.

## DAY 17 Terraform — Infrastructure as Code (Study Time: 2

hrs)

## What is Infrastructure as Code (IaC)

IaC means you define your cloud infrastructure (VMs, networks, databases, Kubernetes clusters) in code files instead of clicking through the cloud portal. Benefits: version controlled, repeatable, consistent across environments, reviewable via PRs, and can be destroyed and recreated exactly.

## What is Terraform

Terraform is the most popular IaC tool, created by HashiCorp. It is cloud-agnostic — the same tool works for Azure, AWS, and GCP. You write configuration in HCL (HashiCorp Configuration Language) and Terraform creates, updates, or destroys resources to match your definition.

### How Terraform Works

| Command | What it Does |
|---|---|
| terraform init | Download required providers and modules — run once when setting up |
| terraform plan | Show what changes Terraform WILL make (dry run) — always review before apply |
| terraform apply | Create or update resources to match configuration — modifies real infrastructure |
| terraform destroy | Delete all resources defined in configuration |
| terraform validate | Check configuration for syntax errors |
| terraform fmt | Format code to standard style |
| terraform output | Print output values defined in configuration |
| terraform state list | List all resources tracked in Terraform state |

### Terraform Core Concepts

| Concept | Explanation |
|---|---|
| Provider | Plugin that knows how to create resources for a specific cloud. e.g., azurerm, aws, google. |
| Resource | A real infrastructure object you want to create — VM, AKS cluster, VNet, database. |
| Variable | Input parameter for your configuration — makes it reusable across environments. |
| Output | Values exported from Terraform — e.g., the IP of a VM you created. |
| Module | Reusable group of resources — like a function in programming. DRY principle. |
| State File | Terraform's record of what resources it has created. Stored locally or in Azure Blob Storage. |
| Backend | Where state file is stored. Use Azure Storage Account for team/production use. |
| Data Source | Read information about existing resources not managed by this Terraform config. |

## Terraform for Azure — Full Example

```
# main.tf
terraform {
  required_providers {
    azurerm = { source = 'hashicorp/azurerm', version = '~>3.0' }
  }
  backend 'azurerm' {
    resource_group_name  = 'terraform-state-rg'
    storage_account_name = 'tfstatestorage'
    container_name       = 'tfstate'
    key                  = 'prod.terraform.tfstate'
  }
}

provider 'azurerm' {
  features {}
}

resource 'azurerm_resource_group' 'main' {
  name     = var.resource_group_name
  location = var.location
}

resource 'azurerm_virtual_network' 'main' {
  name                = 'myVNet'
  resource_group_name = azurerm_resource_group.main.name
  location            = azurerm_resource_group.main.location
  address_space       = ['10.0.0.0/16']
}
```

```
resource 'azurerm_subnet' 'main' {
  name                 = 'mySubnet'
  resource_group_name  = azurerm_resource_group.main.name
  virtual_network_name = azurerm_virtual_network.main.name
  address_prefixes     = ['10.0.1.0/24']
}

resource 'azurerm_kubernetes_cluster' 'main' {
  name                = 'myAKSCluster'
  location            = azurerm_resource_group.main.location
  resource_group_name = azurerm_resource_group.main.name
  dns_prefix          = 'myaks'

  default_node_pool {
    name       = 'default'
    node_count = 3
    vm_size    = 'Standard_DS2_v2'
    vnet_subnet_id = azurerm_subnet.main.id
  }

  identity {
    type = 'SystemAssigned'
  }
}

# variables.tf
variable 'resource_group_name' { default = 'myRG' }
variable 'location' { default = 'East US' }

# outputs.tf
```

```
output 'aks_cluster_name' {
  value = azurerm_kubernetes_cluster.main.name
}
output 'kube_config' {
  value     = azurerm_kubernetes_cluster.main.kube_config_raw
  sensitive = true
}
```

# DAY 18 Ansible — Configuration Management (Study Time: 2 hrs)

## What is Ansible

Ansible is a configuration management and automation tool. It automates the configuration and management of servers. Unlike Terraform (which creates infrastructure), Ansible configures what runs ON that infrastructure — install packages, configure services, deploy apps, manage users.

Ansible is agentless — it uses SSH to connect to servers. No software needs to be installed on managed nodes. It uses YAML files called Playbooks.

### Ansible vs Terraform

| Feature | Ansible | Terraform |
|---|---|---|
| Purpose | Configure software on servers | Create cloud infrastructure |
| Approach | Procedural (step by step) | Declarative (desired state) |
| Agentless | Yes — uses SSH | Yes — uses cloud APIs |
| State Tracking | No built-in state | Maintains state file |
| When to Use | Install nginx, configure OS, deploy apps | Create VM, VNet, AKS, storage |

### Key Ansible Concepts

| Concept | Meaning |
|---|---|
| Inventory | File listing managed servers — IPs or hostnames grouped by role |
| Playbook | YAML file containing automation tasks to run on servers |
| Task | A single action — install package, copy file, restart service |
| Role | Reusable collection of tasks organized in a folder structure — like a module |
| Handler | A task that only runs when notified — e.g., restart nginx only if config changed |
| Variable | Dynamic values used in playbooks — overridable per host or group |

|   |   |
|---|---|
| Module | Built-in Ansible functions — apt, yum, copy, template, service, command, shell |
| Vault | Encrypt sensitive data (passwords, keys) in playbooks with ansible-vault |

## Inventory File

```
# inventory.ini
[webservers]
web1 ansible_host=10.0.1.10 ansible_user=azureuser
web2 ansible_host=10.0.1.11 ansible_user=azureuser

[databases]
db1 ansible_host=10.0.2.10 ansible_user=azureuser

[all:vars]
ansible_ssh_private_key_file=~/.ssh/id_rsa
```

## Playbook Example — Install and Configure Nginx

```
# setup-nginx.yml
---
- name: Set up Nginx web servers
  hosts: webservers
  become: yes           # run as root (sudo)
  vars:
    nginx_port: 80
    app_version: '1.2.3'

  tasks:
    - name: Update apt cache
      apt:
        update_cache: yes
        cache_valid_time: 3600
    - name: Install Nginx
      apt:
        name: nginx
        state: present

    - name: Copy Nginx config
      template:
        src: templates/nginx.conf.j2
        dest: /etc/nginx/sites-enabled/myapp.conf
      notify: Restart Nginx

    - name: Ensure Nginx is running and enabled
      service:
        name: nginx
        state: started
        enabled: yes

  handlers:
    - name: Restart Nginx
      service:
        name: nginx
        state: restarted
```

### Running Ansible

```
ansible-playbook -i inventory.ini setup-nginx.yml
ansible-playbook -i inventory.ini setup-nginx.yml --check  # dry run
ansible-playbook -i inventory.ini setup-nginx.yml --tags 'install'
ansible all -i inventory.ini -m ping   # ping all servers
ansible webservers -i inventory.ini -m shell -a 'uptime'   # run command on group
```

```
ansible-vault encrypt secrets.yml     # encrypt a file
ansible-vault decrypt secrets.yml     # decrypt a file
```

## DAY 19 Monitoring — Prometheus, Grafana & Azure Monitor (Study Time: 2 hrs)

### Why Monitoring Matters in DevOps

Monitoring lets you know if your app is working before your users complain. In DevOps, you are responsible for not just deploying but keeping the app healthy in production. Monitoring covers three pillars: Metrics (numbers over time), Logs (event records), and Traces (request journeys through microservices).

### Prometheus — Metrics Collection

Prometheus is an open-source monitoring system that scrapes (pulls) metrics from applications and infrastructure. It stores them in a time-series database and allows querying with PromQL.

| Concept | Meaning |
|---|---|
| Time-series | Metrics stored as sequences of timestamped values — e.g., CPU every 15s |
| Scraping | Prometheus pulls metrics from targets by calling their /metrics HTTP endpoint |
| Exporter | A small service that exposes metrics for Prometheus — node_exporter for Linux, kube-state-metrics for K8s |
| PromQL | Prometheus Query Language — used to query and compute metrics for dashboards/alerts |
| Alert Rule | A PromQL expression that fires an alert when the condition is true |
| Alertmanager | Receives alerts from Prometheus and routes them to PagerDuty, Slack, email, etc. |

#### ▶ Common PromQL Examples

```
# CPU usage percentage across all nodes
100 - (avg by (instance) (rate(node_cpu_seconds_total{mode='idle'}[5m])) * 100)
```

```
# Memory usage
(node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) / node_memory_MemTotal_bytes * 100
```

```
# HTTP error rate

rate(http_requests_total{status=~'5..'}[5m]) /
rate(http_requests_total[5m]) * 100
```

```
# Kubernetes pod restarts

increase(kube_pod_container_status_restarts_total[1h]) > 3
```

## Grafana — Visualization

Grafana connects to Prometheus (and other data sources like Azure Monitor, Elasticsearch) and creates dashboards and charts. You build dashboards once and they update in real time.

- **Data Sources** — connect Grafana to Prometheus, Azure Monitor, Elasticsearch, MySQL, etc.
- **Panels** — individual charts on a dashboard — time series, gauge, stat, table, heat map
- **Dashboard** — collection of panels organized to give complete visibility into a system
- **Alerts** — Grafana can also send alerts when panel conditions are met

💡 **Tip:** For AKS monitoring, use the Azure Monitor + Container Insights integration. It automatically collects pod metrics, logs, and node performance without setting up Prometheus yourself.

## Azure Monitor

Azure Monitor is Microsoft's native monitoring platform. It collects telemetry from all Azure resources automatically.

| Feature | Purpose |
|---|---|
| Metrics | Numerical data from Azure resources — CPU%, memory, request count, latency. Stored 93 days. |
| Logs | Text event data from resources — stored in Log Analytics Workspace. Queried with KQL. |
| Container Insights | Monitoring for AKS — pod metrics, container logs, node health, all from one dashboard. |
| Application Insights | APM for applications — request tracing, exception tracking, dependency monitoring. |
| Alerts | Rules on metrics or logs that send notifications to email, SMS, PagerDuty, webhook. |
| Action Groups | Defines WHO to notify and HOW when an alert fires — email, SMS, ITSM ticket. |

### KQL — Kusto Query Language (for Azure Monitor Logs)

```
// Pod errors in last 1 hour
```

```
ContainerLog
| where TimeGenerated > ago(1h)
| where LogEntry contains 'ERROR'
| project TimeGenerated, ContainerName, LogEntry
| order by TimeGenerated desc

// Average CPU by node
Perf
| where ObjectName == 'K8SNode'
| where CounterName == 'cpuUsageNanoCores'
| summarize avg(CounterValue) by Computer, bin(TimeGenerated, 5m)
```

# DAY 20 Logging — ELK Stack & Azure Log Analytics

(Study Time: 1.5 hrs)

## The ELK Stack

ELK stands for Elasticsearch, Logstash, and Kibana. Together they form a powerful log management platform.

| Component | Role |
|---|---|
| Elasticsearch | Distributed search and analytics engine. Stores and indexes logs for fast full-text search. |
| Logstash | Log pipeline — ingests logs from many sources, transforms them, sends to Elasticsearch. |
| Kibana | Web UI for Elasticsearch. Search logs, build dashboards, set up alerts. |
| Beats / Filebeat | Lightweight log shipper. Runs on each server/container and sends logs to Logstash or Elasticsearch. |
| Fluentd / Fluent Bit | Alternative log shippers — very popular in Kubernetes. Lighter weight than Logstash. |

### Kubernetes Log Collection Architecture

In Kubernetes, Fluent Bit (DaemonSet on every node) collects container logs and ships them to Elasticsearch or Azure Log Analytics. Kibana or Azure Monitor Logs provides the search UI.

## Structured Logging

Logs should be structured (JSON format) not plain text. Structured logs are parseable, searchable, and consistent.

```
# Bad log (plain text — hard to parse)
2024-01-15 14:23:01 ERROR UserService failed to process user 1234
after 3 retries


# Good log (JSON — easily indexed and searched)
{"timestamp":"2024-01-15T14:23:01Z","level":"ERROR","service":"UserSer
vice",
 "message":"failed to process
user","userId":1234,"retries":3,"duration_ms":450}
```

# DAY 21 Week 3 Revision + DevSecOps Intro (Study Time: 2 hrs)

## Week 3 Summary

- **Azure DevOps** — Boards, Repos, Pipelines (YAML), Artifacts, Test Plans, Service Connections, Environments
- **Terraform** — init, plan, apply, destroy, providers, resources, variables, outputs, modules, remote state
- **Ansible** — Inventory, Playbook, Tasks, Handlers, Roles, Vault, ansible-playbook command
- **Monitoring** — Prometheus metrics + PromQL, Grafana dashboards, Azure Monitor, Container Insights, KQL
- **Logging** — ELK Stack components, Fluent Bit in K8s, structured JSON logging

## DevSecOps — Security in the Pipeline

DevSecOps means integrating security into every stage of the DevOps pipeline, not just at the end. Shift security left — find and fix vulnerabilities as early as possible, ideally during development not after deployment.

| Type | What it Does |
|---|---|
| SAST | Static Application Security Testing — scans source code for vulnerabilities. Tools: SonarQube, Checkmarx, Semgrep. |

| | |
|---|---|
| DAST | Dynamic Application Security Testing — tests running app by sending malicious inputs. Tools: OWASP ZAP. |
| Container Scanning | Scans Docker images for CVEs in base image and dependencies. Tools: Trivy, Azure Defender for Containers. |
| Dependency Scanning | Checks third-party libraries for known vulnerabilities. Tools: OWASP Dependency Check, Snyk. |
| IaC Scanning | Scans Terraform/ARM templates for misconfigurations. Tools: Checkov, tfsec. |
| Secrets Detection | Prevent passwords/API keys from being committed to Git. Tools: GitGuardian, truffleHog, git-secrets. |

📌 **Note:** Never store secrets (passwords, API keys, connection strings) in code or Dockerfiles. Use Azure Key Vault or Kubernetes Secrets.

## Azure Key Vault

Azure Key Vault securely stores and manages secrets, encryption keys, and certificates. Applications fetch secrets from Key Vault at runtime instead of having them hardcoded.

```
# Create Key Vault

az keyvault create --name myKeyVault --resource-group myRG --location eastus


# Add a secret

az keyvault secret set --vault-name myKeyVault --name 'DB-PASSWORD' --value 'supersecret'


# Get a secret (in pipeline or app)

az keyvault secret show --vault-name myKeyVault --name 'DB-PASSWORD' --query value
```

# WEEK 4 — Advanced Topics, SRE & Final Prep

## DAY 22 Azure Networking — VNet, NSG, Load Balancer

(Study Time: 2 hrs)

## Azure Virtual Network (VNet)

A VNet is the fundamental building block of your private network in Azure. All your Azure resources (VMs, AKS, databases) run inside a VNet and communicate privately without going through the internet.

| Component | Purpose |
|---|---|
| VNet | Your private network in Azure. Define your own IP address space (e.g., 10.0.0.0/16). |
| Subnet | A range of IP addresses within a VNet (e.g., 10.0.1.0/24). Resources are placed in subnets. |
| NSG (Network Security Group) | Firewall rules that allow or deny traffic to/from subnets or NICs. Rules have priority. |
| VNet Peering | Connect two VNets so they can communicate privately — even across regions. |
| VPN Gateway | Encrypted tunnel between Azure VNet and on-premises network. |
| Private Endpoint | A private IP within your VNet to access Azure services (Storage, SQL) without internet. |
| Azure Bastion | Secure RDP/SSH access to VMs from browser without public IP on the VM. |
| UDR (User Defined Route) | Custom routing rules to control traffic flow — e.g., route internet traffic through firewall. |

### NSG Rules

NSG rules are evaluated by priority (lowest number = highest priority). Each rule specifies: direction (inbound/outbound), protocol, source, destination, port, and action (allow/deny).

```
# Allow HTTPS inbound (priority 100)
az network nsg rule create \
  --resource-group myRG \
  --nsg-name myNSG \
  --name AllowHTTPS \
  --priority 100 \
  --protocol Tcp \
  --direction Inbound \
  --source-address-prefix '*' \
  --destination-port-range 443 \
  --access Allow
```

## Azure Load Balancer vs Application Gateway

| Feature | Azure Load Balancer | Azure Application Gateway |
|---|---|---|
| Layer | Layer 4 (Transport) | Layer 7 (Application / HTTP) |
| Traffic Type | Any TCP/UDP traffic | HTTP, HTTPS, WebSocket traffic only |
| Routing | Based on IP + Port | Based on URL path, hostname, headers |
| SSL Termination | No | Yes — decrypts HTTPS at gateway |
| WAF | No | Yes — built-in Web Application Firewall |
| Use Case | Non-HTTP apps, VMs load balancing | Web apps, APIs, microservices HTTP routing |

# DAY 23 Azure Services Every DevOps Engineer Must Know (Study Time: 2 hrs)

## Azure Compute

| Service | Purpose & When to Use |
|---|---|
| Azure VMs | Virtual machines. Use for: self-hosted build agents, legacy apps, anything needing full OS control. |
| Azure App Service | PaaS for web apps. Deploy code directly without managing servers. Auto-scaling built in. |
| Azure Functions | Serverless compute. Run code in response to events (HTTP trigger, timer, queue message). Pay per execution. |
| Azure Container Instances (ACI) | Run single containers without Kubernetes. Good for batch jobs, test containers. |
| Azure Kubernetes Service (AKS) | Managed Kubernetes. For microservices, complex containerized applications. |
| Azure Virtual Machine Scale Sets | Auto-scaling group of identical VMs. Automatically add/remove VMs based on load. |

## Azure Storage

| Service | Purpose |
|---|---|
| Azure Blob Storage | Object storage for unstructured data — images, videos, backups, Terraform state, build artifacts. |
| Azure Files | Managed file share (SMB/NFS). Mount as a drive from VMs or containers. |
| Azure Disk | Block storage attached to VMs — like a virtual hard drive. |
| Azure Data Lake Storage | Blob storage optimized for big data analytics workloads. |

| | |
|---|---|
| Azure Queue Storage | Message queue for decoupling services — up to 64KB per message. |
| Azure Table Storage | NoSQL key-value store for structured, non-relational data. |

## Azure Database Services

| Service | Use Case |
|---|---|
| Azure SQL Database | Fully managed Microsoft SQL Server as a service. |
| Azure Database for PostgreSQL | Managed PostgreSQL. Very popular for modern apps. |
| Azure Database for MySQL | Managed MySQL service. |
| Azure Cosmos DB | Globally distributed NoSQL database — multiple APIs (SQL, MongoDB, Cassandra). |
| Azure Cache for Redis | Managed Redis caching service — dramatically speeds up app performance. |

## Azure Identity & Security

| Service | Purpose |
|---|---|
| Azure Active Directory (Entra ID) | Microsoft's identity platform. Manages users, groups, applications, and access control. |
| Managed Identity | An identity for Azure resources (VM, AKS, Function) to authenticate to other services WITHOUT credentials. Best practice. |
| RBAC (Role-Based Access Control) | Assign roles (Reader, Contributor, Owner) to users or identities on specific scopes (resource, RG, subscription). |
| Azure Key Vault | Store secrets, keys, and certificates. Never put passwords in code — use Key Vault. |
| Azure Defender / Defender for Cloud | Cloud security posture management + threat protection for all Azure services. |
| Azure Policy | Enforce organizational standards — e.g., all resources must have tags, VMs must be in specific regions. |

# DAY 24 Deployment Strategies (Study Time: 1.5 hrs)

## Deployment Strategies — Know All of These for Interviews

### Rolling Deployment

Pods are updated gradually, one (or a few) at a time. Old pods are replaced with new version pods until all are updated. Zero downtime if the new version is healthy, but both old and new versions serve traffic simultaneously during the rollout.

```
# Kubernetes rolling update config
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1         # max extra pods above desired count during update
    maxUnavailable: 0   # no pods can be unavailable during update
```

### Blue-Green Deployment

Two identical environments — Blue (current production) and Green (new version). Traffic is switched from Blue to Green all at once. Instant rollback possible by switching traffic back to Blue. Requires double the infrastructure during deployment.

In Kubernetes: two separate Deployments (blue and green). A Service selector is updated to point to the green deployment when ready.

### Canary Deployment

New version is released to a small percentage of users (e.g., 5%) while 95% still use the old version. Monitor for errors. Gradually increase traffic to new version if healthy. Much safer than blue-green for risky changes.

```
# Example: 2 old pods, 1 new canary pod = 33% traffic to new version
kubectl scale deploy/myapp-v1 --replicas=2
kubectl scale deploy/myapp-v2 --replicas=1
```

### Feature Flags (Feature Toggles)

New code is deployed to production but hidden behind a flag that is switched off. Enable the flag for specific users, regions, or a percentage of traffic. Allows code to be deployed without being 'released'. Separate deployment from release.

### Recreate Deployment

All old pods are killed first, then new pods are created. Results in brief downtime. Only use for development environments or apps that cannot run two versions simultaneously.

| Strategy | How it Works | Downtime | Complexity |
|---|---|---|---|
| Rolling | Gradual pod replacement | No | Easy |
| Blue-Green | Instant full switch | No | Needs 2x infra |
| Canary | Gradual traffic shift | No | Complex routing |
| Recreate | Kill all then create new | Yes | Simplest |

# DAY 25 SRE — Site Reliability Engineering (Study Time: 2 hrs)

## What is SRE

Site Reliability Engineering was created at Google. SREs apply software engineering principles to operations and reliability work. The goal is to make systems more reliable, scalable, and efficient through automation, measurement, and collaborative practices.

In practice, SRE teams define reliability targets, build automation to replace manual operations, respond to incidents with data-driven decisions, and conduct post-mortems (blameless) to prevent recurrence.

### Key SRE Concepts

| Term | Explanation |
|---|---|
| SLI (Service Level Indicator) | A measurement of your service's behavior. Example: percentage of HTTP requests completed successfully in under 500ms. |
| SLO (Service Level Objective) | Your reliability target. Example: 99.9% of requests must succeed within 500ms measured over 30 days. |
| SLA (Service Level Agreement) | A contractual commitment to customers. Usually less strict than SLO. SLO breach → SLA breach. |
| Error Budget | 100% - SLO = acceptable failure budget. If SLO is 99.9%, error budget is 0.1% downtime per month (~43 minutes). Spend it on risky deployments. |
| Toil | Manual, repetitive, automatable operational work. SRE goal is to reduce toil below 50% of time. |
| Blameless Post-Mortem | After an incident, analyze what went wrong and improve systems/processes — never blame individuals. |
| On-Call | Rotational responsibility to respond to incidents. SREs use alerting systems (PagerDuty) for on-call. |

### Incident Management Process

- **Detection** — alert fires — monitoring detects anomaly (high error rate, latency spike, pod crash loops)
- **Triage** — on-call engineer assesses severity. Is it affecting users? How many? Is it growing?
- **Mitigation** — restore service as fast as possible — rollback, restart pods, increase replicas, disable feature flag
- **Root Cause Analysis** — investigate WHY it happened — not who. Look at metrics, logs, recent changes
- **Post-Mortem** — document: timeline, impact, root cause, mitigation steps, action items to prevent recurrence

### SLI/SLO Example for a Web App

| Metric | Definition / Target |
|---|---|
| Availability SLI | successful_requests / total_requests * 100 |
| Availability SLO | >= 99.9% over 30 days |
| Latency SLI | % requests completed in < 300ms |
| Latency SLO | >= 95% requests under 300ms |
| Error Budget | 0.1% = 43.8 minutes per month of allowed downtime |

# DAY 26 GitOps — ArgoCD & Flux (Study Time: 1.5 hrs)

## What is GitOps

GitOps is an operating model where Git is the single source of truth for both application code AND infrastructure configuration. Desired state of your Kubernetes cluster is stored in Git. A GitOps controller continuously syncs the cluster to match Git. Any change to the cluster must go through a Git commit and PR — no manual kubectl changes in production.

### GitOps Principles

- **Declarative** — entire desired system state is expressed declaratively in YAML files in Git
- **Versioned** — all changes are tracked in Git history — full audit trail and easy rollbacks
- **Automated** — approved Git changes are applied automatically to the cluster by a controller
- **Continuously Reconciled** — controller watches cluster and Git — if they drift, it corrects back to Git state

## ArgoCD

ArgoCD is the most popular GitOps tool for Kubernetes. It runs inside your cluster and watches a Git repository. When you push a change to the Git repo, ArgoCD detects it and syncs the cluster to match.

```
# ArgoCD Application manifest
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: myapp
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://dev.azure.com/org/project/_git/k8s-manifests
    targetRevision: main
    path: manifests/production
  destination:
    server: https://kubernetes.default.svc
    namespace: production
  syncPolicy:
    automated:
      prune: true      # delete resources removed from Git
      selfHeal: true   # revert manual changes to cluster
```

**Tip:** In an interview, describe GitOps as: 'Git is the source of truth. Changes go through PRs. ArgoCD auto-deploys approved changes. Manual cluster changes are auto-reverted. Full audit trail in Git history.'

# DAY 27 Azure DevOps Pipeline — Advanced Patterns

(Study Time: 2 hrs)

## Pipeline Templates — Reusable Pipelines

Instead of repeating pipeline code across multiple projects, define templates once and reuse them. Templates are YAML files stored in a central repo.

```
# In main azure-pipelines.yml
stages:
- template: pipeline-templates/build-docker.yml@templates
  parameters:
    imageName: 'myapp'
    acrName: 'mycompanyacr'
```

```
# In templates repo — pipeline-templates/build-docker.yml
parameters:
- name: imageName
  type: string
- name: acrName
```

```
  type: string

stages:
- stage: Build
  jobs:
  - job: Docker
    steps:
    - task: Docker@2
      inputs:
        command: buildAndPush
        repository: '${{ parameters.imageName }}'
        containerRegistry: 'MyACR'
        tags: '$(Build.BuildId)'
```

### Pipeline Approvals and Gates

In Azure DevOps Environments, you can configure approvals that must be granted by a named person before a pipeline stage proceeds. This is used to require manual approval before production deployments.

- **Pre-deployment approvals** — a named person must click Approve before the stage starts
- **Branch control gate** — deployment only allowed from specific branches (e.g., only from main)
- **Business hours gate** — deployment only allowed during certain hours — prevent Friday evening deploys
- **Invoke REST API gate** — check an external system before proceeding — e.g., check if maintenance window is open

## DAY 28 Troubleshooting — Common DevOps Issues

(Study Time: 2 hrs)

### Kubernetes Troubleshooting

#### Pod Not Starting — Debugging Steps

```
kubectl get pods                        # check pod status
kubectl describe pod mypod             # events section shows WHY it
failed
kubectl logs mypod                     # application logs
kubectl logs mypod --previous          # logs from last crashed
container
```

| Status | Likely Cause & Fix |
|---|---|
| Pending | No node available to schedule. Check: resource limits too high, node taints, node capacity. |
| CrashLoopBackOff | Container starts and immediately crashes. Check: kubectl logs. Usually app error, wrong config, missing env var. |
| ImagePullBackOff | Cannot pull Docker image. Check: image name/tag correct, ACR credentials configured, network access. |
| OOMKilled | Container exceeded memory limit. Fix: increase memory limit in deployment YAML. |
| Error | Container exited with error. Check logs for specific error message. |

### Pipeline Troubleshooting

- **Build fails** — check compile errors, missing dependencies, wrong JDK version on agent
- **Docker push fails** — check service connection credentials, ACR permissions, network firewall
- **kubectl fails in pipeline** — check kubeconfig/service connection is correct, namespace exists, RBAC permissions
- **Tests fail intermittently** — flaky tests — investigate test dependencies, shared state, timing issues

## Linux Performance Troubleshooting

```
# High CPU — which process?
top -b -n1 | head -20
```

```
# High Memory — what is using RAM?
free -h
ps aux --sort=-%mem | head -10
```

```
# Disk full — what is using space?
df -h
```

```
du -sh /* 2>/dev/null | sort -rh | head -10

# Network issue — is port open?
telnet server 5432
curl -v http://service:8080/health

# System logs for errors
journalctl -xe | tail -50
dmesg | tail -20
```

## DAY 29 Final Interview Preparation (Study Time: 2 hrs)

### Top Topics Interviewers Always Ask

| Question | Answer Summary |
|---|---|
| What is CI/CD? Explain pipeline stages. | Build → Test → Scan → Package → Deploy to staging → E2E test → Deploy to prod |
| Docker vs VM? | Containers share OS kernel, lighter, faster. VMs have full OS, stronger isolation. |
| What is Kubernetes? | Container orchestration — scheduling, scaling, self-healing, networking of containers across nodes. |
| Explain a Kubernetes deployment | Manages pod replicas, rolling updates, rollback. Selector matches pods via labels. |
| What is a Service in K8s? | Stable network endpoint. Pods have changing IPs, Services provide consistent access. |
| Terraform state? | Records what Terraform created. Stored remotely in Azure Blob. Used to detect drift. |
| Blue-green vs canary? | Blue-green: instant switch. Canary: gradual traffic shift. Both allow zero-downtime deploy. |
| SLO vs SLI? | SLI = measurement (actual %). SLO = target (e.g., >= 99.9%). SLA = contractual commitment. |
| How do you handle secrets? | Azure Key Vault / Kubernetes Secrets. Never in code, never in Dockerfiles. Use Managed Identity. |
| What is GitOps? | Git is source of truth. ArgoCD syncs cluster to Git state. All changes via PR. Auto-reverts drift. |

### Scenario-Based Questions — How to Answer

#### 'Walk me through how you would deploy a new version of an app to

### production'

Structure your answer as: 1) Developer commits code → 2) PR created, pipeline triggers → 3) Build and unit tests run → 4) Docker image built and pushed to ACR → 5) Deploy to AKS staging → 6) Integration and smoke tests → 7) Manual approval gate → 8) Canary/rolling deploy to production → 9) Monitor metrics and error rate for 15 minutes → 10) Full rollout or rollback if issues detected.

### 'A pod is in CrashLoopBackOff — what do you do?'

Say: kubectl describe pod to check events. kubectl logs --previous to see crash reason. Most common causes: wrong environment variable, missing ConfigMap/Secret, port conflict, app start error. Fix the config, update deployment. If urgent — rollback to last working image version.

### 'Production is down — what do you do?'

Say: First — check monitoring dashboard for what changed. Second — check recent deployments. If a recent deployment caused it, rollback immediately (kubectl rollout undo). While restoring service, notify stakeholders. After service restored — root cause analysis. Post-mortem within 24-48 hours.

# DAY 30 Final Day — Revision, Gaps & Exam Strategy

(Study Time: 2 hrs)

## Complete Topic Checklist

| Topic | Days Covered |
|---|---|
| Linux basics, permissions, SSH, cron jobs, shell scripting | Day 1-3 |
| Git workflow, branching, Azure Repos, branch policies | Day 4 |
| Networking: TCP, HTTP, DNS, ports, VNet, NSG | Day 5, Day 22 |
| CI/CD concepts, Jenkins Jenkinsfile, GitHub Actions YAML | Day 6 |
| Docker: Dockerfile, commands, Compose, ACR, volumes | Day 8-9 |
| Kubernetes: objects, kubectl, Deployments, Services, Ingress | Day 10-12 |
| AKS: cluster create, node pools, autoscaling, integration | Day 12 |
| Helm: charts, values.yaml, install/upgrade/rollback | Day 13 |
| Azure DevOps: Boards, Repos, Pipelines, Artifacts, Test Plans | Day 15-16 |

| | |
|---|---|
| Terraform: providers, resources, variables, state, AKS example | Day 17 |
| Ansible: inventory, playbook, tasks, handlers, vault | Day 18 |
| Monitoring: Prometheus, PromQL, Grafana, Azure Monitor, KQL | Day 19 |
| Logging: ELK, Fluent Bit, structured logging | Day 20 |
| DevSecOps: SAST, DAST, container scanning, Key Vault | Day 21 |
| Deployment strategies: rolling, blue-green, canary, recreate | Day 24 |
| SRE: SLI, SLO, SLA, error budget, on-call, post-mortem | Day 25 |
| GitOps: ArgoCD, sync policy, drift reconciliation | Day 26 |
| Troubleshooting: pod statuses, pipeline failures, Linux perf | Day 28 |

## Exam and Interview Day Tips

- **Before interview** — revise Day 29 quick-answer table. Get 8 hours sleep. Eat well.
- **During technical questions** — think out loud. Interviewers want to hear your reasoning process not just the final answer.
- **If you don't know** — say 'I haven't worked with that specific tool but I know X which is similar and I would approach it by...' — never bluff.
- **Practical rounds** — write code/YAML neatly, add comments, explain each line. Even wrong answers with good reasoning score well.
- **Questions to ask them** — always ask 2-3 questions — it shows genuine interest and preparation.

## Your Learning is Never Complete

DevOps evolves constantly. After you land your first DevOps role, focus on: becoming deep in one cloud (Azure), getting certified (AZ-400 Azure DevOps Engineer, AZ-104 Azure Administrator, CKA), contributing to your team's pipeline improvements, and building habits around monitoring and incident response.

***You started as a Manual Tester. You are becoming a DevOps Engineer.***

*Every day of studying is compounding. Keep going.*