DevOps Cheat Sheet

5. Cloud Services

- AWS (EC2, S3, IAM, VPC, Lambda)
- Azure (VMs, Storage, AKS, Functions)
- GCP (Compute Engine, GKE, Cloud Run)

6. Configuration Management

- Chef (recipes, cookbooks)
- Puppet (manifests, modules)
- SaltStack (states, grains)

7. Monitoring & Logging

- Prometheus & Grafana (metrics, alerts, visualization)
- ELK Stack (Elasticsearch, Logstash, Kibana)
- Datadog
- New Relic

8. Security & Compliance

- SonarQube (code analysis)
- Trivy (container vulnerability scanning)
- OWASP Dependency-Check

9. Networking, Ports & Load Balancing

- Networking Basics
- Ports
- Nginx (Reverse Proxy & Load Balancing)
- Apache (reverse proxy, load balancing)
- HAProxy (Load Balancing)
- Kubernetes Ingress Controller (For Managing External Traffic)
- Practical Examples: Docker for Nginx, Apache, HAProxy, and Kubernetes Ingress

10. Database Cheat Sheet

---

1. System Administration & Scripting

Linux commands

1. File Management

- ls - List directory contents.
- cd - Change directory.
- pwd - Print working directory.
- cp - Copy files or directories.
- mv - Move or rename files or directories.
- rm - Remove files or directories.
- touch - Create a new empty file.
- mkdir - Create a new directory.
- rmdir - Remove an empty directory.
- cat - Concatenate and display file contents.
- head - Display the first few lines of a file.
- tail - Display the last few lines of a file.
- chmod - Change file permissions.
- chown - Change file ownership.
- find - Search for files in a directory hierarchy.
- locate - Find files by name.
- grep - Search text using patterns.
- diff - Compare two files line by line.
- tar - Archive files.
- zip/unzip - Compress and extract files.

- scp - Securely copy files over SSH.

- **ls**: List files and directories.

  ls -l # Long listing with details

- **cd**: Change directory.

  cd /home/swapna # Move to /home/swapna directory

- **pwd**: Show current directory.

  pwd

- **cp**: Copy files.

  cp file1.txt /tmp # Copy file1.txt to /tmp directory

- **mv**: Move or rename files.

  mv oldname.txt newname.txt # Rename file
  mv file1.txt /tmp # Move to /tmp directory

- **rm**: Remove files or directories.

  rm file1.txt # Remove a file
  rm -rf /tmp/old_directory # Remove directory and contents

- **mkdir**: Create directories.

  mkdir new_folder # Create a directory called new_folder

- **cat**: Display file contents.

cat file.txt

2. System Information and Monitoring

- top - Display running processes and system usage.
- htop - Interactive process viewer.
- ps - Display current processes.
- df - Show disk space usage.
- du - Show directory space usage.
- free - Show memory usage.
- uptime - Show system uptime.
- uname - Show system information.
- whoami - Display the current logged-in user.
- lsof - List open files and associated processes.
- vmstat - Report virtual memory statistics.
- iostat - Report I/O statistics.
- netstat - Display network connections and routing tables.
- ifconfig - Display or configure a network interface.
- ping - Check network connectivity.
- traceroute - Track the route packets take to a destination.

- **top**: View running processes.

  top

- **df**: Show disk usage.

  df -h # Human-readable format

- **free**: Display memory usage.

  free -m # Show memory in MB

- **uptime**: Check system uptime.

  uptime

3. Package Management (Ubuntu/Debian)

- apt-get update - Update package lists.
- apt-get upgrade - Upgrade all packages.
- apt-get install - Install packages.
- apt-get remove - Remove packages.
- dpkg - Install, remove, and manage individual Debian packages.
- **apt-get**: Install, remove, or update packages.

  ```
  sudo apt-get update         # Update package lists

  sudo apt-get install nginx  # Install NGINX
  ```

4. User and Permission Management

- useradd - Add a new user.
- userdel - Delete a user.
- usermod - Modify a user.
- passwd - Change user password.
- groupadd - Create a new group.
- groupdel - Delete a group.
- groups - Show groups of a user.
- su - Switch user.
- sudo - Execute a command as another user, usually root.

- **useradd**: Add a new user.

  ```
  sudo useradd -m newuser    # Create a new user with a home directory
  ```

- **chmod**: Change file permissions.

  ```
  chmod 755 script.sh    # Set permissions for owner and others
  ```

- **chown**: Change file owner.

  ```
  sudo chown newuser file.txt    # Change ownership to newuser
  ```

5. Networking

- curl - Transfer data from or to a server.
- wget - Download files from the internet.
- ssh - Secure shell to a remote server.
- telnet - Connect to a remote machine.
- nslookup - Query DNS records.
- dig - DNS lookup utility.
- iptables - Configure firewall rules.
- firewalld - Firewall management (CentOS/RHEL).
- hostname - Show or set the system hostname.
- **ping**: Check connectivity to a host.

  ping google.com

- **curl**: Send HTTP requests.

  curl https://example.com

- **ifconfig**: View network interfaces.

  ifconfig

6. Process Management

- kill - Send a signal to a process.
- killall - Kill processes by name.
- pkill - Kill processes by pattern matching.
- bg - Move a job to the background.
- fg - Bring a job to the foreground.
- jobs - List background jobs.

- **ps**: Show running processes.

  ps aux | grep nginx # List processes related to nginx

- **kill**: Terminate a process by PID.

  kill 1234 # Kill process with PID 1234

- **pkill**: Kill processes by name.

  pkill nginx # Kill all nginx processes

7. Disk Management

- fdisk - Partition a disk.
- mkfs - Make a filesystem.
- mount - Mount a filesystem.
- umount - Unmount a filesystem.
- lsblk - List block devices.
- blkid - Print block device attributes.
- **fdisk**: Manage disk partitions.

  sudo fdisk -l # List disk partitions

- **mount**: Mount a filesystem.

  sudo mount /dev/sdb1 /mnt # Mount device sdb1 to /mnt

8. Text Processing

- awk - Pattern scanning and processing.
- sed - Stream editor for modifying text.
- sort - Sort lines of text files.
- uniq - Report or omit repeated lines.
- cut - Remove sections from each line of files.
- wc - Word, line, character count.
- tr - Translate or delete characters.
- nl - Number lines of files.

- **grep**: Search text.

  grep "error" /var/log/syslog # Search for "error" in syslog

- **awk**: Process text with patterns.

  awk '{print $1}' file.txt # Print the first column of each line

- **sed**: Edit text in streams.

  sed 's/old/new/g' file.txt # Replace "old" with "new"

9. Logging and Auditing

- dmesg - Print or control kernel ring buffer.
- journalctl - Query the systemd journal.
- logger - Add entries to the system log.
- last - Show listing of last logged-in users.
- history - Show command history.
- tail -f - Monitor logs in real time.

- **tail**: View end of file in real time.

  tail -f /var/log/nginx/access.log # Follow NGINX access log

- **journalctl**: View system logs.

  sudo journalctl -u nginx # Logs for NGINX service

10. Archiving and Backup

- tar - Archive files.
- rsync - Synchronize files and directories.
- **tar**: Archive files.

  ```
  tar -cvf archive.tar /path/to/files    # Create an archive
  tar -xvf archive.tar                   # Extract an archive
  ```

- **rsync**: Sync files and directories.

  ```
  rsync -avz /source /destination    # Sync with compression and archive mode
  ```

11. Shell Scripting

- echo - Display message or text.
- read - Read input from the user.
- export - Set environment variables.
- alias - Create shortcuts for commands.
- sh, - Execute shell scripts.
- **echo**: Display text.

  ```
  echo "Hello, DevOps!"    # Print message
  ```

- **export**: Set environment variables.

  ```
  export PATH=$PATH:/new/path    # Add to PATH variable
  ```

12. System Configuration and Management

- crontab - Schedule periodic tasks.
- systemctl - Control the systemd system and service manager.
- service - Start, stop, or restart services.
- timedatectl - Query and change the system clock.
- reboot - Restart the system.
- shutdown - Power off the system.

- **crontab**: Schedule tasks.

  ```
  crontab -e    # Edit the crontab file
  # Example entry: 0 2 * * * /path/to/backup.sh
  ```

- **systemctl**: Control services.

  ```
  sudo systemctl restart nginx    # Restart NGINX
  ```

13. Containerization & Virtualization

- docker - Manage Docker containers.
- kubectl - Manage Kubernetes clusters.
- **docker**: Manage Docker containers.

  ```
  docker ps    # List running containers
  docker run -d -p 8080:80 nginx    # Run NGINX container
  ```

- **kubectl**: Manage Kubernetes clusters.

  ```
  kubectl get pods    # List all pods
  kubectl apply -f deployment.yaml    # Deploy configuration
  ```

14. Git Version Control

- git status - Show the status of changes.
- git add - Add files to staging.
- git commit - Commit changes.
- git push - Push changes to a remote repository.
- git pull - Pull changes from a remote repository.
- git clone - Clone a repository.

  git commit -m "<message>" - Commit changes with a descriptive message.

git push <remote> <branch> - Push changes to a remote repository.
git pull <remote> <branch> - Pull changes from a remote repository.
git clone <repository> - Clone a repository.
git remote - Manage set of tracked repositories.

- git remote -v - Show URLs of remote repositories.
- git remote add <name> <url> - Add a new remote repository.
- git remote remove <name> - Remove a remote repository by name.
- git remote rename <old-name> <new-name> - Rename a remote repository.

15. Others

- env - Display environment variables.
- date - Show or set the system date and time.
- alias - Create command shortcuts.
- source - Execute commands from a file in the current shell.
- sleep - Pause for a specified amount of time.

16. Network Troubleshooting and Analysis

- **traceroute**: Track packet route.

  traceroute google.com # Show hops to google.com

- **netstat**: View network connections, routing tables, and more.

  netstat -tuln # Show active listening ports with protocol info

- **ss**: Display socket statistics (modern alternative to netstat).

  ss -tuln # Show active listening ports

- **iptables**: Manage firewall rules.

sudo iptables -L # List current iptables rules

17. Advanced File Management

- **find**: Search files by various criteria.

  find /var -name "*.log" # Find all .log files under /var

  find /home -type d -name "test" # Find directories named "test"

- **locate**: Quickly find files by name.

  locate apache2.conf # Locate apache2 configuration file

18. File Content and Manipulation

- **split**: Split files into parts.

  split -l 500 largefile.txt smallfile # Split file into 500-line chunks

- **sort**: Sort lines in files.sort file.txt # Sort lines alphabetically

  sort -n numbers.txt # Sort numerically

- **uniq**: Remove duplicates from sorted files.

  sort file.txt | uniq # Remove duplicate lines

19. Advanced Shell Operations

- **xargs**: Build and execute commands from standard input.

  find . -name "*.txt" | xargs rm # Delete all .txt files

- **tee**: Read from standard input and write to standard output and files.

  ```
  echo "new data" | tee file.txt      # Write output to file and terminal
  ```

20. Performance Analysis

- **iostat**: Display CPU and I/O statistics.

  ```
  iostat -d 2      # Show disk I/O stats every 2 seconds
  ```

- **vmstat**: Report virtual memory stats.

  ```
  vmstat 1 5       # Display 5 samples at 1-second intervals
  ```

- **sar**: Collect and report system activity information.

  ```
  sar -u 5 5       # Report CPU usage every 5 seconds
  ```

21. Disk and File System Analysis

- **lsblk**: List block devices.

  ```
  lsblk -f    # Show filesystems and partitions
  ```

- **blkid**: Display block device attributes.

  ```
  sudo blkid  # Show UUIDs for devices
  ```

- **ncdu**: Disk usage analyzer with a TUI.

  ```
  ncdu /       # Analyze root directory space usage
  ```

22. File Compression and Decompression

- **gzip**: Compress files.

  ```
  gzip largefile.txt    # Compress file with .gz extension
  ```

- **gunzip**: Decompress .gz files.

  gunzip largefile.txt.gz # Decompress file

- **bzip2**: Compress files with higher compression than gzip.

  bzip2 largefile.txt # Compress file with .bz2 extension

23. Environment Variables and Shell Management

- **env**: Display all environment variables.

  env

- **set**: Set or display shell options and variables.

  set | grep PATH # Show the PATH variable

- **unset**: Remove an environment variable.

  unset VAR_NAME # Remove a specific environment variable

24. Networking Utilities

- **arp**: Show or modify the IP-to-MAC address mappings.

  arp -a # Display all IP-MAC mappings

- **nc (netcat)**: Network tool for debugging and investigation.

  nc -zv example.com 80 # Test if a specific port is open

- **nmap**: Network scanner to discover hosts and services.

  nmap -sP 192.168.1.0/24 # Scan all hosts on a subnet

25. System Security and Permissions

- **umask**: Set default permissions for new files.

  umask 022 # Set default permissions to 755 for new files

- **chmod**: Change file or directory permissions.

  chmod 700 file.txt # Owner only read, write, execute

- **chattr**: Change file attributes.

  sudo chattr +i file.txt # Make file immutable

- **lsattr**: List file attributes.

  lsattr file.txt # Show attributes for a file

26. Container and Kubernetes Management

- **docker-compose**: Manage multi-container Docker applications.

  docker-compose up -d # Start containers in detached mode

- **minikube**: Run a local Kubernetes cluster.

  minikube start # Start minikube cluster

- **helm**: Kubernetes package manager.

  helm install myapp ./myapp-chart # Install Helm chart for an app

27. Advanced Git Operations

- **git stash**: Temporarily save changes.

```
git stash    # Stash current changes
```

- **git rebase**: Reapply commits on top of another base commit.

  ```
  git rebase main    # Rebase current branch onto main
  ```

- **git log**: View commit history.

  ```
  git log --oneline --graph    # Compact log with graph view
  ```

28. Troubleshooting and Debugging

- **strace**: Trace system calls and signals.

  ```
  strace -p 1234    # Trace process with PID 1234
  ```

- **lsof**: List open files by processes.

  ```
  lsof -i :8080    # List processes using port 8080
  ```

- **dmesg**: Print kernel ring buffer messages.

  ```
  dmesg | tail -10    # View last 10 kernel messages
  ```

29. Data Manipulation and Processing

- **paste**: Merge lines of files.

  ```
  paste file1.txt file2.txt    # Combine lines from two files
  ```

- **join**: Join lines of two files on a common field.

  ```
  join file1.txt file2.txt    # Join files on matching lines
  ```

- **column**: Format text output into columns.

```
cat data.txt | column -t    # Display data in columns
```

30. File Transfer

- **rsync**: Sync files between local and remote systems.

  ```
  rsync -avz /local/dir user@remote:/remote/dir
  ```

- **scp**: Securely copy files between hosts.

  ```
  scp file.txt user@remote:/path/to/destination    # Copy to remote
  ```

- **ftp**: Transfer files using FTP protocol.

  ```
  ftp example.com    # Connect to FTP server example.com
  ```

31. Job Management and Scheduling

- **bg**: Send a job to the background.

  ```
  ./script.sh &   # Run a script in the background
  ```

- **fg**: Bring a background job to the foreground.

  ```
  fg %1          # Bring job 1 to the foreground
  ```

- **at**: Schedule a command to run once at a specified time.

  ```
  echo "echo Hello, DevOps" | at now + 2 minutes    # Run in 2 minutes
  ```

---

2. Shell scripting

1. Automating Server Provisioning (AWS EC2 Launch)

```
#!/bin/bash

# Variables
INSTANCE_TYPE="t2.micro"
AMI_ID="ami-0abcdef1234567890"  # Replace with the correct AMI ID

KEY_NAME="my-key-pair"  # Replace with your key pair name
SECURITY_GROUP="sg-0abc1234def567890"  # Replace with your security group ID
SUBNET_ID="subnet-0abc1234def567890"  # Replace with your subnet ID
REGION="us-west-2"  # Replace with your AWS region

# Launch EC2 instance
aws ec2 run-instances --image-id $AMI_ID --count 1 --instance-type $INSTANCE_TYPE \
--key-name $KEY_NAME --security-group-ids $SECURITY_GROUP --subnet-id $SUBNET_ID --region $REGION

echo "EC2 instance launched successfully!"
```

---

2. System Monitoring (CPU Usage Alert)

```
#!/bin/bash

# Threshold for CPU usage
CPU_THRESHOLD=80

# Get the current CPU usage
CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk '{print 100 - $1}')

# Check if CPU usage exceeds threshold
if (( $(echo "$CPU_USAGE > $CPU_THRESHOLD" | bc -l) )); then
  echo "Alert: CPU usage is above $CPU_THRESHOLD%. Current usage is $CPU_USAGE%" | mail -s "CPU Usage Alert" user@example.com
fi
```

---

3. Backup Automation (MySQL Backup)

```
#!/bin/bash

# Variables
DB_USER="root"
```

```
DB_PASSWORD="password"

DB_NAME="my_database"

BACKUP_DIR="/backup"

DATE=$(date +%F)


# Create backup directory if it doesn't exist

mkdir -p $BACKUP_DIR


# Backup command

mysqldump -u $DB_USER -p$DB_PASSWORD $DB_NAME > $BACKUP_DIR/backup_$DATE.sql


# Optional: Compress the backup

gzip $BACKUP_DIR/backup_$DATE.sql


echo "Backup completed successfully!"
```

---

4. Log Rotation and Cleanup

```
#!/bin/bash


# Variables
```

```
LOG_DIR="/var/log/myapp"

ARCHIVE_DIR="/var/log/myapp/archive"

DAYS_TO_KEEP=30


# Create archive directory if it doesn't exist

mkdir -p $ARCHIVE_DIR


# Find and compress logs older than 7 days

find $LOG_DIR -type f -name "*.log" -mtime +7 -exec gzip {} \; -exec mv {} $ARCHIVE_DIR \;


# Delete logs older than 30 days

find $ARCHIVE_DIR -type f -name "*.log.gz" -mtime +$DAYS_TO_KEEP -exec rm {} \;


echo "Log rotation and cleanup completed!"
```

---

5. CI/CD Pipeline Automation (Trigger Jenkins Job)

```
#!/bin/bash


# Jenkins details
```

```
JENKINS_URL="http://jenkins.example.com"

JOB_NAME="my-pipeline-job"

USER="your-username"

API_TOKEN="your-api-token"


# Trigger Jenkins job

curl -X POST "$JENKINS_URL/job/$JOB_NAME/build" --user
"$USER:$API_TOKEN"


echo "Jenkins job triggered successfully!"
```

---

6. Deployment Automation (Kubernetes Deployment)

```
#!/bin/bash


# Variables

NAMESPACE="default"

DEPLOYMENT_NAME="my-app"

IMAGE="my-app:v1.0"


# Deploy to Kubernetes
```

```
kubectl set image deployment/$DEPLOYMENT_NAME
$DEPLOYMENT_NAME=$IMAGE --namespace=$NAMESPACE

echo "Deployment updated to version $IMAGE!"
```

---

7. Infrastructure as Code (Terraform Apply)

```
#!/bin/bash

# Variables
TF_DIR="/path/to/terraform/config"

# Navigate to Terraform directory
cd $TF_DIR

# Run terraform apply
terraform apply -auto-approve

echo "Terraform apply completed successfully!"
```

---

8. Database Management (PostgreSQL Schema Migration)

bash

```
#!/bin/bash

# Variables
DB_USER="postgres"
DB_PASSWORD="password"
DB_NAME="my_database"
MIGRATION_FILE="/path/to/migration.sql"

# Run schema migration
PGPASSWORD=$DB_PASSWORD psql -U $DB_USER -d $DB_NAME -f $MIGRATION_FILE

echo "Database schema migration completed!"
```

---

9. User Management (Add User to Group)

```
#!/bin/bash

# Variables
```

```
USER_NAME="newuser"
GROUP_NAME="devops"

# Add user to group
usermod -aG $GROUP_NAME $USER_NAME

echo "User $USER_NAME added to group $GROUP_NAME!"
```

---

10. Security Audits (Check for Open Ports)

```
#!/bin/bash

# Check for open ports
OPEN_PORTS=$(netstat -tuln)

# Check if any ports are open (excluding localhost)
if [[ $OPEN_PORTS =~ "0.0.0.0" || $OPEN_PORTS =~ "127.0.0.1" ]]; then
  echo "Security Alert: Open ports detected!"
  echo "$OPEN_PORTS" | mail -s "Open Ports Security Alert" user@example.com
else
```

```
  echo "No open ports detected."
Fi
```

---

11. Performance Tuning

This script clears memory caches and restarts services to free up system resources.

```
#!/bin/bash
# Clear memory caches to free up resources
sync; echo 3 > /proc/sys/vm/drop_caches
# Restart services to free up resources
systemctl restart nginx
systemctl restart apache2
```

---

12. Automated Testing

This script runs automated tests using a testing framework like pytest for Python or JUnit for Java.

```
#!/bin/bash
```

 # Run unit tests using pytest (Python example)

pytest tests/

 # Or, run JUnit tests (Java example)

mvn test

---

13. Scaling Infrastructure

This script automatically scales EC2 instances in an Auto Scaling group based on CPU usage.

#!/bin/bash

 # Check CPU usage and scale EC2 instances

CPU_USAGE=$(aws cloudwatch get-metric-statistics --namespace AWS/EC2 --metric-name CPUUtilization --dimensions Name=InstanceId,Value=i-1234567890abcdef0 --statistics Average --period 300 --start-time $(date -d '5 minutes ago' --utc +%FT%TZ) --end-time $(date --utc +%FT%TZ) --query 'Datapoints[0].Average' --output text)

if (($(echo "$CPU_USAGE > 80" | bc -l))); then

  aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-auto-scaling-group --desired-capacity 3

fi

---

14. Environment Setup

This script sets environment variables for different environments (development, staging, production).

```
#!/bin/bash

# Set environment variables for different stages

if [ "$1" == "production" ]; then

  export DB_HOST="prod-db.example.com"

  export API_KEY="prod-api-key"

elif [ "$1" == "staging" ]; then

  export DB_HOST="staging-db.example.com"

  export API_KEY="staging-api-key"

else

  export DB_HOST="dev-db.example.com"

  export API_KEY="dev-api-key"

fi
```

---

15. Error Handling and Alerts

This script checks logs for errors and sends a Slack notification if an error is found.

```
#!/bin/bash

# Check logs for error messages and send Slack notification

if grep -i "error" /var/log/myapp.log; then

  curl -X POST -H 'Content-type: application/json' --data '{"text":"Error found in logs!"}' https://hooks.slack.com/services/your/webhook/url

fi
```

---

16. Automated Software Installation and Updates

This script installs Docker if it's not already installed on the system.

```
#!/bin/bash


# Install Docker

if ! command -v docker &> /dev/null; then

  curl -fsSL https://get.docker.com -o get-docker.sh

  sudo sh get-docker.sh

fi
```

---

17. Configuration Management

This script updates configuration files (like nginx.conf) across multiple servers.

```
#!/bin/bash

# Update nginx configuration across all servers
scp nginx.conf user@server:/etc/nginx/nginx.conf
ssh user@server "systemctl restart nginx"
```

---

18. Health Check Automation

This script checks the health of multiple web servers by making HTTP requests.

```
#!/bin/bash
# Check if web servers are running
for server in "server1" "server2" "server3"; do
  curl -s --head http://$server | head -n 1 | grep "HTTP/1.1 200 OK" > /dev/null
  if [ $? -ne 0 ]; then
    echo "$server is down"
  else
    echo "$server is up"
  fi
done
```

---

19. Automated Cleanup of Temporary Files

This script removes files older than 30 days from the /tmp directory to free up disk space.

```
#!/bin/bash

# Remove files older than 30 days in /tmp
find /tmp -type f -mtime +30 -exec rm -f {} \;
```

---

20. Environment Variable Management

This script sets environment variables from a .env file.

```
#!/bin/bash

# Set environment variables from a .env file
export $(grep -v '^#' .env | xargs)
```

---

21. Server Reboot Automation

This script automatically reboots the server during off-hours (between 2 AM and 4 AM).

```
#!/bin/bash
# Reboot server during off-hours
if [ $(date +%H) -ge 2 ] && [ $(date +%H) -lt 4 ]; then
  sudo reboot
fi
```

---

22. SSL Certificate Renewal

This script renews SSL certificates using certbot and reloads the web server.

```
#!/bin/bash
# Renew SSL certificates using certbot
certbot renew
systemctl reload nginx
```

---

23. Automatic Scaling of Containers

This script checks the CPU usage of a Docker container and scales it based on usage.

```
#!/bin/bash
```

 # Check CPU usage of a Docker container and scale if necessary

```
CPU_USAGE=$(docker stats --no-stream --format "{{.CPUPerc}}" my-container | sed 's/%//')

if (( $(echo "$CPU_USAGE > 80" | bc -l) )); then

  docker-compose scale my-container=3

fi
```

---

24. Backup Verification

This script verifies the integrity of backup files and reports any corrupted ones.

```
#!/bin/bash


# Verify backup files integrity

for backup in /backups/*.tar.gz; do

  if ! tar -tzf $backup > /dev/null 2>&1; then

    echo "Backup $backup is corrupted"

  else

    echo "Backup $backup is valid"

  fi

done
```

---

25. Automated Server Cleanup

This script removes unused Docker images, containers, and volumes to save disk space.

#!/bin/bash

 # Remove unused Docker images, containers, and volumes

docker system prune -af

---

26. Version Control Operations

This script pulls the latest changes from a Git repository and creates a release tag.

#!/bin/bash

 # Pull latest changes from Git repository and create a release tag

git pull origin main

git tag -a v$(date +%Y%m%d%H%M%S) -m "Release $(date)"

git push origin --tags

---

27. Application Deployment Rollback

This script reverts to the previous Docker container image if a deployment fails.

```
#!/bin/bash

# Rollback to the previous Docker container image if deployment fails
if [ $? -ne 0 ]; then
  docker-compose down
  docker-compose pull my-app:previous
  docker-compose up -d
fi
```

---

28. Automated Log Collection

This script collects logs from multiple servers and uploads them to an S3 bucket.

```
#!/bin/bash

# Collect logs and upload them to an S3 bucket
tar -czf /tmp/logs.tar.gz /var/log/*
aws s3 cp /tmp/logs.tar.gz s3://my-log-bucket/logs/$(date +%Y%m%d%H%M%S).tar.gz
```

---

29. Security Patch Management

This script checks for available security patches and applies them automatically.

```
#!/bin/bash
# Check and apply security patches
sudo apt-get update
sudo apt-get upgrade -y --only-upgrade
```

---

30. Custom Monitoring Scripts

This script checks if a database service is running and restarts it if necessary.

```
#!/bin/bash
# Check if a database service is running and restart it if necessary
if ! systemctl is-active --quiet mysql; then
  systemctl restart mysql
  echo "MySQL service was down and has been restarted"
else
  echo "MySQL service is running"
fi
```

---

31. DNS Configuration Automation (Route 53)

```
#!/bin/bash


# Variables

ZONE_ID="your-hosted-zone-id"

DOMAIN_NAME="your-domain.com"

NEW_IP="your-new-ip-address"


# Update Route 53 DNS record

aws route53 change-resource-record-sets --hosted-zone-id $ZONE_ID
--change-batch '{

  "Changes": [

    {

      "Action": "UPSERT",

      "ResourceRecordSet": {

        "Name": "'$DOMAIN_NAME'",

        "Type": "A",

        "TTL": 60,

        "ResourceRecords": [

          {

            "Value": "'$NEW_IP'"

          }
```

```
      ]
    }
  }
 ]
}'
```

---

32. Automated Code Linting and Formatting (ESLint and Prettier)

```
#!/bin/bash

# Run ESLint
npx eslint . --fix

# Run Prettier
npx prettier --write "**/*.js"
```

---

33. Automated API Testing (Using curl)

```
#!/bin/bash

# API URL
```

```
API_URL="https://your-api-endpoint.com/endpoint"

# Make GET request and check for 200 OK response
RESPONSE=$(curl --write-out "%{http_code}" --silent --output /dev/null $API_URL)

if [ $RESPONSE -eq 200 ]; then
  echo "API is up and running"
else
  echo "API is down. Response code: $RESPONSE"
fi
```

---

34. Container Image Scanning (Using Trivy)

```
#!/bin/bash
# Image to scan
IMAGE_NAME="your-docker-image:latest"

# Run Trivy scan
trivy image --exit-code 1 --severity HIGH,CRITICAL $IMAGE_NAME
if [ $? -eq 1 ]; then
  echo "Vulnerabilities found in image: $IMAGE_NAME"
```

```
  exit 1
else
  echo "No vulnerabilities found in image: $IMAGE_NAME"
fi
```

---

35. Disk Usage Monitoring and Alerts (Email Notification)

```
#!/bin/bash
# Disk usage threshold
THRESHOLD=80

# Get current disk usage percentage
DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')

# Check if disk usage exceeds threshold
if [ $DISK_USAGE -gt $THRESHOLD ]; then
  echo "Disk usage is above threshold: $DISK_USAGE%" | mail -s "Disk Usage Alert" your-email@example.com
fi
```

---

36. Automated Load Testing (Using Apache Benchmark)

```
#!/bin/bash

# Target URL

URL="https://your-application-url.com"


# Run Apache Benchmark with 1000 requests and 10 concurrent requests

ab -n 1000 -c 10 $URL
```

---

37. Automated Email Reports (Server Health Report)

```
#!/bin/bash


# Server Health Report

REPORT=$(top -n 1 | head -n 10)


# Send report via email

echo "$REPORT" | mail -s "Server Health Report" your-email@example.com
```

---

38. DNS Configuration Automation (Route 53)

Introduction: This script automates the process of updating DNS records in AWS Route 53 when the IP address of a server changes. It ensures that DNS records are updated dynamically when new servers are provisioned.

```
#!/bin/bash

# Variables

ZONE_ID="your-hosted-zone-id"

DOMAIN_NAME="your-domain.com"

NEW_IP="your-new-ip-address"


# Update Route 53 DNS record

aws route53 change-resource-record-sets --hosted-zone-id $ZONE_ID
--change-batch '{

  "Changes": [

    {

      "Action": "UPSERT",

      "ResourceRecordSet": {

        "Name": "'$DOMAIN_NAME'",

        "Type": "A",

        "TTL": 60,

        "ResourceRecords": [

          {
```

```
          "Value": "'$NEW_IP'"
        }
      ]
    }
  }
 ]
}'
```

---

39. Automated Code Linting and Formatting (ESLint and Prettier)

Introduction: This script runs ESLint and Prettier to check and automatically format JavaScript code before deployment. It ensures code quality and consistency.

```
#!/bin/bash

# Run ESLint
npx eslint . --fix

# Run Prettier
npx prettier --write "**/*.js"
```

---

40. Automated API Testing (Using curl)

Introduction: This script automates the process of testing an API by sending HTTP requests and verifying the response status. It helps ensure that the API is functioning correctly.

```
#!/bin/bash


**# API URL**

API_URL="https://your-api-endpoint.com/endpoint"


# Make GET request and check for 200 OK response

RESPONSE=$(curl --write-out "%{http_code}" --silent --output /dev/null $API_URL)


if [ $RESPONSE -eq 200 ]; then

  echo "API is up and running"

else

  echo "API is down. Response code: $RESPONSE"

fi
```

---

41. Container Image Scanning (Using Trivy)

Introduction: This script scans Docker images for known vulnerabilities using Trivy. It ensures that only secure images are deployed in production.

```
#!/bin/bash


# Image to scan

IMAGE_NAME="your-docker-image:latest"


# Run Trivy scan

trivy image --exit-code 1 --severity HIGH,CRITICAL $IMAGE_NAME

if [ $? -eq 1 ]; then

  echo "Vulnerabilities found in image: $IMAGE_NAME"

  exit 1

else

  echo "No vulnerabilities found in image: $IMAGE_NAME"

fi
```

---

42. Disk Usage Monitoring and Alerts (Email Notification)

Introduction: This script monitors disk usage and sends an alert via email if the disk usage exceeds a specified threshold. It helps in proactive monitoring of disk space.

```
#!/bin/bash
```

 # Disk usage threshold

THRESHOLD=80

 # Get current disk usage percentage

DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')

 # Check if disk usage exceeds threshold

if [$DISK_USAGE -gt $THRESHOLD]; then

  echo "Disk usage is above threshold: $DISK_USAGE%" | mail -s "Disk Usage Alert" your-email@example.com

fi

---

43. Automated Load Testing (Using Apache Benchmark)

Introduction: This script runs load tests using Apache Benchmark (ab) to simulate traffic on an application. It helps measure the performance and scalability of the application.

bash

#!/bin/bash

 # Target URL

URL="https://your-application-url.com"

 # Run Apache Benchmark with 1000 requests and 10 concurrent requests

ab -n 1000 -c 10 $URL

---

44. Automated Email Reports (Server Health Report)

Introduction: This script generates a server health report using system commands like top and sends it via email. It helps keep track of server performance and health.

#!/bin/bash

 # Server Health Report

REPORT=$(top -n 1 | head -n 10)

 # Send report via email

echo "$REPORT" | mail -s "Server Health Report" your-email@example.com

---

45. Automating Documentation Generation (Using pdoc for Python)

Introduction: This script generates HTML documentation from Python code using pdoc. It helps automate the process of creating up-to-date documentation from the source code.

#!/bin/bash

 # Generate documentation using pdoc

pdoc --html your-python-module --output-dir docs/

 # Optionally, you can zip the generated docs

zip -r docs.zip docs/

List all cron jobs

crontab -l

Edit cron jobs

crontab -e

Remove all cron jobs

crontab -r

Use a specific editor (e.g., nano)

```
EDITOR=nano crontab -e


# Cron Job Syntax
# * * * * * command_to_execute
# ┬ ┬ ┬ ┬ ┬
# │ │ │ │ │
# │ │ │ │ └──── Day of the week (0-6, Sunday=0)
# │ │ │ └────── Month (1-12 or JAN-DEC)
# │ │ └──────── Day of the month (1-31)
# │ └────────── Hour (0-23)
# └──────────── Minute (0-59)

# Run a script every minute
* * * * * /path/to/script.sh

# Run a script every 5 minutes
*/5 * * * * /path/to/script.sh

# Run a script every 10 minutes
*/10 * * * * /path/to/script.sh
```

```
# Run a script at midnight
0 0 * * * /path/to/script.sh

# Run a script every hour
0 * * * * /path/to/script.sh

# Run a script every 2 hours
0 */2 * * * /path/to/script.sh

# Run a script every Sunday at 3 AM
0 3 * * 0 /path/to/script.sh

# Run a script at 9 AM on the 1st of every month
0 9 1 * * /path/to/script.sh

# Run a script every Monday to Friday at 6 PM
0 18 * * 1-5 /path/to/script.sh

# Run a script on the first Monday of every month
0 9 * * 1 [ "$(date +\%d)" -le 7 ] && /path/to/script.sh
```

```
# Run a script on specific dates (e.g., 1st and 15th of the month)
0 12 1,15 * * /path/to/script.sh

# Run a script between 9 AM and 5 PM, every hour
0 9-17 * * * /path/to/script.sh

# Run a script every reboot
@reboot /path/to/script.sh

# Run a script daily at midnight
@daily /path/to/script.sh

# Run a script weekly at midnight on Sunday
@weekly /path/to/script.sh

# Run a script monthly at midnight on the 1st
@monthly /path/to/script.sh

# Run a script yearly at midnight on January 1st
@yearly /path/to/script.sh
```

Redirect cron job output to a log file

0 0 * * * /path/to/script.sh >> /var/log/script.log 2>&1

Run a job only if the previous instance is not running

0 * * * * flock -n /tmp/job.lock /path/to/script.sh

Run a script with a random delay (0-59 minutes)

RANDOM_DELAY=$((RANDOM % 60)) && sleep $RANDOM_DELAY && /path/to/script.sh

Run a script with environment variables

SHELL=/bin/bash

PATH=/usr/local/bin:/usr/bin:/bin

0 5 * * * /path/to/script.sh

Check cron logs (Ubuntu/Debian)

grep CRON /var/log/syslog

Check cron logs (Red Hat/CentOS)

grep CRON /var/log/cron

Restart cron service (Linux)

sudo systemctl restart cron

Check if cron service is running

sudo systemctl status cron

Python

Python Basics

- Run a script: python script.py
- Start interactive mode: python
- Check Python version: python --version
- Install a package: pip install package_name

Create a virtual environment:

```
python -m venv venv
source venv/bin/activate  # On Linux/macOS
venv\Scripts\activate  # On Windows
```

- Deactivate virtual environment: deactivate

1. File Operations

- **Read a file:**

  python

  ```
  with open('file.txt', 'r') as file:
      content = file.read()
      print(content)
  ```

- **Write to a file:**

  python

  ```
  with open('output.txt', 'w') as file:
      file.write('Hello, DevOps!')
  ```

2. Environment Variables

- **Get an environment variable:**

  python

  ```
  import os

  db_user = os.getenv('DB_USER')
  print(db_user)
  ```

- **Set an environment variable:**

  python

  ```
  import os

  os.environ['NEW_VAR'] = 'value'
  ```

3. Subprocess Management

- **Run shell commands:**

  python

  ```
  import subprocess

  result = subprocess.run(['ls', '-l'], capture_output=True, text=True)
  print(result.stdout)
  ```

4. API Requests

- **Make a GET request:**

  python

  ```
  import requests

  response = requests.get('https://api.example.com/data')
  print(response.json())
  ```

5. JSON Handling

- **Read JSON from a file:**

  python

  ```
  import json

  with open('data.json', 'r') as file:
      data = json.load(file)
      print(data)
  ```

- **Write JSON to a file:**

  python

  ```
  import json

  data = {'name': 'DevOps', 'type': 'Workflow'}
  with open('output.json', 'w') as file:
      json.dump(data, file, indent=4)
  ```

6. Logging

- **Basic logging setup:**

  python

```
import logging

logging.basicConfig(level=logging.INFO)
logging.info('This is an informational message')
```

7. Working with Databases

- **Connect to a SQLite database:**

  ```
  python

  import sqlite3

  conn = sqlite3.connect('example.db')
  cursor = conn.cursor()
  cursor.execute('CREATE TABLE IF NOT EXISTS users (id INTEGER
  PRIMARY KEY, name TEXT)')
  conn.commit()
  conn.close()
  ```

8. Automation with Libraries

- **Using Paramiko for SSH connections:**

  ```
  python

  import paramiko

  ssh = paramiko.SSHClient()
  ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
  ssh.connect('hostname', username='user', password='password')

  stdin, stdout, stderr = ssh.exec_command('ls')
  print(stdout.read().decode())
  ssh.close()
  ```

9. Error Handling

- **Try-except block:**

  python

  ```
  try:
      # code that may raise an exception
      risky_code()
  except Exception as e:
      print(f'Error occurred: {e}')
  ```

10. Docker Integration

- **Using the docker package to interact with Docker:**

  python

  ```
  import docker

  client = docker.from_env()
  containers = client.containers.list()
  for container in containers:
      print(container.name)
  ```

11. Working with YAML Files

- **Read a YAML file:**

  python

  ```
  import yaml

  with open('config.yaml', 'r') as file:
      config = yaml.safe_load(file)
      print(config)
  ```

- **Write to a YAML file:**

  python

```
import yaml

data = {'name': 'DevOps', 'version': '1.0'}
with open('output.yaml', 'w') as file:
    yaml.dump(data, file)
```

12. Parsing Command-Line Arguments

- **Using argparse:**

  ```
  python

  import argparse

  parser = argparse.ArgumentParser(description='Process some integers.')
  parser.add_argument('--num', type=int, help='an integer for the accumulator')

  args = parser.parse_args()
  print(args.num)
  ```

13. Monitoring System Resources

- **Using psutil to monitor system resources:**

  ```
  python

  import psutil

  print(f"CPU Usage: {psutil.cpu_percent()}%")
  print(f"Memory Usage: {psutil.virtual_memory().percent}%")
  ```

14. Handling HTTP Requests with Flask

- **Basic Flask API:**

  ```
  python
  ```

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/health', methods=['GET'])
def health_check():
    return jsonify({'status': 'healthy'})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

15. Creating Docker Containers

- **Using the Docker SDK to create a container:**

  ```
  python

  import docker

  client = docker.from_env()
  container = client.containers.run('ubuntu', 'echo Hello World', detach=True)
  print(container.logs())
  ```

16. Scheduling Tasks

- **Using schedule for task scheduling:**

  ```
  python

  import schedule
  import time

  def job():
      print("Running scheduled job...")

  schedule.every(1).minutes.do(job)

  while True:
      schedule.run_pending()
      time.sleep(1)
  ```

17. Version Control with Git

- **Using GitPython to interact with Git repositories:**

  python

  ```
  import git

  repo = git.Repo('/path/to/repo')
  repo.git.add('file.txt')
  repo.index.commit('Added file.txt')
  ```

18. Email Notifications

- **Sending emails using smtplib:**

  python

  ```
  import smtplib
  from email.mime.text import MIMEText

  msg = MIMEText('This is the body of the email')
  msg['Subject'] = 'Email Subject'
  msg['From'] = 'you@example.com'
  msg['To'] = 'recipient@example.com'

  with smtplib.SMTP('smtp.example.com', 587) as server:
      server.starttls()
      server.login('your_username', 'your_password')
      server.send_message(msg)
  ```

19. Creating Virtual Environments

- **Creating and activating a virtual environment:**

  python

  ```
  import os
  ```

```
import subprocess

# Create virtual environment
subprocess.run(['python3', '-m', 'venv', 'myenv'])

# Activate virtual environment (Windows)
os.system('myenv\\Scripts\\activate')

# Activate virtual environment (Linux/Mac)
os.system('source myenv/bin/activate')
```

20. Integrating with CI/CD Tools

- **Using the requests library to trigger a Jenkins job:**

  ```
  python

  import requests

  url = 'http://your-jenkins-url/job/your-job-name/build'
  response = requests.post(url, auth=('user', 'token'))
  print(response.status_code)
  ```

21. Database Migration

- **Using Alembic for database migrations:**

  ```
  bash

  alembic revision -m "initial migration"
  alembic upgrade head
  ```

22. Testing Code

- **Using unittest for unit testing:**

  ```
  python
  ```

```
import unittest

def add(a, b):
    return a + b

class TestMathFunctions(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2, 3), 5)

if __name__ == '__main__':
    unittest.main()
```

23. Data Transformation with Pandas

- **Using pandas for data manipulation:**

  python

  ```
  import pandas as pd

  df = pd.read_csv('data.csv')
  df['new_column'] = df['existing_column'] * 2
  df.to_csv('output.csv', index=False)
  ```

24. Using Python for Infrastructure as Code

- **Using boto3 for AWS operations:**

  python

  ```
  import boto3

  ec2 = boto3.resource('ec2')
  instances = ec2.instances.filter(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])
  for instance in instances:
      print(instance.id, instance.state)
  ```

25. Web Scraping

- **Using BeautifulSoup to scrape web pages:**

  python

  ```
  import requests
  from bs4 import BeautifulSoup

  response = requests.get('http://example.com')
  soup = BeautifulSoup(response.content, 'html.parser')
  print(soup.title.string)
  ```

26. Using Fabric for Remote Execution

- **Running commands on a remote server:**

  python

  ```
  from fabric import Connection

  conn = Connection(host='user@hostname', connect_kwargs={'password':
  'your_password'})
  conn.run('uname -s')
  ```

27. Automating AWS S3 Operations

- **Upload and download files using boto3:**

  python

  ```
  import boto3

  s3 = boto3.client('s3')
  ```

```
# Upload a file
s3.upload_file('local_file.txt', 'bucket_name', 's3_file.txt')

# Download a file
s3.download_file('bucket_name', 's3_file.txt', 'local_file.txt')
```

28. Monitoring Application Logs

- **Tail logs using tail -f equivalent in Python:**

```
python

import time

def tail_f(file):
    file.seek(0, 2)  # Move to the end of the file
    while True:
        line = file.readline()
        if not line:
            time.sleep(0.1)  # Sleep briefly
            continue
        print(line)

with open('app.log', 'r') as log_file:
    tail_f(log_file)
```

29. Container Health Checks

- **Check the health of a running Docker container:**

```
python

import docker

client = docker.from_env()
```

```
container = client.containers.get('container_id')
print(container.attrs['State']['Health']['Status'])
```

30. Using requests for Rate-Limited APIs

- **Handle rate limiting in API requests:**

  python

  ```
  import requests
  import time

  url = 'https://api.example.com/data'
  while True:
      response = requests.get(url)
      if response.status_code == 200:
          print(response.json())
          break
      elif response.status_code == 429:  # Too Many Requests
          time.sleep(60)  # Wait a minute before retrying
      else:
          print('Error:', response.status_code)
          break
  ```

31. Docker Compose Integration

- **Using docker-compose in Python:**

  python

  ```
  import os
  ```

```
import subprocess

# Start services defined in docker-compose.yml
subprocess.run(['docker-compose', 'up', '-d'])

# Stop services
subprocess.run(['docker-compose', 'down'])
```

32. Terraform Execution

- **Executing Terraform commands with subprocess:**

  python

  ```
  import subprocess

  # Initialize Terraform
  subprocess.run(['terraform', 'init'])

  # Apply configuration
  subprocess.run(['terraform', 'apply', '-auto-approve'])
  ```

33. Working with Prometheus Metrics

- **Scraping and parsing Prometheus metrics:**

  python

  ```
  import requests

  response = requests.get('http://localhost:9090/metrics')
  metrics = response.text.splitlines()

  for metric in metrics:
      print(metric)
  ```

34. Using pytest for Testing

- **Simple test case with pytest:**

  python

  ```
  def add(a, b):
      return a + b

  def test_add():
      assert add(2, 3) == 5
  ```

35. Creating Webhooks

- **Using Flask to create a simple webhook:**

  python

  ```
  from flask import Flask, request

  app = Flask(__name__)

  @app.route('/webhook', methods=['POST'])
  def webhook():
      data = request.json
      print('Received data:', data)
      return 'OK', 200

  if __name__ == '__main__':
      app.run(port=5000)
  ```

36. Using Jinja2 for Configuration Templates

- **Render configuration files with Jinja2:**

  python

```
from jinja2 import Template

template = Template('Hello, {{ name }}!')
rendered = template.render(name='DevOps')
print(rendered)
```

37. Encrypting and Decrypting Data

- **Using cryptography to encrypt and decrypt:**

  python

  ```
  from cryptography.fernet import Fernet

  # Generate a key
  key = Fernet.generate_key()
  cipher_suite = Fernet(key)

  # Encrypt
  encrypted_text = cipher_suite.encrypt(b'Secret Data')

  # Decrypt
  decrypted_text = cipher_suite.decrypt(encrypted_text)
  print(decrypted_text.decode())
  ```

38. Error Monitoring with Sentry

- **Sending error reports to Sentry:**

  python

```
import sentry_sdk

sentry_sdk.init('your_sentry_dsn')

def divide(a, b):
    return a / b

try:
    divide(1, 0)
except ZeroDivisionError as e:
    sentry_sdk.capture_exception(e)
```

39. Setting Up Continuous Integration with GitHub Actions

- **Sample workflow file (.github/workflows/ci.yml):**

  yaml

```
name: CI

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.8'
      - name: Install dependencies
        run: |
          pip install -r requirements.txt
      - name: Run tests
        run: |
          pytest
```

40. Creating a Simple API with FastAPI

- **Using FastAPI for high-performance APIs:**

  python

  ```
  from fastapi import FastAPI

  app = FastAPI()

  @app.get('/items/{item_id}')
  async def read_item(item_id: int):
      return {'item_id': item_id}

  if __name__ == '__main__':
      import uvicorn
      uvicorn.run(app, host='0.0.0.0', port=8000)
  ```

41. Log Aggregation with ELK Stack

- **Sending logs to Elasticsearch:**

  python

  ```
  from elasticsearch import Elasticsearch

  es = Elasticsearch(['http://localhost:9200'])

  log = {'level': 'info', 'message': 'This is a log message'}
  es.index(index='logs', body=log)
  ```

42. Using pandas for ETL Processes

- **Performing ETL with pandas:**

  python

  ```
  import pandas as pd

  # Extract
  data = pd.read_csv('source.csv')

  # Transform
  data['new_column'] = data['existing_column'].apply(lambda x: x * 2)

  # Load
  data.to_csv('destination.csv', index=False)
  ```

43. Serverless Applications with AWS Lambda

- **Deploying a simple AWS Lambda function:**

  python

  ```
  import json

  def lambda_handler(event, context):
      return {
          'statusCode': 200,
          'body': json.dumps('Hello from Lambda!')
      }
  ```

44. Working with Redis

- **Basic operations with Redis using redis-py:**

  python

  ```
  import redis

  r = redis.StrictRedis(host='localhost', port=6379, db=0)

  # Set a key
  r.set('foo', 'bar')

  # Get a key
  print(r.get('foo'))
  ```

45. Using pyngrok for Tunneling

- **Create a tunnel to expose a local server:**

  python

  ```
  from pyngrok import ngrok

  # Start the tunnel
  public_url = ngrok.connect(5000)
  print('Public URL:', public_url)

  # Keep the tunnel open
  input('Press Enter to exit...')
  ```

46. Creating a REST API with Flask-RESTful

- **Building REST APIs with Flask-RESTful:**

  python

  ```
  from flask import Flask
  from flask_restful import Resource, Api

  app = Flask(__name__)
  api = Api(app)

  class HelloWorld(Resource):
      def get(self):
          return {'hello': 'world'}

  api.add_resource(HelloWorld, '/')

  if __name__ == '__main__':
      app.run(debug=True)
  ```

47. Using asyncio for Asynchronous Tasks

- **Running asynchronous tasks in Python:**

  python

```
import asyncio

async def main():
    print('Hello')
    await asyncio.sleep(1)
    print('World')

asyncio.run(main())
```

48. Network Monitoring with scapy

- **Packet sniffing using scapy:**

  python

  ```
  from scapy.all import sniff

  def packet_callback(packet):
      print(packet.summary())

  sniff(prn=packet_callback, count=10)
  ```

49. Handling Configuration Files with configparser

- **Reading and writing to INI configuration files:**

  python

  ```
  import configparser

  config = configparser.ConfigParser()
  config.read('config.ini')

  print(config['DEFAULT']['SomeSetting'])
  ```

```
config['DEFAULT']['NewSetting'] = 'Value'
with open('config.ini', 'w') as configfile:
    config.write(configfile)
```

50. WebSocket Client Example

- **Creating a WebSocket client with websocket-client:**

  python

  ```
  import websocket

  def on_message(ws, message):
      print("Received message:", message)

  ws = websocket.WebSocketApp("ws://echo.websocket.org",
                              on_message=on_message)
  ws.run_forever()
  ```

51. Creating a Docker Image with Python

- **Using docker library to build an image:**

  python

  ```
  import docker
  ```

```
client = docker.from_env()

# Dockerfile content
dockerfile_content = """
FROM python:3.9-slim
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
"""

# Create a Docker image
image, build_logs = client.images.build(fileobj=dockerfile_content.encode('utf-8'),
tag='my-python-app')

for line in build_logs:
    print(line)
```

52. Using psutil for System Monitoring

- **Retrieve system metrics such as CPU and memory usage:**

  python

  ```
  import psutil

  print("CPU Usage:", psutil.cpu_percent(interval=1), "%")
  print("Memory Usage:", psutil.virtual_memory().percent, "%")
  ```

53. Database Migration with Alembic

- **Script to initialize Alembic migrations:**

  python

  ```
  from alembic import command
  ```

```
from alembic import config

alembic_cfg = config.Config("alembic.ini")
command.upgrade(alembic_cfg, "head")
```

54. Using paramiko for SSH Connections

- **Execute commands on a remote server via SSH:**

  ```
  python

  import paramiko

  client = paramiko.SSHClient()
  client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
  client.connect('hostname', username='user', password='your_password')

  stdin, stdout, stderr = client.exec_command('ls -la')
  print(stdout.read().decode())
  client.close()
  ```

55. CloudFormation Stack Creation with boto3

- **Creating an AWS CloudFormation stack:**

  python

  ```
  import boto3

  cloudformation = boto3.client('cloudformation')

  with open('template.yaml', 'r') as template_file:
      template_body = template_file.read()

  response = cloudformation.create_stack(
      StackName='MyStack',
      TemplateBody=template_body,
      Parameters=[
          {
              'ParameterKey': 'InstanceType',
              'ParameterValue': 't2.micro'
          },
      ],
      TimeoutInMinutes=5,
      Capabilities=['CAPABILITY_NAMED_IAM'],
  )
  print(response)
  ```

56. Automating EC2 Instance Management

- **Starting and stopping EC2 instances:**

  python

  ```
  import boto3

  ec2 = boto3.resource('ec2')

  # Start an instance
  instance = ec2.Instance('instance_id')
  instance.start()

  # Stop an instance
  instance.stop()
  ```

57. Automated Backup with shutil

- **Backup files to a specific directory:**

  python

  ```
  import shutil
  import os

  source_dir = '/path/to/source'
  backup_dir = '/path/to/backup'

  shutil.copytree(source_dir, backup_dir)
  ```

58. Using watchdog for File System Monitoring

- **Monitor changes in a directory:**

  python

  ```
  from watchdog.observers import Observer
  from watchdog.events import FileSystemEventHandler

  class MyHandler(FileSystemEventHandler):
      def on_modified(self, event):
          print(f'File modified: {event.src_path}')

  event_handler = MyHandler()
  observer = Observer()
  observer.schedule(event_handler, path='path/to/monitor', recursive=False)
  observer.start()

  try:
      while True:
          time.sleep(1)
  except KeyboardInterrupt:
      observer.stop()
  observer.join()
  ```

59. Load Testing with locust

- **Basic Locust load testing setup:**

  python

  ```
  from locust import HttpUser, task, between

  class MyUser(HttpUser):
      wait_time = between(1, 3)

      @task
      def load_test(self):
  ```

```
    self.client.get('/')

# To run, save this as locustfile.py and run: locust
```

60. Integrating with GitHub API

- **Fetching repository details using GitHub API:**

  python

  ```
  import requests

  url = 'https://api.github.com/repos/user/repo'
  response = requests.get(url, headers={'Authorization': 'token
  YOUR_GITHUB_TOKEN'})
  repo_info = response.json()
  print(repo_info)
  ```

61. Managing Kubernetes Resources with kubectl

- **Using subprocess to interact with Kubernetes:**

  python

  ```
  import subprocess

  # Get pods
  subprocess.run(['kubectl', 'get', 'pods'])

  # Apply a configuration
  subprocess.run(['kubectl', 'apply', '-f', 'deployment.yaml'])
  ```

62. Using pytest for CI/CD Testing

- **Integrate tests in your CI/CD pipeline:**

  python

  ```
  # test_example.py
  def test_addition():
      assert 1 + 1 == 2

  # Run pytest in your CI/CD pipeline
  subprocess.run(['pytest'])
  ```

63. Creating a Simple CLI Tool with argparse

- **Build a command-line interface:**

  python

  ```
  import argparse

  parser = argparse.ArgumentParser(description='Process some integers.')
  parser.add_argument('integers', metavar='N', type=int, nargs='+', help='an integer to be processed')
  parser.add_argument('--sum', dest='accumulate', action='store_const', const=sum, default=max, help='sum the integers (default: find the max)')

  args = parser.parse_args()
  print(args.accumulate(args.integers))
  ```

64. Using dotenv for Environment Variables

- **Load environment variables from a .env file:**

  python

  ```
  from dotenv import load_dotenv
  import os

  load_dotenv()

  database_url = os.getenv('DATABASE_URL')
  print(database_url)
  ```

65. Implementing Web Scraping with BeautifulSoup

- **Scraping a web page for data:**

  python

  ```
  import requests
  from bs4 import BeautifulSoup

  response = requests.get('http://example.com')
  soup = BeautifulSoup(response.text, 'html.parser')

  for item in soup.find_all('h1'):
      print(item.text)
  ```

66. Using PyYAML for YAML Configuration Files

- **Load and dump YAML files:**

  python

  ```
  import yaml

  # Load YAML file
  with open('config.yaml', 'r') as file:
      config = yaml.safe_load(file)
      print(config)

  # Dump to YAML file
  with open('output.yaml', 'w') as file:
      yaml.dump(config, file)
  ```

67. Creating a Simple Message Queue with RabbitMQ

- **Send and receive messages using pika:**

  python

  ```
  import pika

  # Sending messages
  connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
  channel = connection.channel()
  channel.queue_declare(queue='hello')

  channel.basic_publish(exchange='', routing_key='hello', body='Hello World!')
  connection.close()

  # Receiving messages
  def callback(ch, method, properties, body):
  ```

```
    print("Received:", body)

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')

channel.basic_consume(queue='hello', on_message_callback=callback,
auto_ack=True)
channel.start_consuming()
```

68. Using sentry_sdk for Monitoring

- **Integrate Sentry for error tracking:**

  python

  ```
  import sentry_sdk

  sentry_sdk.init("YOUR_SENTRY_DSN")

  try:
      # Your code that may throw an exception
      1 / 0
  except Exception as e:
      sentry_sdk.capture_exception(e)
  ```

69. Using openpyxl for Excel File Manipulation

- **Read and write Excel files:**

  python

  ```
  from openpyxl import Workbook, load_workbook

  # Create a new workbook
  ```

```
wb = Workbook()
ws = wb.active
ws['A1'] = 'Hello'
wb.save('sample.xlsx')

# Load an existing workbook
wb = load_workbook('sample.xlsx')
ws = wb.active
print(ws['A1'].value)
```

70. Using sqlalchemy for Database Interaction

- **Define a model and perform CRUD operations:**

  python

  ```
  from sqlalchemy import create_engine, Column, Integer, String
  from sqlalchemy.ext.declarative import declarative_base
  from sqlalchemy.orm import sessionmaker

  Base = declarative_base()

  class User(Base):
      __tablename__ = 'users'
      id = Column(Integer, primary_key=True)
      name = Column(String)

  engine = create_engine('sqlite:///example.db')
  Base.metadata.create_all(engine)

  Session = sessionmaker(bind=engine)
  session = Session()

  # Create
  new_user = User(name='Alice')
  session.add(new_user)
  ```

71. Monitoring Docker Containers with docker-py

- **Fetch and print the status of running containers:**

  python

  ```
  import docker

  client = docker.from_env()
  containers = client.containers.list()

  for container in containers:
      print(f'Container Name: {container.name}, Status: {container.status}')
  ```

72. Using flask to Create a Simple API

- **Basic API setup with Flask:**

  python

  ```
  from flask import Flask, jsonify

  app = Flask(__name__)

  @app.route('/api/data', methods=['GET'])
  def get_data():
      return jsonify({"message": "Hello, World!"})

  if __name__ == '__main__':
      app.run(debug=True)
  ```

73. Automating Certificate Renewal with certbot

- **Script to renew Let's Encrypt certificates:**

  python

  import subprocess

  # Renew certificates
  subprocess.run(['certbot', 'renew'])

74. Using numpy for Data Analysis

- **Performing basic numerical operations:**

  python

  import numpy as np

  data = np.array([1, 2, 3, 4, 5])
  mean_value = np.mean(data)
  print("Mean Value:", mean_value)

75. Creating and Sending Emails with smtplib

- **Send an email using Python:**

  python

  import smtplib
  from email.mime.text import MIMEText

  sender = 'you@example.com'
  recipient = 'recipient@example.com'
  msg = MIMEText('This is a test email.')
  msg['Subject'] = 'Test Email'
  msg['From'] = sender

```
msg['To'] = recipient

with smtplib.SMTP('smtp.example.com') as server:
    server.login('username', 'password')
    server.send_message(msg)
```

76. Using schedule for Task Scheduling

- **Schedule tasks at regular intervals:**

```
python

import schedule
import time

def job():
    print("Job is running...")

schedule.every(10).minutes.do(job)

while True:
    schedule.run_pending()
    time.sleep(1)
```

77. Using matplotlib for Data Visualization

- **Plotting a simple graph:**

```
python

import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

plt.plot(x, y)
plt.xlabel('X-axis')
```

```
plt.ylabel('Y-axis')
plt.title('Simple Plot')
plt.show()
```

78. Creating a Custom Python Package

- **Structure your project as a package:**

  markdown

  ```
  my_package/
      ├── __init__.py
      ├── module1.py
      └── module2.py
  ```

- **setup.py for packaging:**

  python

  ```
  from setuptools import setup, find_packages

  setup(
      name='my_package',
      version='0.1',
      packages=find_packages(),
      install_requires=[
          'requests',
          'flask'
      ],
  )
  ```

79. Using pytest for Unit Testing

- **Writing a simple unit test:**

  python

  ```
  # test_sample.py
  ```

```
def add(a, b):
    return a + b

def test_add():
    assert add(1, 2) == 3
```

80. Implementing OAuth with requests-oauthlib

- **Authenticate with an API using OAuth:**

  python

  ```
  from requests_oauthlib import OAuth1Session

  oauth = OAuth1Session(client_key='YOUR_CLIENT_KEY',
  client_secret='YOUR_CLIENT_SECRET')
  response = oauth.get('https://api.example.com/user')
  print(response.json())
  ```

81. Using pandas for Data Manipulation

- **Load and manipulate data in a CSV file:**

  python

  ```
  import pandas as pd

  df = pd.read_csv('data.csv')
  print(df.head())

  # Filter data
  filtered_df = df[df['column_name'] > 10]
  print(filtered_df)
  ```

82. Using requests for HTTP Requests

- **Making a GET and POST request:**

  python

  ```
  import requests

  # GET request
  response = requests.get('https://api.example.com/data')
  print(response.json())

  # POST request
  data = {'key': 'value'}
  response = requests.post('https://api.example.com/data', json=data)
  print(response.json())
  ```

83. Creating a Basic Web Server with http.server

- **Simple HTTP server to serve files:**

  python

  ```
  from http.server import SimpleHTTPRequestHandler, HTTPServer

  PORT = 8000
  handler = SimpleHTTPRequestHandler

  with HTTPServer(('', PORT), handler) as httpd:
      print(f'Serving on port {PORT}')
      httpd.serve_forever()
  ```

84. Using Flask for Webhooks

- **Handling incoming webhook requests:**

  python

  ```
  from flask import Flask, request

  app = Flask(__name__)

  @app.route('/webhook', methods=['POST'])
  def webhook():
      data = request.json
      print(data)
      return '', 200

  if __name__ == '__main__':
      app.run(port=5000)
  ```

85. Creating a Bash Script with subprocess

- **Run shell commands from Python:**

  python

  ```
  import subprocess

  subprocess.run(['echo', 'Hello, World!'])
  ```

86. Using docker-compose with Python

- **Programmatically run Docker Compose commands:**

  python

  ```
  import subprocess

  subprocess.run(['docker-compose', 'up', '-d'])
  ```

87. Using moto for Mocking AWS Services in Tests

- **Mocking AWS S3 for unit testing:**

  python

  ```
  import boto3
  from moto import mock_s3

  @mock_s3
  def test_s3_upload():
      s3 = boto3.client('s3', region_name='us-east-1')
      s3.create_bucket(Bucket='my-bucket')
      s3.upload_file('file.txt', 'my-bucket', 'file.txt')
      # Test logic here
  ```

88. Using asyncio for Asynchronous Tasks

- **Run multiple tasks concurrently:**

  python

  ```
  import asyncio

  async def say_hello():
      print("Hello")
      await asyncio.sleep(1)
      print("World")

  async def main():
      await asyncio.gather(say_hello(), say_hello())

  asyncio.run(main())
  ```

89. Using flask-cors for Cross-Origin Resource Sharing

- **Allow CORS in a Flask app:**

  python

  ```
  from flask import Flask
  from flask_cors import CORS

  app = Flask(__name__)
  CORS(app)

  @app.route('/data', methods=['GET'])
  def data():
      return {"message": "Hello from CORS!"}

  if __name__ == '__main__':
      app.run()
  ```

90. Using pytest Fixtures for Setup and Teardown

- **Create a fixture to manage resources:**

  python

  ```
  import pytest

  @pytest.fixture
  def sample_data():
      data = {"key": "value"}
      yield data  # This is the test data
      # Teardown code here (if necessary)

  def test_sample_data(sample_data):
      assert sample_data['key'] == 'value'
  ```

91. Using http.client for Low-Level HTTP Requests

- **Make a raw HTTP GET request:**

  python

  ```
  import http.client

  conn = http.client.HTTPSConnection("www.example.com")
  conn.request("GET", "/")
  response = conn.getresponse()
  print(response.status, response.reason)
  data = response.read()
  conn.close()
  ```

92. Implementing Redis Caching with redis-py

- **Basic operations with Redis:**

  python

  ```
  import redis

  r = redis.StrictRedis(host='localhost', port=6379, db=0)

  # Set and get value
  r.set('key', 'value')
  print(r.get('key').decode('utf-8'))
  ```

93. Using json for Data Serialization

- **Convert Python objects to JSON:**

```
python

import json

data = {"key": "value"}
json_data = json.dumps(data)
print(json_data)
```

94. Using xml.etree.ElementTree for XML Processing

- **Parse an XML file:**

```
python

import xml.etree.ElementTree as ET

tree = ET.parse('data.xml')
root = tree.getroot()

for child in root:
    print(child.tag, child.attrib)
```

95. Creating a Virtual Environment with venv

- **Programmatically create a virtual environment:**

```
python

import venv

venv.create('myenv', with_pip=True)
```

96. Using psutil for System Monitoring

- **Get system memory usage:**

  python

  import psutil

  memory = psutil.virtual_memory()
  print(f'Total Memory: {memory.total}, Available Memory: {memory.available}')

97. Using sqlite3 for Lightweight Database Management

- **Basic SQLite operations:**

  python

  import sqlite3

  conn = sqlite3.connect('example.db')
  c = conn.cursor()

  c.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)''')
  c.execute("INSERT INTO users (name) VALUES ('Alice')")
  conn.commit()

  for row in c.execute('SELECT * FROM users'):
      print(row)

  conn.close()

98. Using pytest to Run Tests in Parallel

- **Run tests concurrently:**

  bash

```
pytest -n 4  # Run tests in parallel with 4 workers
```

99. Using argparse for Command-Line Arguments

- **Parse command-line arguments:**

  ```
  python

  import argparse

  parser = argparse.ArgumentParser(description='Process some integers.')
  parser.add_argument('integers', metavar='N', type=int, nargs='+',
                      help='an integer for the accumulator')
  parser.add_argument('--sum', dest='accumulate', action='store_const',
                      const=sum, default=max,
                      help='sum the integers (default: find the max)')

  args = parser.parse_args()
  print(args.accumulate(args.integers))
  ```

100. Using jsonschema for JSON Validation

- **Validate JSON against a schema:**

  ```
  python

  from jsonschema import validate
  from jsonschema.exceptions import ValidationError
  ```

```
schema = {
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "age": {"type": "integer", "minimum": 0}
    },
    "required": ["name", "age"]
}

data = {"name": "John", "age": 30}

try:
    validate(instance=data, schema=schema)
    print("Data is valid")
except ValidationError as e:
    print(f"Data is invalid: {e.message}")
```

---

2. Version Control

- **Git**

1. Git Setup and Configuration

```
git --version                    # Check Git version
git config --global user.name "Your Name"  # Set global username
git config --global user.email "your.email@example.com"  # Set global email
git config --global core.editor "vim"  # Set default editor
git config --global init.defaultBranch main  # Set default branch name
git config --list                # View Git configuration
git help <command>               # Get help for a Git command
```

---

2. Creating and Cloning Repositories

```
git init                          # Initialize a new Git repository
git clone <repo_url>              # Clone an existing repository
git remote add origin <repo_url>  # Link local repo to a remote repo
git remote -v                     # List remote repositories
```

---

3. Staging and Committing Changes

```
git status                        # Check the status of changes
git add <file>                    # Add a file to the staging area
git add .                         # Add all files to the staging area
git commit -m "Commit message"    # Commit staged files
git commit -am "Commit message"   # Add & commit changes in one step
git commit --amend -m "New message"   # Modify the last commit message
```

---

4. Viewing History and Logs

```
git log                           # Show commit history
git log --oneline                 # Show history in one-line format
git log --graph --decorate --all  # Display commit history as a graph
git show <commit_id>              # Show details of a specific commit
git diff                          # Show unstaged changes
git diff --staged                 # Show staged but uncommitted changes
git blame <file>                  # Show who modified each line of a file
```

---

5. Branching and Merging

```
git branch                        # List all branches
git branch <branch_name>          # Create a new branch
```

```
git checkout <branch_name>          # Switch to another branch
git checkout -b <branch_name>       # Create and switch to a new branch
git merge <branch_name>             # Merge a branch into the current branch
git branch -d <branch_name>         # Delete a local branch
git branch -D <branch_name>         # Force delete a branch
```

---

6. Working with Remote Repositories

```
git fetch                       # Fetch changes from remote repo
git pull origin <branch_name>       # Pull latest changes
git push origin <branch_name>       # Push changes to remote repo
git push -u origin <branch_name>    # Push and set upstream tracking
git remote show origin              # Show details of the remote repo
git remote rm origin                # Remove the remote repository
```

---

7. Undoing Changes

```
git restore <file>                  # Unstage changes
git restore --staged <file>         # Unstage a file from the staging area
git reset HEAD~1                    # Undo the last commit but keep changes
git reset --hard HEAD~1             # Undo the last commit and discard changes
git revert <commit_id>              # Create a new commit to undo changes
git stash                       # Save uncommitted changes temporarily
git stash pop                   # Apply stashed changes
git stash drop                  # Remove stashed changes
```

---

8. Tagging and Releases

```
git tag                          # List all tags
git tag <tag_name>               # Create a tag
git tag -a <tag_name> -m "Tag message" # Create an annotated tag
git push origin <tag_name>       # Push a tag to remote
git tag -d <tag_name>            # Delete a local tag
git push --delete origin <tag_name>   # Delete a remote tag
```

---

9. Working with Submodules

```
git submodule add <repo_url> <path>   # Add a submodule
git submodule update --init --recursive  # Initialize and update submodules
git submodule foreach git pull origin main  # Update all submodules
```

---

10. Git Aliases (Shortcuts)

```
git config --global alias.st status   # Create alias for status command
git config --global alias.co checkout # Create alias for checkout command
git config --global alias.br branch   # Create alias for branch command
git config --global alias.cm commit   # Create alias for commit command
git config --list | grep alias        # View all configured aliases
```

---

11. Deleting Files and Folders

```
git rm <file>                    # Remove a file and stage deletion
git rm -r <folder>               # Remove a directory
git rm --cached <file>           # Remove file from repo but keep locally
```

---

12. Force Push and Rollback (Use with Caution)

```
git push --force                    # Force push changes
git reset --hard <commit_id>        # Reset repo to a specific commit
git reflog                          # Show history of HEAD changes
git reset --hard ORIG_HEAD          # Undo the last reset
```

---

2. GitHub

Authentication & Configuration

gh auth logout – Log out of GitHub CLI

gh auth status – Check authentication status

gh auth refresh – Refresh authentication token

gh config set editor <editor> – Set the default editor (e.g., nano, vim)

gh config get editor – Get the current editor setting

Repository Management

gh repo list – List repositories for the authenticated user

gh repo delete <name> – Delete a repository

gh repo rename <new-name> – Rename a repository

gh repo fork --clone=false <url> – Fork a repository without cloning

Branch & Commit Management

gh branch list – List branches in a repository

gh branch delete <branch> – Delete a branch

gh browse – Open the repository in a browser

gh co <branch> – Check out a branch

Issue & Pull Request Handling

gh issue create – Create a new issue

gh issue close <issue-number> – Close an issue

gh issue reopen <issue-number> – Reopen a closed issue

gh issue comment <issue-number> --body "Comment" – Add a comment to an issue

gh pr list – List open pull requests

gh pr checkout <pr-number> – Check out a pull request branch

gh pr close <pr-number> – Close a pull request

Gists & Actions

gh gist create <file> – Create a new gist

gh gist list – List all gists

gh workflow list – List GitHub Actions workflows

gh run list – List workflow runs

Webhooks:

- Go to **Repo → Settings → Webhooks → Add Webhook**
- Events: push, pull_request, issues, etc.
- Payload sent in JSON format to the specified URL

---

3. GitLab

Commands

Project & Repository Management

gitlab project create <name> – Create a new project
gitlab repo list – List repositories
gitlab project delete <id> – Delete a project
gitlab repo fork <repo> – Fork a repository
gitlab repo clone <url> – Clone a GitLab repository
gitlab repo archive <repo> – Archive a repository

Issues & Merge Requests

gitlab issue list – List all issues in a project
gitlab issue create --title "<title>" --description "<desc>" – Create an issue
gitlab issue close <issue_id> – Close an issue
gitlab issue reopen <issue_id> – Reopen an issue
gitlab merge_request list – List merge requests
gitlab merge_request create --source-branch <branch> --target-branch <branch> --title "<title>" – Create a merge request
gitlab merge_request close <mr_id> – Close a merge request

Pipeline & CI/CD

gitlab pipeline trigger – Trigger a pipeline
gitlab pipeline list – List all pipelines
gitlab pipeline retry <pipeline_id> – Retry a failed pipeline
gitlab pipeline cancel <pipeline_id> – Cancel a running pipeline
gitlab pipeline delete <pipeline_id> – Delete a pipeline
gitlab runner register – Register a CI/CD runner

gitlab runner list – List registered runners
gitlab runner unregister <runner_id> – Unregister a runner

User & Group Management

gitlab user list – List users in GitLab
gitlab user create --name "<name>" --email "<email>" – Create a new user
gitlab group list – List groups in GitLab
gitlab group create --name "<group_name>" --path "<group_path>" – Create a group
gitlab group delete <group_id> – Delete a group

Access & Permissions

gitlab project member list <project_id> – List project members
gitlab project member add <project_id> <user_id> <access_level> – Add a user to a project
gitlab group member list <group_id> – List group members
gitlab group member add <group_id> <user_id> <access_level> – Add a user to a group

Repository Protection & Settings

gitlab branch protect <branch> – Protect a branch
gitlab branch unprotect <branch> – Unprotect a branch
gitlab repository mirror – Set up repository mirroring
gitlab repository settings update – Update repository settings

Webhooks:

- Go to **Settings → Webhooks**
- Select triggers: Push events, Tag push, Merge request, etc.
- Use GitLab CI/CD with .gitlab-ci.yml

---

4. Bitbucket

Commands

Repository Management

- bitbucket repo create <name> – Create a repository
- bitbucket repo list – List all repositories
- bitbucket repo delete <name> – Delete a repository
- bitbucket repo clone <repo-url> – Clone a repository
- bitbucket repo fork <repo> – Fork a repository
- bitbucket repo update <repo> – Update repository settings

Branch Management

- bitbucket branch create <branch-name> – Create a new branch
- bitbucket branch list – List all branches
- bitbucket branch delete <branch-name> – Delete a branch

Pipeline Management

- bitbucket pipeline run – Run a pipeline
- bitbucket pipeline list – List pipelines
- bitbucket pipeline stop <pipeline-id> – Stop a running pipeline
- bitbucket pipeline rerun <pipeline-id> – Rerun a pipeline

Issue Tracking

- bitbucket issue list – List all issues
- bitbucket issue create "<title>" --kind=<bug/task/enhancement> – Create an issue
- bitbucket issue update <issue-id> --status=<open/closed/resolved> – Update issue status
- bitbucket issue delete <issue-id> – Delete an issue

Pull Request Management

- bitbucket pullrequest create --source <branch> --destination <branch> – Create a pull request
- bitbucket pullrequest list – List pull requests

- bitbucket pullrequest merge <id> – Merge a pull request
- bitbucket pullrequest approve <id> – Approve a pull request
- bitbucket pullrequest decline <id> – Decline a pull request

Webhooks:

  - Go to **Repo → Repository Settings → Webhooks**
  - Choose event types like repo:push, pullrequest:created, etc.

---

2. Continuous Integration (CI) & Continuous Deployment (CD)

- **Jenkins** (pipelines, declarative scripts)

Jenkins Installation (Ubuntu)

 # Install Java (Jenkins requires Java)
sudo apt update && sudo apt install -y openjdk-17-jdk

 # Add Jenkins repository & install Jenkins
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/" | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt update && sudo apt install -y jenkins

 # Start & Enable Jenkins
sudo systemctl enable --now jenkins

 # Check Jenkins status

sudo systemctl status jenkins

 # Access Jenkins UI at http://your-server-ip:8080
sudo cat /var/lib/jenkins/secrets/initialAdminPassword

---

1. Basic Jenkins Commands

systemctl start jenkins # Start Jenkins service
systemctl stop jenkins # Stop Jenkins service
systemctl restart jenkins # Restart Jenkins service
systemctl status jenkins # Check Jenkins service status
journalctl -u jenkins -f # View real-time logs

2. Jenkins CLI Commands

java -jar jenkins-cli.jar -s http://localhost:8080 list-jobs # List all jobs
java -jar jenkins-cli.jar -s http://localhost:8080 build <job-name> # Trigger a job
java -jar jenkins-cli.jar -s http://localhost:8080 delete-job <job-name> # Delete a job
java -jar jenkins-cli.jar -s http://localhost:8080 enable-job <job-name> # Enable a job
java -jar jenkins-cli.jar -s http://localhost:8080 disable-job <job-name> # Disable a job
java -jar jenkins-cli.jar -s http://localhost:8080 who-am-i # Show current user info

3. Jenkins Environment Variables

JENKINS_HOME # Jenkins home directory
BUILD_NUMBER # Current build number
JOB_NAME # Job name
WORKSPACE # Workspace directory
GIT_COMMIT # Git commit hash of the build
BUILD_URL # URL of the build
NODE_NAME # Name of the node the build is running on

4. Jenkins Pipeline (Declarative)

groovy

```
pipeline {
    agent any
    environment {
        APP_ENV = 'production'
    }
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-repo.git'
            }
        }
        stage('Build') {
            steps {
                sh 'mvn clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
        }
        stage('Deploy') {
            steps {
                sh 'scp target/*.jar user@server:/deploy/'
            }
        }
    }
}
```

5. Jenkins Pipeline (Scripted)

groovy

```
node {
    stage('Checkout') {
        git 'https://github.com/your-repo.git'
    }
    stage('Build') {
        sh 'mvn clean package'
    }
    stage('Test') {
        sh 'mvn test'
    }
    stage('Deploy') {
        sh 'scp target/*.jar user@server:/deploy/'
    }
}
```

6. Jenkins Webhook (GitHub Example)

1. **Go to GitHub Repo > Settings > Webhooks**
2. **Add URL**: http://<jenkins-url>/github-webhook/
3. **Select** application/json as content type
4. **Choose** Just the push event
5. **Save and trigger a push event to test**

7. Manage Plugins via CLI

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin <plugin-name>  # Install a plugin
java -jar jenkins-cli.jar -s http://localhost:8080 list-plugins                  # List installed plugins
java -jar jenkins-cli.jar -s http://localhost:8080 safe-restart                  # Restart safely after installing plugins
```

8. Manage Jenkins Jobs via CLI

java -jar jenkins-cli.jar -s http://localhost:8080 create-job my-job < job-config.xml # Create a job
java -jar jenkins-cli.jar -s http://localhost:8080 get-job my-job > job-config.xml # Export job config
java -jar jenkins-cli.jar -s http://localhost:8080 update-job my-job < job-config.xml # Update job config

9. Backup & Restore Jenkins

cp -r $JENKINS_HOME /backup/jenkins_$(date +%F) # Backup Jenkins
cp -r /backup/jenkins_<date>/* $JENKINS_HOME # Restore Jenkins

10. Jenkins Security Commands

java -jar jenkins-cli.jar -s http://localhost:8080 reload-configuration # Reload config
java -jar jenkins-cli.jar -s http://localhost:8080 safe-shutdown # Safe shutdown
java -jar jenkins-cli.jar -s http://localhost:8080 restart # Restart Jenkins

11. Jenkins Credentials via CLI

java -jar jenkins-cli.jar -s http://localhost:8080 create-credentials-by-xml system::system::jenkins _ < credentials.xml

12. Jenkins Node Management

java -jar jenkins-cli.jar -s http://localhost:8080 list-nodes # List all nodes
java -jar jenkins-cli.jar -s http://localhost:8080 create-node <node-name> # Create a new node
java -jar jenkins-cli.jar -s http://localhost:8080 delete-node <node-name> # Delete a node

13. Jenkins Job Trigger Examples

groovy

```
trigger {
   cron('H 4 * * *')  # Run at 4 AM every day
}
trigger {
   pollSCM('H/5 * * * *')  # Check SCM every 5 minutes
}
```

14. Jenkins File Parameter Example

groovy

```
pipeline {
   agent any
   parameters {
      file(name: 'configFile')
   }
   stages {
      stage('Read File') {
         steps {
            sh 'cat ${configFile}'
         }
      }
   }
}
```

15. Jenkins Docker Pipeline Example

groovy

```
pipeline {
   agent {
      docker {
         image 'maven:3.8.7'
      }
```

```
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn clean package'
            }
        }
    }
}
```

Jenkins Pipeline Scripts with DevOps Tools

1. Basic CI/CD Pipeline

groovy

```
pipeline {
    agent any
    stages {
        stage('Clone Repository') {
            steps {
                git 'https://github.com/user/repo.git'
            }
        }
        stage('Build') {
            steps {
                sh 'mvn clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
```

```
        }
    }
    stage('Deploy') {
        steps {
            sh 'scp target/app.jar user@server:/deploy/path'
        }
    }
  }
}
```

---

2. Docker Build & Push

groovy

```
pipeline {
  agent any
  environment {
    DOCKER_HUB_USER = 'your-dockerhub-username'
  }
  stages {
    stage('Build Docker Image') {
      steps {
        sh 'docker build -t my-app:latest .'
      }
    }
    stage('Push to Docker Hub') {
      steps {
        withDockerRegistry([credentialsId: 'docker-hub-credentials', url: '']) {
          sh 'docker tag my-app:latest $DOCKER_HUB_USER/my-app:latest'
          sh 'docker push $DOCKER_HUB_USER/my-app:latest'
        }
      }
    }
  }
```

```
}
```

---

3. Kubernetes Deployment

```
groovy

pipeline {
    agent any
    stages {
        stage('Deploy to Kubernetes') {
            steps {
                sh 'kubectl apply -f k8s/deployment.'
            }
        }
    }
}
```

---

4. Terraform Deployment

```
groovy

pipeline {
    agent any
    stages {
        stage('Terraform Init') {
            steps {
                sh 'terraform init'
            }
        }
        stage('Terraform Apply') {
            steps {
                sh 'terraform apply -auto-approve'
            }
```

```
        }
    }
}
```

---

5. Security Scanning with Trivy

groovy

```
pipeline {
    agent any
    stages {
        stage('Scan with Trivy') {
            steps {
                sh 'trivy image my-app:latest'
            }
        }
    }
}
```

---

6. SonarQube Code Analysis

groovy

```
pipeline {
    agent any
    environment {
        SONAR_TOKEN = credentials('sonar-token')
    }
    stages {
        stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv('SonarQube') {
                    sh 'mvn sonar:sonar -Dsonar.login=$SONAR_TOKEN'
```

```
            }
        }
    }
  }
}
```

- **GitHub Actions** (workflows, syntax)

GitHub Actions allows automation for CI/CD pipelines directly within GitHub repositories.

- **Workflows** (.github/workflows/*.yml): Defines automation steps.
- **Jobs:** Runs tasks inside a workflow.
- **Steps:** Individual commands executed in jobs.
- **Actions:** Predefined or custom reusable commands.

Commands

 # Initialize a GitHub Actions workflow

mkdir -p .github/workflows && touch .github/workflows/main.yml

 # Validate GitHub Actions workflow syntax

act -l # List available workflows

act -j <job-name> # Run a specific job locally

act -w .github/workflows/main.yml # Run the workflow locally

 # Set up GitHub Actions runner

gh workflow list # List workflows in the repo

gh workflow run <workflow-name> # Manually trigger a workflow

gh workflow enable <workflow-name> # Enable a workflow

gh workflow disable <workflow-name> # Disable a workflow

Manage workflow runs

gh run list # List recent workflow runs

gh run view <run-id> # View details of a specific workflow run

gh run rerun <run-id> # Rerun a failed workflow

gh run cancel <run-id> # Cancel a running workflow

gh run delete <run-id> # Delete a workflow run

 # Manage workflow artifacts

gh run download -n <artifact-name> # Download artifacts from a workflow run

gh run view --log # View logs of the latest workflow run

 # Manage secrets for GitHub Actions

gh secret list # List repository secrets

gh secret set <SECRET_NAME> --body <value> # Add or update a secret

gh secret remove <SECRET_NAME> # Remove a secret

 # Using GitHub Actions Cache

actions/cache@v3 # GitHub Actions cache

actions/upload-artifact@v3 # Upload artifacts

actions/download-artifact@v3 # Download artifacts

 # Run a workflow manually via API

curl -X POST -H "Authorization: token <YOUR_GITHUB_TOKEN>" \

    -H "Accept: application/vnd.github.v3+json" \

https://api.github.com/repos/<owner>/<repo>/actions/workflows/<workflow_file>/dispatches \

    -d '{"ref":"main"}'

Github Actions Workflow:

🔹 **Basic GitHub Actions Workflow**

📌 **File:** .github/workflows/ci-cd.yml

name: CI/CD Pipeline

on:
  push:

```
    branches:
      - main
  pull_request:

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Set Up Java
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '17'

      - name: Build with Maven
        run: mvn clean package

      - name: Upload Build Artifact
        uses: actions/upload-artifact@v4
        with:
          name: application
          path: target/*.jar
```

🔹 **Docker Build & Push to Docker Hub**

📌 **File:** .github/workflows/docker.yml

```
name: Docker Build & Push
```

```
on:
  push:
    branches:
      - main

jobs:
  docker:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Build and Push Docker Image
        run: |
          docker build -t my-app:latest .
          docker tag my-app:latest ${{ secrets.DOCKER_USERNAME
}}/my-app:latest
          docker push ${{ secrets.DOCKER_USERNAME }}/my-app:latest
```

🔹 **Kubernetes Deployment**

📌 **File:** .github/workflows/k8s.yml

```
name: Deploy to Kubernetes

on:
```

```
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Set Up Kubectl
        uses: azure/setup-kubectl@v3
        with:
          version: 'latest'

      - name: Apply Kubernetes Manifest
        run: kubectl apply -f k8s/deployment.
```

🔹 **Terraform Deployment**

📌 **File:** .github/workflows/terraform.yml

```
name: Terraform Deployment

on:
  push:
    branches:
      - main

jobs:
  terraform:
    runs-on: ubuntu-latest
```

```
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Set Up Terraform
        uses: hashicorp/setup-terraform@v3

      - name: Terraform Init & Apply
        run: |
          terraform init
          terraform apply -auto-approve
```

🔹 **Security Scanning with Trivy**

📌 **File:** .github/workflows/trivy.yml

```
name: Security Scan with Trivy

on:
  push:
    branches:
      - main

jobs:
  scan:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Run Trivy Image Scan
        run: |
          docker pull ${{ secrets.DOCKER_USERNAME }}/my-app:latest
```

trivy image ${{ secrets.DOCKER_USERNAME }}/my-app:latest

🔹 **SonarQube Code Analysis**

📌 **File:** .github/workflows/sonarqube.yml

```
name: SonarQube Analysis

on:
  push:
    branches:
      - main

jobs:
  sonar:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Set Up Java
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '17'

      - name: Run SonarQube Analysis
        run: mvn sonar:sonar -Dsonar.login=${{ secrets.SONAR_TOKEN }}
```

🔹 **Upload & Deploy to AWS S3**

📌 **File:** .github/workflows/s3-upload.yml

```yaml
name: Upload to S3

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Upload Files to S3
        run: |
          aws s3 sync . s3://my-bucket-name --delete
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_KEY }}
```

- **GitLab CI/CD** (stages, jobs, runners)

GitLab CI/CD Basics

- **.gitlab-ci.yml**: Defines the CI/CD pipeline in the repository root.
- **Stages:** Pipeline execution order (build, test, deploy).
- **Jobs:** Specific tasks in each stage.
- **Runners:** Machines executing jobs (shared or self-hosted).
- **Artifacts:** Files preserved after job execution.

🔹 **Basic GitLab CI/CD Pipeline**

📌 **File:** .gitlab-ci.yml

```
stages:
  - build
  - test
  - deploy

build:
  stage: build
  script:
    - echo "Building application..."
    - mvn clean package
  artifacts:
    paths:
      - target/*.jar

test:
  stage: test
  script:
    - echo "Running tests..."
    - mvn test

deploy:
  stage: deploy
  script:
    - echo "Deploying application..."
    - scp target/*.jar user@server:/deploy/path
  only:
    - main
```

🔹 **Docker Build & Push to GitLab Container Registry**

📌 **File:** .gitlab-ci.yml

```
variables:
  IMAGE_NAME: registry.gitlab.com/your-namespace/your-repo

stages:
  - build
  - push

build:
  stage: build
  script:
    - docker build -t $IMAGE_NAME:latest .
  only:
    - main

push:
  stage: push
  script:
    - echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER
--password-stdin $CI_REGISTRY
    - docker push $IMAGE_NAME:latest
  only:
    - main
```

🔹 **Kubernetes Deployment**

📌 **File:** .gitlab-ci.yml

```
stages:
  - deploy
```

```
deploy:
  stage: deploy
  image: bitnami/kubectl
  script:
    - kubectl apply -f k8s/deployment.
  only:
    - main
```

🔹 **Terraform Deployment**

📌 **File:** .gitlab-ci.yml

```
image: hashicorp/terraform:latest

stages:
  - terraform

terraform:
  stage: terraform
  script:
    - terraform init
    - terraform apply -auto-approve
  only:
    - main
```

🔹 **Security Scanning with Trivy**

📌 **File:** .gitlab-ci.yml

```
stages:
  - security_scan

security_scan:
  stage: security_scan
  script:
    - docker pull registry.gitlab.com/your-namespace/your-repo:latest
    - trivy image registry.gitlab.com/your-namespace/your-repo:latest
  only:
    - main
```

🔹 **SonarQube Code Analysis**

📌 **File:** .gitlab-ci.yml

```
image: maven:3.8.7

stages:
  - analysis

sonarqube:
  stage: analysis
  script:
    - mvn sonar:sonar -Dsonar.login=$SONAR_TOKEN
  only:
    - main
```

🔹 **AWS S3 Upload**

📌 **File:** .gitlab-ci.yml

```
stages:
  - deploy

deploy_s3:
  stage: deploy
  script:
    - aws s3 sync . s3://my-bucket-name --delete
  only:
    - main
  environment:
    name: production
```

🔹 **Notify on Slack**

📌 **File:** .gitlab-ci.yml

```
notify:
  stage: notify
  script:
    - curl -X POST -H 'Content-type: application/json' --data '{"text":"Deployment completed successfully!"}' $SLACK_WEBHOOK_URL
  only:
    - main
```

🔹 **Tekton**

🔹 **What is Tekton?**

Tekton is a **Kubernetes-native CI/CD framework** that allows you to create and run pipelines for **automating builds, testing, security scans, and deployments**. It provides reusable components such as **Tasks, Pipelines, and PipelineRuns**, making it ideal for cloud-native DevOps workflows.

🔹 **Installation (On Kubernetes Cluster)**

1️⃣ **Install Tekton Pipelines**

kubectl apply -f
https://storage.googleapis.com/tekton-releases/pipeline/latest/release.

2️⃣ **Verify Installation**
kubectl get pods -n tekton-pipelines

3️⃣ **Install Tekton CLI (tkn)**
curl -LO https://github.com/tektoncd/cli/releases/latest/download/tkn-linux-amd64
chmod +x tkn-linux-amd64
sudo mv tkn-linux-amd64 /usr/local/bin/tkn

4️⃣ **Check Tekton CLI Version**
tkn version

🔹 **Tekton Basics**

- **Tasks:** The smallest execution unit in Tekton.

- **Pipelines:** A sequence of tasks forming a CI/CD process.
- **PipelineRuns:** Executes a pipeline.
- **TaskRuns:** Executes a task.
- **Workspaces:** Used for sharing data between tasks.
- **Resources:** Defines input/output artifacts (e.g., Git repositories, images).

🔹 **Install Tekton on Kubernetes**

kubectl apply -f
https://storage.googleapis.com/tekton-releases/pipeline/latest/release.

Verify installation:

kubectl get pods -n tekton-pipelines

Commands:

```
# Install Tekton CLI
kubectl apply -f
https://storage.googleapis.com/tekton-releases/pipeline/latest/release.yaml

# Check Tekton Installation
tkn version  # Show Tekton CLI version
kubectl get pods -n tekton-pipelines  # List Tekton pods
kubectl get crds | grep tekton  # List Tekton-related CRDs

# Tekton Pipeline Commands
tkn pipeline list  # List all pipelines
tkn pipeline describe <pipeline-name>  # Describe a specific pipeline
tkn pipeline start <pipeline-name> --showlog  # Start a pipeline and show logs
tkn pipeline delete <pipeline-name>  # Delete a pipeline
```

Tekton Task Commands
tkn task list # List all tasks
tkn task describe <task-name> # Describe a specific task
tkn task start <task-name> --showlog # Start a task and show logs
tkn task delete <task-name> # Delete a task

Tekton PipelineRun Commands
tkn pipelinerun list # List pipeline runs
tkn pipelinerun describe <pipelinerun-name> # Describe a pipeline run
tkn pipelinerun logs <pipelinerun-name> # Show logs of a pipeline run
tkn pipelinerun delete <pipelinerun-name> # Delete a pipeline run

Tekton TaskRun Commands
tkn taskrun list # List task runs
tkn taskrun describe <taskrun-name> # Describe a task run
tkn taskrun logs <taskrun-name> # Show logs of a task run
tkn taskrun delete <taskrun-name> # Delete a task run

Tekton Resources Commands
tkn resource list # List all pipeline resources
tkn resource describe <resource-name> # Describe a specific resource
tkn resource delete <resource-name> # Delete a resource

Tekton Triggers Commands
tkn triggerbinding list # List trigger bindings
tkn triggertemplate list # List trigger templates
tkn eventlistener list # List event listeners
tkn eventlistener logs <listener-name> # Show logs of an event listener

Tekton Debugging & Monitoring
kubectl logs -l app=tekton-pipelines-controller -n tekton-pipelines # View Tekton controller logs
kubectl get pods -n tekton-pipelines # List running Tekton pods
kubectl describe pod <pod-name> -n tekton-pipelines # Get details of a specific pod

Delete All Tekton Resources
kubectl delete pipelineruns --all -n <namespace>
kubectl delete taskruns --all -n <namespace>
kubectl delete pipelines --all -n <namespace>
kubectl delete tasks --all -n <namespace>

🔹 **Basic Tekton Task**

📌 **File:** task.

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: echo-task
spec:
  steps:
    - name: echo-message
      image: alpine
      script: |
        #!/bin/sh
        echo "Hello from Tekton Task!"
```

Apply:

kubectl apply -f task.

Run the task:

tkn task start echo-task

🔹 **Tekton Pipeline with Tasks**

📌 **File:** pipeline.

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: sample-pipeline
spec:
  tasks:
    - name: build-task
      taskRef:
        name: build-task
    - name: deploy-task
      taskRef:
        name: deploy-task
      runAfter:
        - build-task
```

Apply:

```
kubectl apply -f pipeline.
```

🔹 **Tekton PipelineRun**

📌 **File:** pipelinerun.

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
```

```
  name: sample-pipelinerun
spec:
  pipelineRef:
    name: sample-pipeline
```

Run the pipeline:

kubectl apply -f pipelinerun.

Check status:

tkn pipelinerun describe sample-pipelinerun

🔹 **Build & Push Docker Image**

📌 **File:** task-build-push.

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: build-push-task
spec:
  steps:
    - name: build-and-push
      image: gcr.io/kaniko-project/executor:latest
      script: |
        #!/bin/sh
        /kaniko/executor --context=/workspace/source
--destination=docker.io/myrepo/myapp:latest
```

🔹 **Kubernetes Deployment with Tekton**

📌 **File:** task-k8s-deploy.

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: deploy-task
spec:
  steps:
    - name: apply-manifest
      image: bitnami/kubectl
      script: |
        #!/bin/sh
        kubectl apply -f k8s/deployment.
```

🔹 **Tekton with Terraform**

📌 **File:** task-terraform.

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: terraform-task
spec:
  steps:
    - name: terraform-apply
      image: hashicorp/terraform:latest
      script: |
        #!/bin/sh
```

```
terraform init
terraform apply -auto-approve
```

🔹 **Security Scanning with Trivy**

📌 **File:** task-trivy.

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: trivy-scan
spec:
  steps:
    - name: run-trivy
      image: aquasec/trivy:latest
      script: |
        #!/bin/sh
        trivy image docker.io/myrepo/myapp:latest
```

🔹 **SonarQube Code Analysis**

📌 **File:** task-sonarqube.

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: sonarqube-task
spec:
```

```
  steps:
    - name: sonar-scan
      image: maven:3.8.7
      script: |
        #!/bin/sh
        mvn sonar:sonar -Dsonar.login=$SONAR_TOKEN
```

🔹 **Notify on Slack**

📌 **File:** task-slack.

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: slack-notify
spec:
  steps:
    - name: send-slack-message
      image: curlimages/curl:latest
      script: |
        #!/bin/sh
        curl -X POST -H 'Content-type: application/json' --data '{"text":"Deployment completed successfully!"}' $SLACK_WEBHOOK_URL
```

Circle CI

Introduction

CircleCI is a cloud-based CI/CD tool that automates software testing and deployment. It provides seamless integration with GitHub, Bitbucket, and other version control systems, enabling automated builds, tests, and deployments.

Installation

- Sign up at CircleCI
- Connect your repository (GitHub, Bitbucket)
- Configure the .circleci/config.yml file in your project

CircleCI Commands (Pipeline Example)

circleci checkout # Check out the repository to the current directory
circleci sphere list # List all the available workspaces in your account
circleci config process # Process the CircleCI config file and output the final configuration
circleci step halt # Halt the current job execution, useful in workflows
circleci job follow <job_id> # Stream the logs of a specific job in real-time
circleci pipeline trigger <pipeline_id> # Trigger a pipeline by its ID
circleci pipeline list # List all the pipelines for your project
circleci project status <project_slug> # View the status of the project
circleci sphere create <sphere_name> # Create a new workspace
circleci sphere remove <sphere_name> # Remove a workspace
circleci sync # Sync CircleCI configuration for a given project
circleci orb publish <orb_name> <version> <path_to_orb> # Publish a new version of an orb

---

CircleCI Pipeline Script Example

Basic CircleCI Pipeline Configuration

version: 2.1 # Define the CircleCI version

```
# Define jobs
jobs:
  build:
    docker:
      - image: circleci/python:3.8  # Use a Python 3.8 Docker image
    steps:
      - checkout  # Check out the code
      - run:
          name: Install dependencies
          command: pip install -r requirements.txt
      - run:
          name: Run tests
          command: pytest

  deploy:
    docker:
      - image: circleci/python:3.8
    steps:
      - checkout
      - run:
          name: Deploy application
          command: ./deploy.sh  # Custom deploy script

# Define workflows (Job execution order)
workflows:
  version: 2
  build_and_deploy:
    jobs:
      - build
      - deploy:
          requires:
            - build  # Ensure deployment happens after build succeeds
```

Advanced CircleCI Features

1. Running Jobs Based on Branches

```
jobs:
  deploy:
    docker:
      - image: circleci/python:3.8
    steps:
      - checkout
      - run:
          name: Deploy to Production
          command: ./deploy_production.sh
workflows:
  version: 2
  deploy_to_production:
    jobs:
      - deploy:
          filters:
            branches:
              only: main  # Deploy only on the 'main' branch
```

2. Caching Dependencies to Speed Up Builds

```
jobs:
  build:
    docker:
      - image: circleci/python:3.8
    steps:
      - checkout
      - restore_cache:
          keys:
```

```
        - v1-dependencies-{{ checksum "requirements.txt" }}
    - run:
        name: Install dependencies
        command: pip install -r requirements.txt
    - save_cache:
        paths:
          - ~/.cache/pip  # Save pip cache
        key: v1-dependencies-{{ checksum "requirements.txt" }}
```

3. Using Environment Variables for Sensitive Data

```
jobs:
  deploy:
    docker:
      - image: circleci/python:3.8
    steps:
      - checkout
      - run:
          name: Deploy using environment variables
          command: ./deploy.sh
    environment:
      API_KEY: $API_KEY  # Use stored API keys
```

4. Running Jobs Conditionally Based on File Changes

```
jobs:
  deploy:
    docker:
      - image: circleci/python:3.8
    steps:
      - checkout
```

```
      - run:
          name: Deploy Application
          command: ./deploy.sh
    filters:
      branches:
        only: main
    requires:
      - build
    when:
      changes:
        - Dockerfile  # Only run deploy if the Dockerfile changes
```

5. Running Tests in Parallel

```
jobs:
  test:
    docker:
      - image: circleci/python:3.8
    parallelism: 4  # Run 4 test jobs in parallel
    steps:
      - checkout
      - run:
          name: Run tests
          command: pytest
```

6. Using Multiple Docker Containers

```
jobs:
  build:
    docker:
      - image: circleci/python:3.8
```

```
    - image: circleci/postgres:13  # Additional container for PostgreSQL
      environment:
        POSTGRES_USER: circleci
  steps:
    - checkout
    - run:
        name: Install dependencies
        command: pip install -r requirements.txt
    - run:
        name: Run database migrations
        command: python manage.py migrate
    - run:
        name: Run tests
        command: pytest
```

7. Running Jobs Manually (Manual Approvals)

```
jobs:
  manual_deploy:
    docker:
      - image: circleci/python:3.8
    steps:
      - checkout
      - run:
          name: Deploy to Production
          command: ./deploy.sh
    when: manual  # Only run when triggered manually
```

8. Sending Notifications on Job Failure

```
workflows:
  version: 2
  notify_on_failure:
    jobs:
```

```
    - build
  notification:
    email:
      - user@example.com  # Send email notifications on failures
```

9. Running Multiple Jobs in Parallel

```
workflows:
  version: 2
  build_and_deploy:
    jobs:
      - build
      - deploy:
          requires:
            - build  # Deploy after build completes
          filters:
            branches:
              only: main
```

ArgoCD (GitOps)

Introduction

Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes. It enables the deployment of applications from Git repositories to Kubernetes clusters, ensuring that the live state of the cluster matches the desired state defined in Git.

Installation

1. **Install Argo CD CLI**

macOS:
brew install argocd

Linux:
curl -sSL -o /usr/local/bin/argocd
https://github.com/argoproj/argo-cd/releases/download/v2.5.4/argocd-linux-amd64
chmod +x /usr/local/bin/argocd

Install Argo CD on Kubernetes
kubectl create namespace argocd
kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.

Accessing the Argo CD UI

Access the Argo CD API Server (Local Port Forwarding)
kubectl port-forward svc/argocd-server -n argocd 8080:443

1. **Login to the Argo CD UI**

Initial Password (default is admin and the password is the name of the pod running Argo CD):
kubectl get pods -n argocd
kubectl logs <argocd-server-pod-name> -n argocd | grep "admin"

Argo CD Commands

Login to Argo CD via CLI
argocd login <ARGOCD_SERVER> --username admin --password <password>

View the current applications
argocd app list

1. **Sync an Application**
    - Syncs the application with the desired state from the Git repository.

argocd app sync <app-name>

Get Application Status
argocd app get <app-name>

2. **Create an Application**
    - Creates an app in Argo CD by specifying the Git repository, target namespace, and project.

```
argocd app create <app-name> \
  --repo <git-repository-url> \
  --path <path-to-k8s-manifests> \
  --dest-server https://kubernetes.default.svc \
  --dest-namespace <namespace>
```

Delete an Application
argocd app delete <app-name>

3. **Refresh Application**
    - Refreshes the application state from the Git repository.

argocd app refresh <app-name>

Application Resources and Syncing

Sync Status Check
argocd app sync <app-name> --prune

1. **Compare with Live**
    - Compare the live state of an application with the Git repository.

argocd app diff <app-name>

2. **Manual Sync**
    - Manually sync the application state.

argocd app sync <app-name> --force

Managing Projects

Create a Project

argocd proj create <project-name> \
  --description "<description>" \
  --dest-namespace <namespace> \
  --dest-server <server-url>

List Projects

argocd proj list

Add a Git Repo to a Project

argocd proj add-repo <project-name> --repo <git-repository-url>

GitOps and Source Repositories

Add a Git Repository

argocd repo add <git-repo-url> --username <username> --password <password> --type git

List Repositories
argocd repo list

Remove a Git Repository
argocd repo rm <git-repo-url>

Notifications and Alerts

1. **Enable Notifications**
    - Install Argo CD Notifications to integrate with Slack, email, etc.

kubectl apply -k github.com/argoproj-labs/argocd-notifications/manifests/install

2. **Set Up Notification Settings**
    - Configure notification settings in the Argo CD UI.

Application Health and Troubleshooting

Check Application Health
argocd app health <app-name>

1. **Check Logs**
    - View logs for troubleshooting.

kubectl logs <pod-name> -n argocd

2. **App Rollback**
    - Rollback to a previous revision of an application.

argocd app rollback <app-name> <revision>

Argo CD in CI/CD Pipelines

- **Integrate with CI/CD**
    - Add Argo CD commands in Jenkins, GitLab CI, or GitHub Actions pipelines to automatically deploy updates to Kubernetes based on changes in Git repositories.

Best Practices

- **Declarative GitOps**: Keep all manifests in Git, and let Argo CD automatically synchronize and deploy them.
- **Namespaces and Projects**: Use projects to group applications and limit resource access across environments.
- **RBAC**: Use Role-Based Access Control (RBAC) to secure Argo CD's access and resource usage.

Flux CD

Introduction

Flux CD is a GitOps tool for Kubernetes that automates deployment, updates, and rollback of applications using Git as the source of truth.

Installation

Install Flux CLI

curl -s https://fluxcd.io/install.sh | sudo

Verify Installation

flux --version

Bootstrap Flux in a Cluster

```
flux bootstrap github \
  --owner=<GITHUB_USER_OR_ORG> \
  --repository=<REPO_NAME> \
  --branch=main \
  --path=clusters/my-cluster \
  --personal
```

Key Flux CD Commands

General Commands

```
flux check                  # Check Flux installation
flux install                # Install Flux components in a cluster
flux bootstrap github ...       # Set up Flux in GitHub repository
flux version                # Show Flux CLI version
```

Managing Deployments

```
flux get sources git            # List Git sources
flux get kustomizations         # List kustomizations
flux reconcile kustomization <name> # Force sync a kustomization
flux suspend kustomization <name>   # Pause updates for a kustomization
flux resume kustomization <name>    # Resume updates for a kustomization
```

Git Repository Management

```
flux create source git my-app \
  --url=https://github.com/my-org/my-app \
  --branch=main \
  --interval=1m

flux create kustomization my-app \
  --source=my-app \
  --path="./deploy" \
```

```
  --prune=true \
  --interval=5m
```

Helm Chart Management

```
flux create source helm my-chart \
  --url=https://charts.bitnami.com/bitnami \
  --interval=1h

flux create helmrelease my-app \
  --source=my-chart \
  --chart=nginx \
  --values=./values. \
  --interval=5m
```

Monitoring and Debugging

```
flux logs                       # View Flux logs
flux get sources helm           # List Helm sources
flux get helmreleases           # List deployed Helm releases
flux trace kustomization <name>    # Trace errors in a kustomization
flux suspend source git <name>     # Suspend Git syncing
flux resume source git <name>      # Resume Git syncing
```

Uninstall Flux

```
flux uninstall --silent
```

---

3. Infrastructure as Code (IaC)

- **Terraform**

Terraform Installation on Ubuntu/Debian:

1. **Terraform Installation**:
    - **Official Link**: Terraform Installation

Commands:
wget -O - https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform

2. **AWS CLI Installation on Ubuntu**:
    - **Official Link**: AWS CLI Installation

Commands:
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

3. **Kubectl Installation on Ubuntu**:

Commands:
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

Step-by-Step Configuration Guide with AWS EC2 Instance

1. Main Terraform Configuration: main.tf:

Terraform Configuration Example:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }
  required_version = ">= 1.2.0"
}

provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "app_server" {
  ami           = "ami-08d70e59c07c61a3a"
  instance_type = "t2.micro"

  tags = {
    Name = var.instance_name
  }
}
```

2. Input Variables: variables.tf:

Example:
hcl

```
variable "instance_name" {
  description = "Value of the Name tag for the EC2 instance"
  type        = string
  default     = "ExampleAppServerInstance"
}
```

3. Output Values: outputs.tf:

Example:
hcl

```
output "instance_id" {
  description = "ID of the EC2 instance"
  value       = aws_instance.app_server.id
}


output "instance_public_ip" {
  description = "Public IP address of the EC2 instance"
  value       = aws_instance.app_server.public_ip
}
```

4. Running the Configuration:

Initialize Terraform:
terraform init

Apply the Configuration:
terraform apply

- Confirm by typing yes when prompted.

Inspect Output Values:
terraform output

Destroy the Infrastructure:
terraform destroy

Terraform Advanced Configuration Use Cases

1. Provider Configuration:

```
provider "aws" {
  region = "us-west-2"
}
```

2. Resource Creation:

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  tags = {
    Name = "ExampleInstance"
  }
}
```

3. Variable Management:

```
variable "region" {
  default = "us-west-2"
}

provider "aws" {
  region = var.region
}
```

4. State Management:

Example for using remote state in S3:
hcl

```
terraform {
  backend "s3" {
    bucket         = "my-tfstate-bucket"
    key            = "terraform/state"
    region         = "us-west-2"
    encrypt        = true
    dynamodb_table = "terraform-locks"
  }
}
```

5. Modules:

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  name   = "my-vpc"
  cidr   = "10.0.0.0/16"

  azs             = ["us-west-2a", "us-west-2b"]
  public_subnets  = ["10.0.1.0/24", "10.0.2.0/24"]
  private_subnets = ["10.0.3.0/24", "10.0.4.0/24"]
}
```

Terraform Commands Cheat Sheet

- **terraform init**: Initializes the Terraform configuration.
- **terraform fmt**: Formats configuration files.
- **terraform validate**: Validates the configuration files.
- **terraform plan**: Previews changes to be applied.
- **terraform apply**: Applies the changes to reach the desired state.
- **terraform destroy**: Destroys the infrastructure and removes it from the state.
- **terraform show**: Displays the current state of resources.
- **terraform state list**: Lists resources in the current state.
- **terraform taint <resource>**: Marks a resource for recreation.
- **terraform import <resource> <resource_id>**: Imports existing resources into Terraform.
- **terraform providers**: Lists the providers used in the configuration.

Terraform Best Practices

- Use **Version Control** to manage your Terraform code.

- Break your code into **Modules** for reusability.
- Use **Remote State** (e.g., AWS S3, Terraform Cloud) to store state files.
- Always run **terraform plan** before **terraform apply**.
- Use **terraform fmt** & **terraform validate** to ensure code correctness.
- Avoid **hardcoding secrets**; use environment variables or secret management tools.
- Keep configurations **modular** and **well-documented**.

Ansible (playbooks, roles, inventory)

1. Ansible Basics

Check version:
ansible --version

Check inventory:
ansible-inventory --list -y

Ping all hosts:
ansible all -m ping

Run command on all hosts:
ansible all -a "uptime"

2. Inventory & Configuration

- Default inventory: /etc/ansible/hosts

Custom inventory:
ansible -i inventory.ini all -m ping

Define hosts in inventory.ini:
ini

[web]

web1 ansible_host=192.168.1.10 ansible_user=ubuntu

[db]

db1 ansible_host=192.168.1.20 ansible_user=root

3. Ad-Hoc Commands

Run as a specific user:
ansible all -m ping -u ubuntu --become

Copy file to remote host:
ansible all -m copy -a "src=/etc/hosts dest=/tmp/hosts"

Install a package (example: nginx):
ansible all -m apt -a "name=nginx state=present" --become

4. Playbook Structure

```
- name: Install Nginx
  hosts: web
  become: yes
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: present
```

Run the playbook:
ansible-playbook install_nginx.yml

5. Variables & Facts

Define variables in vars.yml:
nginx_version: latest

Use variables in playbook:

```
- name: Install Nginx
  apt:
    name: nginx={{ nginx_version }}
    state: present
```

Display all facts:
ansible all -m setup

6. Handlers & Notifications

```
- name: Restart Nginx
  hosts: web
  become: yes
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
```

```
    state: present
  notify: Restart Nginx

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

7. Loops & Conditionals

Loop over items:

```
- name: Install multiple packages
  apt:
    name: "{{ item }}"
    state: present
  loop:
    - nginx
    - curl
    - git
```

Conditional execution:

```
- name: Restart service only if Nginx is installed
  service:
```

```
    name: nginx
    state: restarted
  when: ansible_facts['pkg_mgr'] == 'apt'
```

8. Roles & Reusability

Create a role:
Ansible-galaxy init my_role

Run a role in a playbook:
```
- hosts: web
  roles:
    - my_role
```

9. Debugging & Testing

Debug a variable:

```
- debug:
    msg: "The value of nginx_version is {{ nginx_version }}"
```

Check playbook syntax:
ansible-playbook myplaybook.yml --syntax-check

Run in dry mode:
ansible-playbook myplaybook.yml --check

Ansible Playbook

1. Playbook Structure

```
- name: Example Playbook
  hosts: all
  become: yes
  tasks:
    - name: Print a message
      debug:
        msg: "Hello, Ansible!"
```

Run the playbook:
ansible-playbook playbook.yml

---

2. Defining Hosts & Privilege Escalation

```
- name: Install Nginx
  hosts: web
  become: yes
```

Run as a specific user:

```
- name: Install package
  apt:
    name: nginx
    state: present
```

```
  become_user: root
```

---

3. Tasks & Modules

```
- name: Ensure Nginx is installed
  hosts: web
  become: yes
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: present
```

- **Common Modules**
  - command: Run shell commands
  - copy: Copy files
  - service: Manage services
  - user: Manage users
  - file: Set file permissions

---

4. Using Variables

Define variables inside the playbook:

```
vars:
  package_name: nginx
```

Use them in tasks:

```
- name: Install {{ package_name }}
  apt:
    name: "{{ package_name }}"
    state: present
```

Load external variables from vars.yml:

```
- name: Load Variables
  include_vars: vars.yml
```

---

5. Conditionals

```
- name: Restart Nginx only if installed
  service:
    name: nginx
    state: restarted
  when: ansible_facts['pkg_mgr'] == 'apt'
```

---

6. Loops

```
- name: Install multiple packages
  apt:
    name: "{{ item }}"
    state: present
  loop:
```

```
    - nginx
    - git
    - curl
```

---

7. Handlers

```
- name: Install Nginx
  apt:
    name: nginx
    state: present
  notify: Restart Nginx

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

---

8. Debugging & Testing

```
- name: Debug Variable
  debug:
    msg: "The server is running {{ ansible_distribution }}"
```

Check syntax:
ansible-playbook playbook.yml --syntax-check

Dry run:
ansible-playbook playbook.yml --check

---

9. Roles (Best Practice)

Create a role:
ansible-galaxy init my_role

Use the role in a playbook:
- hosts: web

  roles:

    - my_role

- **CloudFormation** (stacks, templates)

1. CloudFormation Concepts

- **Stack** → A group of AWS resources defined in a template.
- **Template** → /JSON file defining resources and configurations.
- **StackSet** → Deploys stacks across multiple accounts and regions.
- **Change Set** → Previews updates before applying changes.
- **Rollback** → Automatic stack rollback if an error occurs.
- **Drift Detection** → Identifies manual changes made outside CloudFormation.

---

2. CloudFormation Template Example

```
AWSTemplateFormatVersion: "2010-09-09"

Description: "Basic AWS CloudFormation Example"

Resources:

  MyBucket:

    Type: "AWS::S3::Bucket"

  MyEC2Instance:

    Type: "AWS::EC2::Instance"

    Properties:

      InstanceType: "t2.micro"

      ImageId: "ami-0abcdef1234567890"

Outputs:

  InstanceID:

    Description: "The Instance ID"

    Value: !Ref MyEC2Instance
```

---

3. CloudFormation CLI Commands

Stack Operations

aws cloudformation create-stack --stack-name my-stack --template-body file://template.

aws cloudformation update-stack --stack-name my-stack --template-body file://template.

aws cloudformation delete-stack --stack-name my-stack

Viewing Stack Details

aws cloudformation describe-stacks --stack-name my-stack

aws cloudformation list-stack-resources --stack-name my-stack

aws cloudformation describe-stack-events --stack-name my-stack

Change Set (Preview Changes)

aws cloudformation create-change-set --stack-name my-stack --template-body file://template. --change-set-name my-change-set

aws cloudformation describe-change-set --stack-name my-stack --change-set-name my-change-set

aws cloudformation execute-change-set --stack-name my-stack --change-set-name my-change-set

Drift Detection

aws cloudformation detect-stack-drift --stack-name my-stack

aws cloudformation describe-stack-drift-detection-status --stack-name my-stack

---

4. CloudFormation Best Practices

✅ Use Parameters for Flexibility
Define parameters to make templates reusable:

```
Parameters:
  InstanceType:
    Type: String
    Default: "t2.micro"
    AllowedValues: ["t2.micro", "t2.small", "t2.medium"]
```

✅ Use Mappings for Region-Specific Configurations

```
Mappings:
  RegionMap:
    us-east-1:
      AMI: "ami-12345678"
    us-west-1:
      AMI: "ami-87654321"
```

✅ Use Conditions for Conditional Resource Creation

```
Conditions:
  IsProd: !Equals [!Ref "Environment", "prod"]
Resources:
  MyDatabase:
    Type: "AWS::RDS::DBInstance"
    Condition: IsProd
```

✅ Use Outputs to Export Values

```
Outputs:
  S3BucketName:
    Description: "Name of the created S3 bucket"
    Value: !Ref MyBucket
    Export:
      Name: MyBucketExport
```

✅ Use Nested Stacks for Large Templates

Break large stacks into smaller, reusable nested stacks.

---

5. CloudFormation Troubleshooting

Issue	Solution

Stack creation fails	Check describe-stack-events for error details.
Parameter validation error	Ensure correct parameter types and values.
Rollback triggered	Check logs and describe-stack-events to debug.
Resources stuck in DELETE_FAILED	Manually delete dependencies before retrying.
Template validation error	Use aws cloudformation validate-template --template-body file://template..

---

4. Containerization & Orchestration

- **Docker** (build, run, volumes, networks, compose)

Basic Commands

- docker ps – List running containers
- docker ps -a – List all containers
- docker info – Get Docker configuration
- docker version – Get Docker version

Image Commands

- docker build -t <image>:<tag> . – Build an image from a Dockerfile
- docker login <repository> – Authenticate with a remote repository
- docker push <image>:<tag> – Push an image to a repository
- docker pull <image>:<tag> – Pull an image from a repository

- docker images – List locally available images
- docker create <image>:<tag> – Create a container from an image
- docker rmi <image> – Delete an image
- docker save <image> – Save an image as a tarball
- docker search <image> – Search for an image in a repository

Container Commands

- docker inspect <container> – View container details
- docker stats <container> – Display live resource usage
- docker logs <container> – View container logs
- docker run <container> – Run a container
- docker kill <container> – Force stop a running container
- docker start <container> – Start a stopped container
- docker stop <container> – Gracefully stop a running container
- docker restart <container> – Restart a container
- docker rm <container> – Remove a container
- docker port <container> – Show port mappings
- docker pause <container> – Suspend container processes
- docker unpause <container> – Resume container processes

Network Commands

- docker network ls – List networks
- docker network inspect <network> – View network details
- docker network create <network> – Create a network
- docker network rm <network> – Delete a network
- docker network connect <network> <container> – Connect a container to a network
- docker network disconnect <network> <container> – Disconnect a container from a network

Volume Commands

- docker volume ls – List volumes
- docker volume inspect <volume> – View volume details
- docker volume create <volume> – Create a volume
- docker volume rm <volume> – Delete a volume

Copy & Execution Commands

- docker cp <container>:<source_path> <dest_path> – Copy from container to host
- docker cp <source_path> <container>:<dest_path> – Copy from host to container
- docker exec -ti <container> <command> – Run a command inside a running container

Dockerfile Commands

- FROM <image>:<tag> – Base image for the container
- COPY <source> <destination> – Copy files/directories
- ADD <source> <destination> – Copy files & extract archives
- CMD ["command", "arg1"] – Default command executed in container
- ENTRYPOINT ["command", "arg1"] – Container's main command
- LABEL key=value – Add metadata
- ENV key=value – Set environment variables
- EXPOSE <port> – Declare exposed ports
- RUN <command> – Run command during image build
- WORKDIR <path> – Set working directory

System & Diagnostics

- docker system df – Show Docker disk usage
- docker system info – Display system details
- docker diff <container> – Show modified files in a container
- docker top <container> – Show running processes inside a container

---

General Best Practices for Dockerfiles:

1. **Minimize Layers**: Combine RUN, COPY, and ADD commands to reduce layers and image size.
2. **Use Specific Versions**: Always specify versions for base images (e.g., FROM python:3.9-slim).
3. **.dockerignore**: Use .dockerignore to exclude unnecessary files (e.g., .git, node_modules).
4. **Multi-Stage Builds**: Separate the build process and runtime environment to optimize image size.

5. **Non-root User**: Always create and use a non-root user for security.
6. **Leverage Docker Cache**: Copy dependencies first, so Docker can cache them for faster builds.

Dockerfile Examples with different Programming language

1. Python (Flask/Django)

dockerfile

```
FROM python:3.9-slim AS base

WORKDIR /app


# Install dependencies

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt


# Copy the app files

COPY . .


EXPOSE 5000


# Run as a non-root user

RUN useradd -m appuser

USER appuser


CMD ["python", "app.py"]
```

Best Practices:

- --no-cache-dir to prevent caching Python packages.
- Copy requirements.txt first to leverage Docker cache.
- Use a non-root user (appuser).

---

2. Node.js

dockerfile

FROM node:16-alpine AS build

WORKDIR /app

 # Install dependencies

COPY package.json package-lock.json ./

RUN npm install --production

 # Copy the app code

COPY . .

EXPOSE 3000

 # Run as a non-root user

RUN addgroup --system app && adduser --system --ingroup app app

USER app

CMD ["node", "app.js"]

Best Practices:

- Use --production to avoid installing devDependencies.
- Multi-stage builds for optimized images.
- Use a non-root user (app).

---

3. Java (Spring Boot)

dockerfile

FROM openjdk:17-jdk-slim AS build

WORKDIR /app

 # Copy the jar file

COPY target/myapp.jar myapp.jar

EXPOSE 8080

 # Run as a non-root user

RUN addgroup --system app && adduser --system --ingroup app app

USER app

CMD ["java", "-jar", "myapp.jar"]

Best Practices:

- Multi-stage builds for separating build and runtime.
- Use -jdk-slim for smaller images.
- Non-root user (app).

---

4. Ruby on Rails

dockerfile

```
FROM ruby:3.0-alpine

# Install dependencies
RUN apk add --no-cache build-base

WORKDIR /app

# Install Ruby gems
COPY Gemfile Gemfile.lock ./
RUN bundle install --without development test

# Copy the app code
COPY . .

EXPOSE 3000

# Run as a non-root user
```

RUN addgroup --system app && adduser --system --ingroup app app

USER app

CMD ["rails", "server", "-b", "0.0.0.0"]

Best Practices:

- Install dependencies in one RUN statement.
- Avoid devDependencies in production.
- Use non-root user (app).

---

5. Go

dockerfile

FROM golang:1.16-alpine AS build

WORKDIR /app

 # Copy and install dependencies

COPY go.mod go.sum ./

RUN go mod tidy

 # Copy the app code and build

COPY . .

RUN go build -o myapp .

 # Use a minimal base image for running

```
FROM alpine:latest

WORKDIR /app

# Copy the binary
COPY --from=build /app/myapp .

EXPOSE 8080

# Run as a non-root user
RUN addgroup --system app && adduser --system --ingroup app app
USER app

CMD ["./myapp"]
```

Best Practices:

- Multi-stage build to separate build and runtime.
- alpine for smaller runtime images.
- Non-root user (app).

---

6. Angular (Frontend)

dockerfile

```
# Build stage
```

```
FROM node:16 AS build

WORKDIR /app

COPY . .

RUN npm install

RUN npm run build --prod

# Production stage using nginx

FROM nginx:alpine

# Copy build artifacts from the build stage

COPY --from=build /app/dist/ /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Best Practices:

- Multi-stage build: separate build and serving phases.
- Use nginx:alpine for a minimal serving environment.
- Copy only production build files.

---

7. PHP (Laravel)

dockerfile

```
FROM php:8.0-fpm

# Install dependencies
RUN apt-get update && apt-get install -y libzip-dev && docker-php-ext-install zip

WORKDIR /var/www/html

# Install Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

# Install PHP dependencies
COPY composer.json composer.lock ./
RUN composer install --no-dev --no-scripts

# Copy application files
COPY . .

EXPOSE 9000

# Run as a non-root user
RUN useradd -ms /bin/ appuser
```

```
USER appuser


CMD ["php-fpm"]
```

Best Practices:

- Use Composer for PHP dependency management.
- Avoid dev dependencies in production (--no-dev).
- Run PHP-FPM as a non-root user.

---

8. Best Practices for Security and Optimization:

- **Minimize Image Size**: Use smaller base images like alpine or slim, and multi-stage builds to reduce the final image size.
- **Use a Non-root User**: Always run applications as a non-root user to enhance security.
- **Pin Versions**: Avoid using the latest tag for images. Use specific versions to ensure predictable builds.
- **Leverage Caching**: Place frequently changing files (e.g., source code) after dependencies to take advantage of Docker's build cache.
- **Avoid ADD Unless Necessary**: Use COPY instead of ADD unless you need to fetch files from a URL or extract archives.

Docker Compose Commands

- **docker-compose up**: Start all services in the background
- **docker-compose up -d**: Start services in detached mode
- **docker-compose up --build**: Rebuild images before starting services
- **docker-compose down**: Stop and remove containers, networks, volumes
- **docker-compose down -v**: Remove volumes along with containers
- **docker-compose stop**: Stop running containers without removing them
- **docker-compose start**: Restart stopped containers

- **docker-compose restart**: Restart all containers
- **docker-compose ps**: List running containers
- **docker-compose logs**: Show logs from containers
- **docker-compose logs -f**: Follow container logs
- **docker-compose exec <service> <cmd>**: Execute a command inside a running container
- **docker-compose run <service> <cmd>**: Run a one-time command inside a service
- **docker-compose config**: Validate and view merged configuration
- **docker-compose version**: Show Docker Compose version

Docker Compose (docker-compose.yml) Example

```
version: '3.8'

services:

  app:

    image: my-app:latest

    container_name: my_app

    ports:

      - "8080:80"

    environment:

      - NODE_ENV=production

    volumes:

      - ./app:/usr/src/app

    depends_on:

      - db


  db:
```

```
    image: postgres:latest
    container_name: my_db
    restart: always
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydatabase
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

Key Directives

- **version**: Defines the Compose file format version
- **services**: Defines all application services
- **image**: Specifies the container image
- **container_name**: Names the container explicitly
- **build**: Specifies build context for Dockerfile
- **ports**: Maps container ports to host
- **volumes**: Mounts persistent storage
- **environment**: Passes environment variables
- **depends_on**: Specifies dependencies between services
- **restart**: Defines restart policy (always, unless-stopped, on-failure)

General Docker Compose Structure

```
version: '3'
services:
  service_name:
    image: <image-name>        # The image to use
    build: .                   # Path to the Dockerfile if you need to build the image
    container_name: <name>     # Container name (optional)
    ports:
      - "<host-port>:<container-port>"  # Exposing ports
    environment:
      - VAR_NAME=value         # Set environment variables
    volumes:
      - <host-path>:<container-path> # Mount volumes for persistent data
    depends_on:
      - other_service          # Define service dependencies
    networks:
      - <network-name>         # Assign the service to a network
```

Docker Compose Example Configurations

1. **Python (Flask) + Redis Example:**

```
version: '3'
services:
  web:
```

```
    build: ./app
    ports:
      - "5000:5000"
    environment:
      - FLASK_APP=app.py
      - FLASK_ENV=development
    volumes:
      - ./app:/app
    networks:
      - app_network

  redis:
    image: "redis:alpine"
    networks:
      - app_network

networks:
  app_network:
    driver: bridge
```

2. **Node.js (Express) + MongoDB Example:**

```
version: '3'
services:
```

```
  app:
    build: ./node-app
    ports:
      - "3000:3000"
    environment:
      - MONGO_URI=mongodb://mongo:27017/mydb
    depends_on:
      - mongo
    networks:
      - backend

  mongo:
    image: mongo:latest
    volumes:
      - mongo_data:/data/db
    networks:
      - backend

networks:
  backend:
    driver: bridge

volumes:
```

```
  mongo_data:
```

3. **Nginx + PHP (Laravel) Example:**

```
version: '3'
services:
  nginx:
    image: nginx:alpine
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./html:/usr/share/nginx/html
    ports:
      - "8080:80"
    depends_on:
      - php
    networks:
      - frontend

  php:
    image: php:8.0-fpm
    volumes:
      - ./html:/var/www/html
    networks:
      - frontend
```

```
networks:
  frontend:
    driver: bridge
```

Best Practices

- **Use Versioning**: Always specify a version for Docker Compose files (e.g., version: '3')
- **Define Volumes**: Use named volumes for persistent data (e.g., database storage)
- **Environment Variables**: Use environment variables for configuration (e.g., database connection strings)
- **Use depends_on**: Ensure proper start order for dependent services
- **Custom Networks**: Use custom networks for better service communication management
- **Avoid latest Tag**: Always use specific version tags for predictable builds

---

Advanced Options

- **Build Arguments**: Pass information during the image build process

```
build:
  context: .
  args:
    NODE_ENV: production
```

- **Health Checks**: Add health checks to monitor service status

```
services:
  web:
```

```
  image: my-web-app
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost/health"]
    interval: 30s
    retries: 3
```

- **Scaling Services**: Scale services using the command

```
docker-compose up --scale web=3
```

- **<u>Kubernetes (K8s)</u>**

1. Kubernetes Basics

kubectl cluster-info – Display cluster information
kubectl get nodes – List all nodes in the cluster
kubectl get pods – List all pods in the current namespace
kubectl get services – List all services
kubectl get deployments – List all deployments

2. Managing Pods

kubectl run my-pod --image=nginx – Create a pod with Nginx
kubectl delete pod my-pod – Delete a pod
kubectl logs my-pod – View pod logs
kubectl exec -it my-pod -- /bin/sh – Access a pod's shell

3. Managing Deployments

kubectl create deployment my-deploy --image=nginx – Create a deployment
kubectl scale deployment my-deploy --replicas=3 – Scale deployment to 3 replicas
kubectl rollout status deployment my-deploy – Check rollout status
kubectl rollout undo deployment my-deploy – Rollback to the previous version

4. Managing Services

kubectl expose deployment my-deploy --type=NodePort --port=80 – Expose deployment as a service
kubectl get svc – List services
kubectl describe svc my-service – Get service details

5. Namespaces

kubectl get ns – List all namespaces
kubectl create namespace dev – Create a new namespace
kubectl delete namespace dev – Delete a namespace

6. ConfigMaps & Secrets

kubectl create configmap my-config --from-literal=key=value – Create a ConfigMap
kubectl get configmap – List ConfigMaps
kubectl create secret generic my-secret --from-literal=password=12345 – Create a secret
kubectl get secrets – List secrets

7. Troubleshooting

kubectl get events – View cluster events
kubectl describe pod my-pod – Get detailed pod information
kubectl logs my-pod – View logs of a specific pod
kubectl top pod – Show resource usage of pods

8. Helm (Package Manager for Kubernetes)

helm repo add stable https://charts.helm.sh/stable – Add a Helm repo
helm install my-release stable/nginx – Install a Helm chart
helm list – List installed releases
helm delete my-release – Uninstall a release

Persistent Volumes & Storage

kubectl get pvc – List persistent volume claims
kubectl get pv – List persistent volumes
kubectl describe pvc <pvc> – Describe a persistent volume claim
kubectl delete pvc <pvc> – Delete a persistent volume claim

Autoscaling

kubectl autoscale deployment <deployment> --cpu-percent=50 --min=1 --max=10 – Enable autoscaling
kubectl get hpa – View horizontal pod autoscaler

Kubernetes Debugging

kubectl get events --sort-by=.metadata.creationTimestamp – Show events
kubectl describe pod <pod> – Show pod details

kubectl logs <pod> – Check logs
kubectl exec -it <pod> -- /bin/sh – Access pod shell

Kubernetes YAML Configurations

1. Pod

yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: nginx
      image: nginx:latest
      ports:
        - containerPort: 80
```

2. Deployment

yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

3. ReplicaSet

yaml

```
apiVersion: apps/v1
kind: ReplicaSet
```

```
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: nginx
          image: nginx:latest
```

4. Service (ClusterIP, NodePort, LoadBalancer)

ClusterIP (default)

yaml

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

NodePort

```
yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  type: NodePort
  ports:
```

```
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30080
```

LoadBalancer

yaml

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

5. ConfigMap

yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  key1: value1
  key2: value2
```

6. Secret

yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  password: cGFzc3dvcmQ=  # Base64 encoded value
```

7. Persistent Volume (PV)

yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /mnt/data
```

8. Persistent Volume Claim (PVC)

```
yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
```

```
  resources:
    requests:
      storage: 500Mi
```

9. Ingress

```
yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: my-service
                port:
```

```
        number: 80
```

10. Horizontal Pod Autoscaler (HPA)

yaml

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: my-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deployment
  minReplicas: 2
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
```

```
      averageUtilization: 50
```

11. CronJob

```
yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: my-cronjob
spec:
  schedule: "*/5 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: my-cron
              image: busybox
              command: ["echo", "Hello from CronJob"]
          restartPolicy: OnFailure
```

---

5. Cloud Services

- **AWS** (EC2, S3, IAM, VPC, Lambda)

AWS Cheat Sheet (EC2, S3, IAM, VPC, Lambda)

EC2 (Elastic Compute Cloud)

- aws ec2 describe-instances – List all instances
- aws ec2 start-instances --instance-ids <id> – Start an instance
- aws ec2 stop-instances --instance-ids <id> – Stop an instance
- aws ec2 terminate-instances --instance-ids <id> – Terminate an instance
- aws ec2 create-key-pair --key-name <name> – Create a key pair
- aws ec2 describe-security-groups – List security groups

S3 (Simple Storage Service)

- aws s3 ls – List buckets
- aws s3 mb s3://<bucket> – Create a bucket
- aws s3 cp <file> s3://<bucket>/ – Upload a file
- aws s3 rm s3://<bucket>/<file> – Delete a file
- aws s3 rb s3://<bucket> --force – Delete a bucket
- aws s3 sync <local-dir> s3://<bucket>/ – Sync local and S3

IAM (Identity and Access Management)

- aws iam list-users – List IAM users
- aws iam create-user --user-name <name> – Create a user
- aws iam attach-user-policy --user-name <name> --policy-arn <policy> – Attach a policy
- aws iam list-roles – List IAM roles
- aws iam create-role --role-name <name> --assume-role-policy-document file://policy.json – Create a role
- aws iam list-policies – List policies

VPC (Virtual Private Cloud)

- aws ec2 describe-vpcs – List VPCs
- aws ec2 create-vpc --cidr-block <CIDR> – Create a VPC
- aws ec2 delete-vpc --vpc-id <id> – Delete a VPC
- aws ec2 create-subnet --vpc-id <id> --cidr-block <CIDR> – Create a subnet
- aws ec2 describe-security-groups – List security groups
- aws ec2 describe-internet-gateways – List internet gateways

Lambda (Serverless Computing)

- aws lambda list-functions – List all Lambda functions
- aws lambda create-function --function-name <name> --runtime <runtime> --role <role> --handler <handler> – Create a function
- aws lambda update-function-code --function-name <name> --zip-file fileb://<file>.zip – Update function code
- aws lambda delete-function --function-name <name> – Delete a function
- aws lambda invoke --function-name <name> output.json – Invoke a function

---

- **Azure** (VMs, Storage, AKS, Functions)

Azure Cheat Sheet

1. Azure Virtual Machines (VMs)

- az vm create --resource-group <rg> --name <vm-name> --image <image>
  → Create a VM
- az vm list -o table
  → List all VMs
- az vm stop --name <vm-name> --resource-group <rg>
  → Stop a VM

- az vm start --name <vm-name> --resource-group <rg>
  → Start a VM
- az vm delete --name <vm-name> --resource-group <rg>
  → Delete a VM
- az vm resize --name <vm-name> --resource-group <rg> --size <vm-size>
  → Resize a VM
- az vm show --name <vm-name> --resource-group <rg>
  → Show VM details
- az vm open-port --port <port-number> --name <vm-name> --resource-group <rg>
  → Open a port on a VM

---

2. Azure Storage

- az storage account create --name <storage-name> --resource-group <rg> --location <region>
  → Create a storage account
- az storage container create --name <container-name> --account-name <storage-name>
  → Create a blob container
- az storage blob upload --file <file-path> --container-name <container-name> --account-name <storage-name>
  → Upload a file to Blob Storage
- az storage blob list --container-name <container-name> --account-name <storage-name>
  → List blobs in a container
- az storage account delete --name <storage-name> --resource-group <rg>
  → Delete a storage account

---

3. Azure Kubernetes Service (AKS)

- az aks create --resource-group <rg> --name <aks-name> --node-count <num> --generate-ssh-keys
  → Create an AKS cluster
- az aks get-credentials --resource-group <rg> --name <aks-name>
  → Get kubeconfig for AKS cluster
- kubectl get nodes
  → List AKS cluster nodes
- kubectl get pods -A
  → List all pods in AKS
- kubectl apply -f <file.>
  → Deploy an application in AKS
- kubectl delete -f <file.>
  → Remove an application from AKS
- az aks delete --name <aks-name> --resource-group <rg>
  → Delete an AKS cluster

---

4. Azure Functions

- az functionapp create --resource-group <rg> --name <app-name> --consumption-plan-location <region> --runtime <runtime>
  → Create an Azure Function App
- az functionapp list -o table
  → List all Function Apps
- az functionapp delete --name <app-name> --resource-group <rg>
  → Delete a Function App
- func init <app-name>
  → Initialize a local Azure Functions project
- func new
  → Create a new function
- func start
  → Run functions locally
- func azure functionapp publish <app-name>
  → Deploy function to Azure

- **GCP** (Compute Engine, GKE, Cloud Run)

1. Compute Engine (VMs)

- gcloud compute instances create <vm-name> --zone=<zone> --machine-type=<type> --image=<image>
  → Create a VM instance
- gcloud compute instances list
  → List all VM instances
- gcloud compute instances start <vm-name> --zone=<zone>
  → Start a VM instance
- gcloud compute instances stop <vm-name> --zone=<zone>
  → Stop a VM instance
- gcloud compute instances delete <vm-name> --zone=<zone>
  → Delete a VM instance
- gcloud compute ssh <vm-name> --zone=<zone>
  → SSH into a VM instance
- gcloud compute firewall-rules create <rule-name> --allow tcp:<port>
  → Open a specific port

---

2. Google Kubernetes Engine (GKE)

- gcloud container clusters create <cluster-name> --num-nodes=<num> --zone=<zone>
  → Create a GKE cluster
- gcloud container clusters get-credentials <cluster-name> --zone=<zone>
  → Get credentials for the GKE cluster
- kubectl get nodes
  → List cluster nodes
- kubectl get pods -A
  → List all running pods
- kubectl apply -f <file.>
  → Deploy an application to GKE

- kubectl delete -f <file.>
  → Remove an application from GKE
- gcloud container clusters delete <cluster-name> --zone=<zone>
  → Delete a GKE cluster

---

3. Cloud Run (Serverless Containers)

- gcloud run deploy <service-name> --image=<gcr.io/project/image> --platform=managed --region=<region> --allow-unauthenticated
  → Deploy an application to Cloud Run
- gcloud run services list
  → List all deployed Cloud Run services
- gcloud run services update-traffic <service-name> --to-latest
  → Update Cloud Run service to the latest image
- gcloud run services delete <service-name>
  → Delete a Cloud Run service
- gcloud run services update <service-name> --set-env-vars VAR_NAME=value
  → Set environment variables in a Cloud Run service

---

6. Configuration Management

- **Chef** (recipes, cookbooks)

Basic Concepts

- **Recipe**: Defines a set of resources to configure a system.
- **Cookbook**: A collection of recipes, templates, and attributes.
- **Resource**: Represents system objects (e.g., package, service, file).
- **Node**: A machine managed by Chef.
- **Run List**: Specifies the order in which recipes are applied.

- **Attributes**: Variables used to customize recipes.

Commands

```
chef-client                    # Run Chef on a node
knife cookbook create my_cookbook     # Create a new cookbook
knife node list                 # List all nodes
knife role list                 # List all roles
chef-solo -c solo.rb -j run_list.json # Run Chef in solo mode
```

Example Recipe

```
package 'nginx' do
  action :install
end

service 'nginx' do
  action [:enable, :start]
end

file '/var/www/html/index.html' do
  content '<h1>Welcome to Chef</h1>'
end
```

- **Puppet** (manifests, modules)

Basic Concepts

- **Manifest**: A file defining resources and configurations (.pp).
- **Module**: A collection of manifests, templates, and files.
- **Class**: A reusable block of Puppet code.
- **Node**: A system managed by Puppet.
- **Fact**: System information collected by Facter.
- **Resource**: The basic unit of configuration (e.g., package, service).

Commands

puppet apply my_manifest.pp # Apply a local manifest

puppet module install my_module # Install a module

puppet agent --test # Run Puppet on an agent node

puppet resource service nginx # Check a resource state

Example Manifest

puppet

```
class nginx {
  package { 'nginx':
    ensure => installed,
  }
```

```
  service { 'nginx':
    ensure => running,
    enable => true,
  }

  file { '/var/www/html/index.html':
    content => '<h1>Welcome to Puppet</h1>',
    mode    => '0644',
  }
}

include nginx
```

- **SaltStack** (states, grains)

Basic Concepts

- **State**: Defines configurations and how they should be enforced.
- **Grain**: System metadata like OS, CPU, and memory.
- **Pillar**: Secure data storage for variables.
- **Minion**: A node managed by the Salt master.
- **Master**: The central server controlling minions.

Commands

```
salt '*' test.ping                # Check connectivity with minions
```

```
salt '*' pkg.install nginx             # Install a package on all minions
salt '*' service.start nginx           # Start a service
salt '*' grains.items                  # Show all grains for a minion
salt '*' state.apply webserver         # Apply a state to minions
```

Example State (nginx.sls)

```
nginx:
  pkg.installed: []
  service.running:
    - enable: true

/var/www/html/index.html:
  file.managed:
    - source: salt://webserver/index.html
    - mode: 644
```

---

7. Monitoring & Logging

Prometheus & Grafana (Metrics, Alerts, Visualization)

Prometheus Basics

prometheus.yml configuration

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'my-app'
    static_configs:
      - targets: ['localhost:9090']
```

- **Query Metrics (PromQL)**
  - up → Check if a target is up
  - http_requests_total → Total HTTP requests
  - rate(http_requests_total[5m]) → Requests per second in the last 5 min
  - sum by (status) (rate(http_requests_total[1m])) → Request count by status

Grafana Basics

Install & Start

```
sudo apt install grafana

sudo systemctl start grafana-server
```

- **Data Sources:**
  - Prometheus → http://localhost:9090
  - Elasticsearch → http://localhost:9200
- **Create Dashboard & Alerts**

- Add Panel → Select metric
- Set Alert Conditions → **Thresholds, No Data, Query Errors**

ELK Stack (Elasticsearch, Logstash, Kibana)

Elasticsearch Commands

Basic Index Operations

curl -X GET "localhost:9200/_cat/indices?v"

curl -X PUT "localhost:9200/my-index"

curl -X DELETE "localhost:9200/my-index"

Search & Query

curl -X GET "localhost:9200/my-index/_search?pretty"

curl -X POST "localhost:9200/my-index/_doc/1" -H "Content-Type: application/json" -d '{"name": "test"}'

Logstash Configuration

logstash.conf

```
input {
  file {
    path => "/var/log/syslog"
    start_position => "beginning"
  }
```

```
}
output {
  elasticsearch {
    hosts => ["localhost:9200"]
  }
}
```

Kibana Basics

Start Kibana
systemctl start kibana

- **Useful Queries**
  - message: "error" → Search for logs with "error"
  - status:[400 TO 500] → Find logs with HTTP errors

Datadog

Agent Installation & Setup

Install Agent (Linux)
DD_API_KEY=your_api_key -c "$(curl -L https://s3.amazonaws.com/dd-agent/scripts/install_script.sh)"

- **Log Monitoring**
  - /etc/datadog-agent/datadog.

logs_enabled: true

- Restart agent

systemctl restart datadog-agent

- **Metric Queries**
  - avg:system.cpu.user{*} → CPU usage
  - top(avg:system.disk.used{*}, 5, 'mean') → Top 5 disk users

---

New Relic

Install New Relic Agent

For Linux Servers
curl -Ls https://download.newrelic.com/install/newrelic-cli/scripts/install.sh |

newrelic install

- **Query Logs & Metrics**

NRQL Queries (New Relic Query Language)
sql

SELECT average(cpuPercent) FROM SystemSample SINCE 30 minutes ago

SELECT count(*) FROM Transaction WHERE appName = 'my-app'

---

8. Security & Compliance

1. SonarQube (Code Analysis)

Basic Commands

Start SonarQube Server
./sonar.sh start

Run analysis with Maven
mvn sonar:sonar

Run analysis with SonarScanner CLI
sonar-scanner -Dsonar.projectKey=<project-key> -Dsonar.sources=.

- **View results**: Open **http://localhost:9000**

1. SonarQube Integration

Jenkins Integration

groovy

```
pipeline {
    agent any
    environment {
        SONAR_SCANNER_HOME = tool 'SonarQubeScanner'
    }
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-repo.git'
            }
```

```
        }
        stage('SonarQube Analysis') {
            steps {
                script {
                    withSonarQubeEnv('SonarQubeServer') {
                        sh 'mvn sonar:sonar'
                    }
                }
            }
        }
    }
}
```

GitLab CI/CD Integration

```
yaml

stages:
  - code_analysis


sonarqube_scan:
  stage: code_analysis
```

```
  image: maven:3.8.7-openjdk-17

  script:

    - mvn sonar:sonar -Dsonar.host.url=$SONAR_HOST_URL
-Dsonar.login=$SONAR_TOKEN

  variables:

    SONAR_HOST_URL: "http://sonarqube-server:9000"

    SONAR_TOKEN: "your-sonarqube-token"
```

GitHub Actions Integration

yaml

```
name: SonarQube Analysis

on:

  push:

    branches:

      - main

jobs:

  sonar_scan:

    runs-on: ubuntu-latest
```

```
  steps:
    - name: Checkout Code
      uses: actions/checkout@v4

    - name: Set up JDK
      uses: actions/setup-java@v3
      with:
        distribution: 'temurin'
        java-version: '17'

    - name: Run SonarQube Scan
      run: mvn sonar:sonar -Dsonar.host.url=$SONAR_HOST_URL
-Dsonar.login=$SONAR_TOKEN
      env:
        SONAR_HOST_URL: "http://sonarqube-server:9000"
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
```

ArgoCD Integration (PreSync Hook)

yaml

```
apiVersion: batch/v1
kind: Job
```

```
metadata:
  name: sonarqube-analysis
  annotations:
    argocd.argoproj.io/hook: PreSync
spec:
  template:
    spec:
      containers:
        - name: sonar-scanner
          image: maven:3.8.7-openjdk-17
          command: ["mvn", "sonar:sonar"]
          env:
            - name: SONAR_HOST_URL
              value: "http://sonarqube-server:9000"
            - name: SONAR_TOKEN
              valueFrom:
                secretKeyRef:
                  name: sonar-secret
                  key: sonar-token
      restartPolicy: Never
```

---

2. Trivy (Container Vulnerability Scanning)

Basic Commands

Scan a Docker image
trivy image <image-name>

Scan a Kubernetes cluster
trivy k8s cluster

Generate a JSON report
trivy image --format json -o report.json <image-name>

Jenkins Integration

groovy

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-repo.git'
            }
        }
        stage('Trivy Scan') {
            steps {
                sh 'trivy image your-docker-image:latest'
```

```
        }
      }
    }
}
```

GitLab CI/CD Integration

yaml

```
stages:
  - security_scan

trivy_scan:
  stage: security_scan
  image: aquasec/trivy
  script:
    - trivy image your-docker-image:latest --format json -o trivy_report.json
  artifacts:
    paths:
      - trivy_report.json
```

GitHub Actions Integration

```yaml
name: Trivy Scan
on:
  push:
    branches:
      - main

jobs:
  trivy_scan:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Run Trivy Scan
        run: |
          docker pull your-docker-image:latest
          trivy image your-docker-image:latest --format json --output trivy_report.json

      - name: Upload Trivy Report
        uses: actions/upload-artifact@v4
```

```
      with:
        name: trivy-report
        path: trivy_report.json
```

ArgoCD Integration (PreSync Hook)

```
yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: trivy-scan
  annotations:
    argocd.argoproj.io/hook: PreSync
spec:
  template:
    spec:
      containers:
        - name: trivy-scanner
          image: aquasec/trivy
          command: ["trivy", "image", "your-docker-image:latest"]
      restartPolicy: Never
```

Kubernetes Integration (Admission Controller)

yaml

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: trivy-webhook
webhooks:
  - name: trivy-scan.k8s
    rules:
      - apiGroups: [""]
        apiVersions: ["v1"]
        operations: ["CREATE"]
        resources: ["pods"]
    clientConfig:
      service:
        name: trivy-webhook-service
        namespace: security
        path: /validate
    admissionReviewVersions: ["v1"]
```

sideEffects: None

OWASP Dependency-Check (Software Dependency Analysis)

Basic Commands

Run a scan on a project
./dependency-check/bin/dependency-check.sh --scan /path/to/project

Run a scan using Maven plugin
mvn org.owasp:dependency-check-maven:check

Jenkins Integration

groovy

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-repo.git'
            }
        }
        stage('OWASP Dependency Check') {
            steps {
```

```
            sh 'mvn org.owasp:dependency-check-maven:check'
        }
    }
  }
}
```

GitLab CI/CD Integration

yaml

```
stages:
  - security_scan

owasp_dependency_check:
  stage: security_scan
  image: maven:3.8.7-openjdk-17
  script:
    - mvn org.owasp:dependency-check-maven:check
  artifacts:
    paths:
      - target/dependency-check-report.html
```

GitHub Actions Integration

yaml

```
name: OWASP Dependency Check

on:
  push:
    branches:
      - main

jobs:
  owasp_dependency_check:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Run OWASP Dependency-Check
        run: mvn org.owasp:dependency-check-maven:check

      - name: Upload OWASP Report
```

```
      uses: actions/upload-artifact@v4
      with:
        name: owasp-report
        path: target/dependency-check-report.html
```

ArgoCD Integration (PreSync Hook)

yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: owasp-dependency-check
  annotations:
    argocd.argoproj.io/hook: PreSync
spec:
  template:
    spec:
      containers:
      - name: owasp-check
        image: maven:3.8.7-openjdk-17
        command: ["mvn", "org.owasp:dependency-check-maven:check"]
```

restartPolicy: Never

---

9. Networking, Ports & Load Balancing

Networking Basics

- **IP Addressing**
  - Private IPs: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
  - Public IPs: Assigned by ISPs
  - CIDR Notation: 192.168.1.0/24 (Subnet Mask: 255.255.255.0)
- **Ports**
  - HTTP: 80
  - HTTPS: 443
  - SSH: 22
  - DNS: 53
  - FTP: 21
  - MySQL: 3306
  - PostgreSQL: 5432
- **Protocols**
  - TCP (Reliable, connection-based)
  - UDP (Fast, connectionless)
  - ICMP (Used for ping)
  - HTTP(S), FTP, SSH, DNS

---

2. Network Commands

Linux Networking
Show network interfaces

ip a # Show IP addresses

ifconfig # Older command

Check connectivity
ping google.com

Trace route
traceroute google.com

DNS lookup
nslookup google.com

dig google.com

Test ports
telnet google.com 80

nc -zv google.com 443

Firewall Rules (iptables)
List firewall rules
sudo iptables -L -v

Allow SSH
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

Block an IP
sudo iptables -A INPUT -s 192.168.1.100 -j DROP

Netcat (nc)

Start a simple TCP listener
nc -lvp 8080

Send data to a listening server
echo "Hello" | nc 192.168.1.100 8080

---

3. Kubernetes Networking

List services and their endpoints
kubectl get svc -o wide

Get pods and their IPs
kubectl get pods -o wide

Port forward a service
kubectl port-forward svc/my-service 8080:80

Expose a pod
kubectl expose pod mypod --type=NodePort --port=80

---

4. Docker Networking

List networks
docker network ls

Inspect a network
docker network inspect bridge

Create a custom network
docker network create mynetwork

Run a container in a custom network
docker run -d --network=mynetwork nginx

---

5. Cloud Networking (AWS, Azure, GCP)

AWS

List VPCs
aws ec2 describe-vpcs

List subnets
aws ec2 describe-subnets

Open security group port
aws ec2 authorize-security-group-ingress --group-id sg-12345 --protocol tcp --port 22 --cidr 0.0.0.0/0

Azure
List VNets
az network vnet list -o table

List NSGs
az network nsg list -o table

Open a port in NSG
az network nsg rule create --resource-group MyGroup --nsg-name MyNSG --name AllowSSH --protocol Tcp --direction Inbound --priority 100 --source-address-prefixes '*' --source-port-ranges '*' --destination-port-ranges 22 --access Allow

---

AWS VPC Basics

- **Definition:** A logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network.
- **CIDR Block:** Define the IP range (e.g., 10.0.0.0/16).

- **Components:**
  - **Subnets:** Divide your VPC into public (with internet access) and private (without direct internet access) segments.
  - **Route Tables:** Control the traffic routing for subnets.
  - **Internet Gateway (IGW):** Allows communication between instances in your VPC and the internet.
  - **NAT Gateway/Instance:** Enables outbound internet access for instances in private subnets.
  - **VPC Peering:** Connects multiple VPCs.
  - **VPN Connections & Direct Connect:** Securely link your on-premises network with your VPC.
  - **VPC Endpoints:** Privately connect your VPC to supported AWS services.

Security Groups Essentials

- **Definition:** Virtual firewalls that control inbound and outbound traffic for your EC2 instances.
- **Key Characteristics:**
  - **Stateful:** Return traffic is automatically allowed regardless of inbound/outbound rules.
  - **Default Behavior:** All outbound traffic is allowed; inbound is denied until explicitly allowed.
- **Rule Components:**
  - **Protocol:** (TCP, UDP, ICMP, etc.)
  - **Port Range:** Specific ports or a range (e.g., port 80 for HTTP).
  - **Source/Destination:** IP addresses or CIDR blocks (e.g., 0.0.0.0/0 for all).
- **Usage:**
  - Assign one or more security groups to an instance.
  - Modify rules anytime without stopping or restarting the instance.

Common AWS CLI Commands

VPC Operations

Create a VPC:
aws ec2 create-vpc --cidr-block 10.0.0.0/16

Create a Subnet:
aws ec2 create-subnet --vpc-id <vpc-id> --cidr-block 10.0.1.0/24

Create & Attach an Internet Gateway:
aws ec2 create-internet-gateway

aws ec2 attach-internet-gateway --vpc-id <vpc-id> --internet-gateway-id <igw-id>

Associate a Route Table:
aws ec2 associate-route-table --subnet-id <subnet-id> --route-table-id <rtb-id>

Security Group Operations

Create a Security Group:
aws ec2 create-security-group --group-name MySecurityGroup --description "My security group" --vpc-id <vpc-id>

Authorize Inbound Traffic:
aws ec2 authorize-security-group-ingress --group-id <sg-id> --protocol tcp --port 80 --cidr 0.0.0.0/0

Authorize Outbound Traffic (if restricting defaults):
aws ec2 authorize-security-group-egress --group-id <sg-id> --protocol tcp --port 443 --cidr 0.0.0.0/0

List Security Groups:
aws ec2 describe-security-groups

Best Practices

- **Least Privilege:** Only open ports and protocols that are necessary.
- **Layered Security:** Use both Security Groups and Network ACLs for enhanced security.

- **Monitoring & Auditing:** Regularly review and update your security group rules.
- **Naming Conventions:** Adopt consistent naming for easy identification and management.
- **Documentation:** Keep notes on why certain rules exist to help with future troubleshooting.

Ports

DevOps Essential Port 🔥

🔹 **Networking & Security**

- **SSH** - 22 (Secure remote access)
- **FTP** - 21 (File Transfer Protocol)
- **SFTP** - 22 (Secure File Transfer Protocol)
- **Telnet** - 23 (Unsecured remote access)
- **SMTP** - 25, 587 (Email sending)
- **DNS** - 53 (Domain Name System)
- **DHCP** - 67/68 (Dynamic IP assignment)
- **HTTP** - 80 (Default web traffic)
- **HTTPS** - 443 (Secure web traffic)
- **SMB** - 445 (Windows file sharing)
- **LDAP** - 389 (Directory services)
- **LDAPS** - 636 (Secure LDAP)
- **RDP** - 3389 (Remote Desktop Protocol)

🔹 **CI/CD & DevOps Tools**

- **Jenkins** - 8080 (CI/CD automation server)
- **Git** - 9418 (Git repository access)
- **SonarQube** - 9000 (Code quality analysis)
- **Nexus Repository** - 8081 (Artifact repository)
- **Harbor** - 443 (Container registry)

- **GitLab CI/CD** - 443, 80, 22 (GitLab services & SSH)
- **Bitbucket** - 7990 (Bitbucket web UI)
- **TeamCity** - 8111 (CI/CD server)

🔹 Containerization & Orchestration

- **Docker Registry** - 5000 (Private Docker Registry)
- **Kubernetes API Server** - 6443 (Cluster API)
- **Kubelet** - 10250 (Node agent)
- **ETCD (Kubernetes Storage)** - 2379-2380 (Key-value store)
- **Flannel (Kubernetes Networking)** - 8285/8286 (Overlay network)
- **Calico (Kubernetes Networking)** - 179 (BGP Protocol)
- **Istio Ingress Gateway** - 15021, 15090 (Service mesh ingress)

🔹 Monitoring & Logging

- **Prometheus** - 9090 (Metrics monitoring)
- **Grafana** - 3000 (Visualization dashboard)
- **Elasticsearch** - 9200 (Search & analytics engine)
- **Logstash** - 5044 (Log ingestion)
- **Fluentd** - 24224 (Log collector)
- **Kibana** - 5601 (Log visualization)
- **Loki** - 3100 (Log aggregation)
- **Jaeger** - 16686 (Tracing UI)

🔹 Databases

- **PostgreSQL** - 5432 (Relational database)
- **MySQL/MariaDB** - 3306 (Relational database)
- **MongoDB** - 27017 (NoSQL database)
- **Redis** - 6379 (In-memory database)
- **Cassandra** - 9042 (NoSQL distributed database)

- **CockroachDB** - 26257 (Distributed SQL database)
- **Neo4j** - 7474 (Graph database UI), 7687 (Bolt protocol)
- **InfluxDB** - 8086 (Time-series database)
- **Couchbase** - 8091 (Web UI), 11210 (Data access)

🔹 **Message Brokers & Caching**

- **Kafka** - 9092 (Event streaming)
- **RabbitMQ** - 5672 (Message broker)
- **ActiveMQ** - 61616 (JMS messaging)
- **NATS** - 4222 (High-performance messaging)
- **Memcached** - 11211 (In-memory caching)

🔹 **Web Servers & Reverse Proxies**

- **Nginx** - 80, 443 (Web server & reverse proxy)
- **Apache HTTP** - 80, 443 (Web server)
- **HAProxy** - 443, 80 (Load balancer)
- **Caddy** - 2019 (API endpoint)

🔹 **Cloud Services & Storage**

- **AWS S3** - 443 (Object storage API)
- **AWS RDS** - 3306, 5432, 1433 (Managed databases)
- **Azure Blob Storage** - 443 (Storage API)
- **Google Cloud Storage** - 443 (Object storage API)
- **MinIO** - 9000 (S3-compatible storage)

Nginx (Reverse Proxy & Load Balancing)

What is a Reverse Proxy?

A **Reverse Proxy** is a server that forwards client requests to backend servers. It helps:
✅ **Improve security** by hiding backend servers.
✅ **Handle traffic** and **reduce load** on backend servers.
✅ **Improve performance** with caching and compression.

What is Load Balancing?

Load Balancing distributes traffic across multiple servers to:
✅ **Prevent overloading** of a single server.
✅ **Ensure high availability** (if one server fails, others handle traffic).
✅ **Improve speed** and **performance**.

1️⃣ Nginx Reverse Proxy & Load Balancing

🔹 **Reverse Proxy (Forward Requests to Backend)**

📌 When a user visits **example.com**, Nginx forwards the request to the backend server.

📝 **Configuration File (nginx.conf)**

```
server {
    listen 80;  # Listen for requests on port 80
    server_name example.com;  # Your domain name

    location / {
```

```
        proxy_pass http://backend_servers;  # Forward requests to backend
        proxy_set_header Host $host;  # Keep the original host
        proxy_set_header X-Real-IP $remote_addr;  # Send real client IP
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

✅ **What This Does:**

- Nginx **forwards requests** from example.com to **backend servers**.
- proxy_pass tells Nginx where to send traffic.
- **Keeps client IP** and **host name** intact for logs.

🔹 **Load Balancing (Distribute Traffic Across Multiple Servers)**

📌 Instead of sending all traffic to **one server**, Nginx distributes it across **multiple servers**.

📝 **Configuration File (nginx.conf)**

nginx

```
upstream backend_servers {
    server server1.example.com;  # Backend Server 1
    server server2.example.com;  # Backend Server 2
}

server {
    listen 80;
    server_name example.com;

    location / {
        proxy_pass http://backend_servers;  # Send traffic to multiple backend servers
    }
```

}

✅ **What This Does:**

- upstream defines **multiple backend servers**.
- **Traffic is balanced** between server1 and server2.
- Default method: **Round-robin** (each request goes to the next server).

2️⃣ Apache (reverse proxy, load balancing)

🔹 **Enable Required Modules**

📌 **Before using Apache as a Reverse Proxy, enable these modules:**

```
a2enmod proxy
a2enmod proxy_http
a2enmod proxy_balancer
a2enmod lbmethod_byrequests
systemctl restart apache2  # Restart Apache for changes
```

✅ **What This Does:**

- These modules allow **Apache to forward requests** and **balance traffic**.

🔹 **Reverse Proxy (Forward Requests to Backend Servers)**

📝 **Configuration File (apache.conf)**

```
<VirtualHost *:80>
    ServerName example.com  # Domain handled by Apache
```

```
    ProxyPass "/" "http://backend_servers/"
    ProxyPassReverse "/" "http://backend_servers/"
</VirtualHost>
```

✅ What This Does:

- Apache listens on **example.com** and forwards requests to **backend servers**.
- ProxyPassReverse ensures **responses return correctly** to the client.

🔹 Load Balancing (Send Traffic to Multiple Backend Servers)

📝 Configuration File (apache.conf)

```
<Proxy "balancer://mycluster">
    BalancerMember "http://server1.example.com"
    BalancerMember "http://server2.example.com"
</Proxy>

<VirtualHost *:80>
    ServerName example.com

    ProxyPass "/" "balancer://mycluster/"
    ProxyPassReverse "/" "balancer://mycluster/"
</VirtualHost>
```

✅ What This Does:

- BalancerMember defines **multiple backend servers**.
- Apache **automatically distributes** traffic using **round-robin**.

3️⃣ HAProxy (Load Balancing)

📌 HAProxy is a **lightweight and high-performance** Load Balancer for web applications.

🔹 Install HAProxy

```
apt install haproxy  # Ubuntu/Debian
yum install haproxy  # RHEL/CentOS
```

🔹 Load Balancing with HAProxy

📝 Configuration File (haproxy.cfg)

```
frontend http_front
    bind *:80  # Accept traffic on port 80
    default_backend backend_servers  # Forward traffic to backend servers

backend backend_servers
    balance roundrobin  # Distribute traffic evenly
    server server1 server1.example.com:80 check  # First server
    server server2 server2.example.com:80 check  # Second server
```

✅ What This Does:

- frontend handles **incoming requests**.
- backend defines multiple **backend servers**.
- **Round-robin** ensures traffic is **evenly distributed**.
- check makes sure **only healthy servers** receive traffic.

🔹 Restart HAProxy

```
systemctl restart haproxy
systemctl enable haproxy  # Enable on startup
```

4️⃣ Kubernetes Ingress Controller

🔹 Install Nginx Ingress Controller

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/cloud/deploy.yaml

✅ What This Does:

- Installs **Nginx Ingress Controller** for managing external traffic in Kubernetes.

🔹 Define an Ingress Resource

📝 Configuration File (ingress.yaml)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /  # Optional URL rewrite
spec:
  rules:
    - host: example.com  # Define the domain
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
```

```
      name: my-service  # Forward traffic to this Kubernetes service
      port:
        number: 80
```

✅ What This Does:

- Routes traffic from **example.com** to **my-service** inside Kubernetes.
- **Annotations modify behavior** (like URL rewriting).

🔹 Verify Ingress is Working

```
kubectl get ingress
kubectl describe ingress my-ingress
```

✅ What This Does:

- kubectl get ingress → Checks if Ingress exists.
- kubectl describe ingress → Shows detailed configuration.

🔹 Comparison Table

Tool	Feature	Use Case
Nginx	Reverse Proxy	Forwards requests to backend servers
Nginx	Load Balancing	Distributes traffic across multiple servers
Apache	Reverse Proxy	Works similarly to Nginx
Apache	Load Balancing	Uses balancer:// for traffic distribution

HAProxy	Load Balancing	High-performance, efficient traffic handling
Kubernetes Ingress	Traffic Routing	Manages external traffic in Kubernetes

---

🔹 **Which One Should You Use?**

✔ **For a simple website/API** → Use **Nginx Reverse Proxy**.
✔ **For balancing multiple servers** → Use **Nginx, Apache, or HAProxy**.
✔ **For Kubernetes applications** → Use **Ingress Controller**.

Practical Examples: Docker for Nginx, Apache, HAProxy, and Kubernetes Ingress

Step-by-step practical examples using Docker for Nginx, Apache, HAProxy, and Kubernetes Ingress.

1. Nginx Reverse Proxy & Load Balancer (With Docker)

Scenario: We have two backend servers running a Python Flask application, and we want Nginx to act as a Reverse Proxy and Load Balancer.

Step 1 Create Two Backend Servers (Flask)
Create a directory for the project

mkdir nginx-loadbalancer && cd nginx-loadbalancer

server1.py (Backend Server 1)

from flask import Flask

app = Flask(__name__)

```
@app.route('/')
def home():
    return "Hello from Server 1"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

server2.py (Backend Server 2)

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home():
    return "Hello from Server 2"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Step 2 Create a Dockerfile for Backend Servers
Dockerfile

```
FROM python:3.9
WORKDIR /app
```

```
COPY server1.py /app/

RUN pip install flask

CMD ["python", "server1.py"]
```

For server2.py, create another Dockerfile and replace server1.py with server2.py

Step 3 Create an Nginx Configuration File
nginx.conf

```
nginx

events {}

http {
    upstream backend_servers {
        server server1:5000;
        server server2:5000;
    }
    server {
        listen 80;
        server_name localhost;


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

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

Step 4 Create a Docker Compose File

docker-compose.yml

```
version: '3'

services:
  server1:
    build: .
    container_name: server1
    ports:
      - "5001:5000"

  server2:
    build: .
    container_name: server2
    ports:
      - "5002:5000"

  nginx:
```

```
    image: nginx:latest
    container_name: nginx_proxy
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - server1
      - server2
```

Step 5 Run the Containers

```
docker-compose up --build
```

Step 6 Test Load Balancing
Run the following command

```
curl http://localhost
```

Expected Output (requests will alternate)

```
Hello from Server 1
Hello from Server 2
Hello from Server 1
Hello from Server 2
```

2 Apache Reverse Proxy & Load Balancer (With Docker)

Step 1 Create Apache Configuration File
apache.conf

```
<VirtualHost *:80>
    ServerName localhost

    <Proxy "balancer://mycluster">
        BalancerMember "http://server1:5000"
        BalancerMember "http://server2:5000"
    </Proxy>

    ProxyPass "/" "balancer://mycluster/"
    ProxyPassReverse "/" "balancer://mycluster/"
</VirtualHost>
```

Step 2 Create docker-compose.yml for Apache

```
version: '3'
services:
  server1:
    build: .
    container_name: server1
```

```
  ports:
    - "5001:5000"

server2:
  build: .
  container_name: server2
  ports:
    - "5002:5000"

apache:
  image: httpd:latest
  container_name: apache_proxy
  ports:
    - "80:80"
  volumes:
    - ./apache.conf:/usr/local/apache2/conf/httpd.conf
  depends_on:
    - server1
    - server2
```

Step 3 Run Apache Proxy

```
docker-compose up --build
```

3 HAProxy Load Balancer (With Docker)

Step 1 Create HAProxy Configuration File haproxy.cfg

```
frontend http_front
    bind *:80
    default_backend backend_servers

backend backend_servers
    balance roundrobin
    server server1 server1:5000 check
    server server2 server2:5000 check
```

Step 2 Create docker-compose.yml for HAProxy

```
version: '3'
services:
  server1:
    build: .
    container_name: server1
    ports:
      - "5001:5000"
```

server2:

```
build: .
container_name: server2
ports:
  - "5002:5000"
```

haproxy:

```
image: haproxy:latest
container_name: haproxy_loadbalancer
ports:
  - "80:80"
volumes:
  - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
depends_on:
  - server1
  - server2
```

Step 3 Run HAProxy

docker-compose up --build

---

4. Kubernetes Ingress Controller

Step 1 Deploy Nginx Ingress Controller

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/cloud/deploy.

Step 2 Create Ingress Resource
ingress.

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: my-ingress

spec:

  rules:

    - host: example.com

      http:

        paths:

          - path: /

            pathType: Prefix

            backend:

              service:
```

```
      name: my-service
      port:
        number: 80
```

Step 3 Apply Ingress

kubectl apply -f ingress.

Comparison Table

Tool	Feature	Use Case
Nginx	Reverse Proxy	Forwards requests to backend servers
Nginx	Load Balancing	Distributes traffic across multiple servers
Apache	Reverse Proxy	Works similarly to Nginx
Apache	Load Balancing	Uses balancer for traffic distribution

HAProxy	Load Balancing	High-performance, efficient traffic handling
Kubernetes Ingress	Traffic Routing	Manages external traffic in Kubernetes

---

10. Database Cheat Sheet

Databases are essential for CI/CD pipelines, monitoring, logging, and cloud automation. DevOps engineers interact with databases to store configurations, manage infrastructure state, and automate deployments. This guide covers SQL, NoSQL, and cloud databases with relevant DevOps commands and use cases.

Database Automation for DevOps

Why Automate Databases in DevOps?

✔ Eliminate manual work in database provisioning, backup, and monitoring
✔ Ensure consistency across environments (dev, staging, production)
✔ Reduce downtime with automated backups and performance monitoring
✔ Enable CI/CD pipelines to manage database migrations

---

1. SQL Databases (MySQL, PostgreSQL, MariaDB)

Database Management

```
SHOW DATABASES;  -- List all databases
```

```
CREATE DATABASE db_name;  -- Create a database

DROP DATABASE db_name;  -- Delete a database

USE db_name;  -- Select a database
```

User Management

```
CREATE USER 'devops'@'localhost' IDENTIFIED BY 'password';

GRANT ALL PRIVILEGES ON db_name.* TO 'devops'@'localhost';

SHOW GRANTS FOR 'devops'@'localhost';

FLUSH PRIVILEGES;
```

Table Management

```
SHOW TABLES;

CREATE TABLE users (id INT PRIMARY KEY AUTO_INCREMENT, name
VARCHAR(100), email VARCHAR(100) UNIQUE);

DROP TABLE users;
```

Data Operations

```
INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');

SELECT * FROM users;

UPDATE users SET name = 'Bob' WHERE id = 1;

DELETE FROM users WHERE id = 1;
```

Database Backups & Restore

mysqldump -u root -p db_name > backup.sql

mysql -u root -p db_name < backup.sql

pg_dump -U postgres db_name > backup.sql

psql -U postgres -d db_name -f backup.sql

---

2. NoSQL Databases

MongoDB

show dbs;

use mydb;

db.createCollection("users");

db.users.insertOne({name: "Alice", email: "alice@example.com"});

db.users.find();

db.users.updateOne({name: "Alice"}, {$set: {email: "alice@xyz.com"}});

db.users.deleteOne({name: "Alice"});

mongodump --out /backup/

mongorestore /backup/

Redis

redis-cli

SET key "value";

GET key;

DEL key;

FLUSHALL;

Cassandra (CQL)

DESC KEYSPACES;

CREATE KEYSPACE mykeyspace WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};

USE mykeyspace;

CREATE TABLE users (id UUID PRIMARY KEY, name TEXT, email TEXT);

INSERT INTO users (id, name, email) VALUES (uuid(), 'Alice', 'alice@example.com');

SELECT * FROM users;

---

3. Database Automation for DevOps

Terraform for AWS RDS

provider "aws" {

  region = "us-east-1"

```
}


resource "aws_db_instance" "default" {
  identifier = "devops-db"
  engine = "mysql"
  instance_class = "db.t3.micro"
  allocated_storage = 20
  username = "admin"
  password = "password"
}
```

Docker Database Containers

```
docker run -d --name mysql-container -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 mysql:latest

docker run -d --name postgres-container -e POSTGRES_PASSWORD=root -p 5432:5432 postgres:latest

docker run -d --name mongo-container -p 27017:27017 mongo:latest
```

Database Monitoring with Prometheus & Grafana

✔ Install mysqld_exporter for MySQL metrics
✔ Use pg_exporter for PostgreSQL monitoring
✔ Connect Prometheus to Redis Exporter

Automating MySQL Setup with Ansible

```
- name: Install MySQL and Configure Database
  hosts: db_servers
  become: yes
  tasks:
    - name: Install MySQL Server
      apt:
        name: mysql-server
        state: present

    - name: Start MySQL Service
      service:
        name: mysql
        state: started
        enabled: yes

    - name: Create MySQL Database
      mysql_db:
        name: devops_db
        state: present
```

```
- name: Create MySQL User
  mysql_user:
    name: devops_user
    password: DevOps@123
    priv: "devops_db.*:ALL"
    state: present
```

```
ansible-playbook -i inventory mysql-setup.yml
```

Automating Database Backup & Restore with Jenkins

```
groovy
pipeline {
    agent any
    environment {
        MYSQL_ROOT_PASSWORD = credentials('mysql-root-pass')
    }
    stages {
        stage('Backup Database') {
            steps {
                sh 'mysqldump -u root -p$MYSQL_ROOT_PASSWORD devops_db > /var/backups/devops_db.sql'
```

```
        }
    }
    stage('Restore Database') {
        steps {
            sh 'mysql -u root -p$MYSQL_ROOT_PASSWORD devops_db < /var/backups/devops_db.sql'
        }
    }
  }
}
```

✔ Store mysql-root-pass as a credential in Jenkins
✔ Run the pipeline to schedule automated backups

Monitoring MySQL Performance with Prometheus & Grafana

wget https://github.com/prometheus/mysqld_exporter/releases/download/v0.14.0/mysqld_exporter-0.14.0.linux-amd64.tar.gz

tar xvf mysqld_exporter-0.14.0.linux-amd64.tar.gz

mv mysqld_exporter /usr/local/bin/

mysqld_exporter --config.my-cnf=/etc/.my.cnf &

```
scrape_configs:
  - job_name: 'mysql'
    static_configs:
      - targets: ['localhost:9104']

docker-compose up -d
```

Deploying MongoDB for DevOps Pipelines with Docker

```
version: '3.8'
services:
  mongo:
    image: mongo
    container_name: mongodb
    restart: always
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: DevOps@123
    ports:
      - "27017:27017"

docker-compose up -d
```

```
docker-compose down
```

Automating DynamoDB for Terraform State Storage

```
aws dynamodb create-table --table-name TerraformState \
    --attribute-definitions AttributeName=id,AttributeType=S \
    --key-schema AttributeName=id,KeyType=HASH \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5

aws dynamodb scan --table-name TerraformState
```

Summary of DevOps Database Automation

Tool	Purpose
Terraform	Automate AWS RDS database provisioning
Ansible	Install & configure MySQL on multiple servers
Jenkins	Automate database backup & restore

Prometheus & Grafana	Monitor MySQL performance in real-time
Docker	Deploy MongoDB for microservices
AWS CLI	Manage cloud databases like DynamoDB

---

11. Storage Cheat Sheet

1. Storage Types in DevOps

✔ **Block Storage** – Used for databases, VMs, containers (e.g., EBS, Cinder)
✔ **File Storage** – Used for shared access & persistence (e.g., NFS, EFS)
✔ **Object Storage** – Used for backups, logs, and media (e.g., S3, MinIO)

---

2. Linux Storage Commands

Disk Management

List disks & partitions:

lsblk

fdisk -l

df -h # Show disk usage

Check disk space usage:
du -sh /path/to/directory

Mount & unmount a disk:
mount /dev/sdb1 /mnt

umount /mnt

Filesystem Operations

Format a disk:
mkfs.ext4 /dev/sdb1

Check filesystem usage:
df -Th

Check disk health:
smartctl -a /dev/sdb

---

3. Cloud Storage Commands

AWS S3

List buckets:
aws s3 ls

Upload a file:
aws s3 cp file.txt s3://mybucket/

Download a file:
aws s3 cp s3://mybucket/file.txt .

Sync directories:
aws s3 sync /local/path s3://mybucket/

Azure Blob Storage

List storage accounts:
az storage account list

Upload a file:
az storage blob upload --container-name mycontainer --file file.txt --name file.txt

Download a file:
az storage blob download --container-name mycontainer --name file.txt --file file.txt

Google Cloud Storage (GCS)

List buckets:
gsutil ls

Upload a file:
gsutil cp file.txt gs://mybucket/

Download a file:
gsutil cp gs://mybucket/file.txt .

---

4. Persistent Storage for Kubernetes

PV & PVC (Persistent Volumes & Claims)

Create a Persistent Volume (PV):
yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/mnt/data"
```

Create a Persistent Volume Claim (PVC):
yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
```

```
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

Mounting Storage in a Pod

```
yaml
apiVersion: v1
kind: Pod
metadata:
  name: storage-pod
spec:
  containers:
    - name: app
      image: nginx
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: storage-volume
  volumes:
    - name: storage-volume
```

```
    persistentVolumeClaim:
      claimName: my-pvc
```

---

5. DevOps Storage Automation with Terraform

AWS S3 Bucket Creation with Terraform

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_s3_bucket" "devops_bucket" {
  bucket = "devops-backup-bucket"
  acl    = "private"
}

output "bucket_name" {
  value = aws_s3_bucket.devops_bucket.id
}
```

Azure Storage Account with Terraform

```
provider "azurerm" {
```

```
  features {}
}

resource "azurerm_storage_account" "example" {
  name                     = "devopsstorageacc"
  resource_group_name      = "devops-rg"
  location                 = "East US"
  account_tier             = "Standard"
  account_replication_type = "LRS"
}
```

---

6. Storage Monitoring & Backup

Linux Storage Monitoring

Monitor disk usage in real-time:
iotop

Check disk performance:
iostat -x 1

Backup Strategies

✔ **Full Backup:** Copies all data
✔ **Incremental Backup:** Backs up only changed files
✔ **Snapshot Backup:** Captures a point-in-time copy

Linux Backup Using rsync

rsync -av --delete /source/ /backup/

AWS S3 Backup

aws s3 sync /data s3://backup-bucket/

Azure Blob Storage Backup

az storage blob upload-batch --destination mycontainer --source /data

---

7. CI/CD Storage in DevOps

✔ **Artifacts Storage:**

- **Nexus, Artifactory, Docker Hub, Amazon ECR**
- Store and manage binaries, containers, and packages.

✔ **Logging Storage:**

- **Elasticsearch, Loki, Splunk, AWS CloudWatch, Azure Monitor**
- Collect and analyze logs for troubleshooting.

✔ **State Management in Infrastructure-as-Code:**

- **Terraform State Files (S3, Azure Blob, GCS)**
- Store Terraform state files remotely for collaboration.

---

8. Best Practices for DevOps Storage

✔ Use **object storage (S3, MinIO, GCS)** for logs and backups.
✔ Automate storage provisioning using **Terraform or Ansible**.
✔ Implement **encryption (AES-256, KMS, Secrets Manager)** for security.

✔ Optimize performance with **data compression & caching (Redis, CDN)**.
✔ Regularly monitor storage with **Prometheus, Grafana, CloudWatch**.

---

12. Helm Cheat

Helm is a package manager for Kubernetes that helps you install, update, and manage applications easily. It works like "apt" for Ubuntu or "yum" for CentOS but for Kubernetes.

---

1. What is Helm?

- **Helm** helps deploy applications in Kubernetes using pre-configured templates called **Helm Charts**.
- A **Chart** is a collection of files that describe a Kubernetes application.

---

2. Helm Basics

Commands:

helm version # Check which version of Helm is installed

helm help # Get help with Helm commands

helm repo list # Show all added Helm repositories

What does it mean?

- helm version → Shows which version of Helm you are using.
- helm help → Gives information about available commands.
- helm repo list → Shows the sources from where Helm can download charts.

---

3. Adding and Updating Repositories

A **repository** is like a store where Helm charts are stored.

sh

```
helm repo add bitnami https://charts.bitnami.com/bitnami  # Add a repository

helm repo update  # Get the latest list of available charts

helm search repo nginx  # Search for a chart (e.g., nginx)
```

Explanation:

- helm repo add → Adds a new chart repository (e.g., Bitnami, which has pre-built applications).
- helm repo update → Updates the list of available applications.
- helm search repo nginx → Searches for a chart named "nginx" in the repositories.

---

4. Installing Applications using Helm

To install an application, use this command:

```
helm install my-nginx bitnami/nginx
```

Explanation:

- helm install → Installs an application.
- my-nginx → A name you choose for this installation.

- bitnami/nginx → The application (nginx) from the Bitnami repository.

Check if the application is running:

kubectl get pods # See running applications in Kubernetes

---

5. Listing Installed Applications

helm list # Show all installed applications

Explanation:

- This will show all Helm-managed applications running in Kubernetes.

---

6. Checking Application Details

helm status my-nginx # Check the status of your installed application

helm get values my-nginx # See configuration values used

---

7. Updating an Installed Application

If you need to **update** an application (e.g., change its configuration), use:

helm upgrade my-nginx bitnami/nginx --set service.type=LoadBalancer

Explanation:

- helm upgrade → Updates an already installed application.
- --set service.type=LoadBalancer → Changes the configuration (in this case, changing the service type).

---

8. Uninstalling an Application

If you want to remove an application:

helm uninstall my-nginx

Explanation:

- helm uninstall → Deletes the installed application.

---

9. Debugging Helm Charts

Before installing an application, check for errors using:

helm lint my-chart # Checks for issues in a Helm chart

helm install --debug --dry-run my-test bitnami/nginx # Simulate installation

Explanation:

- helm lint → Checks if the Helm chart has issues.
- --debug --dry-run → Runs the installation without making changes, useful for testing.

---

10. Creating Your Own Helm Chart

If you want to create a custom chart for your application:

helm create my-chart # Creates a new Helm chart