DevOps Playbook

Table of Contents

1. Linux Command Line

Linux is the foundation of DevOps operations. These commands help you navigate systems, manage files, configure permissions, and automate tasks.

Basic Linux Commands

1. **pwd** - Print the current working directory.
2. **ls** - List files and directories.
    - **ls -la** - List all files (including hidden) in long format.
3. **cd** - Change directory.
    - **cd ..** - Go up one directory.
    - **cd ~** or **cd** - Go to the home directory.
4. **touch** - Create an empty file.
5. **mkdir** - Create a new directory.
    - **mkdir -p /path/to/nested/dir** - Create parent directories as needed.
6. **rm** - Remove files or directories.
    - **rm -f <file>** - Force remove a file.

    - **rm -r <dir>** - Recursively remove a directory.
    - **rm -rf <dir>** - Force and recursively remove a directory.
7. **rmdir** - Remove *empty* directories.
8. **cp** - Copy files or directories.
    - **cp <source> <destination>**
    - **cp -r <dir_source> <dir_destination>** - Recursively copy a directory.
9. **mv** - Move or rename files and directories.
    - **mv <old_name> <new_name>** - Rename.
    - **mv <source> <destination_dir>** - Move.
10. **cat** - Display the content of a file.
11. **echo** - Display a line of text.
    - **echo "text" > file.txt** - Overwrite file with text.
    - **echo "text" >> file.txt** - Append text to file.
12. **clear** - Clear the terminal screen.

Intermediate Linux Commands

13. **chmod** - Change file permissions.
    - **chmod +x script.sh** - Make a script executable.
    - **chmod 755 <file>** - Set permissions (Owner: rwx, Group: r-x, Others: r-x).
14. **chown** - Change file ownership.
    - **chown <user>:<group> <file>**
15. **find** - Search for files and directories.
    - **find . -name "*.txt"** - Find all files ending in .txt in the current directory.
16. **grep** - Search for text in a file.
    - **grep "error" /var/log/syslog** - Search for "error" in a file.
    - **grep -r "text" /path/to/dir** - Recursively search for "text" in a directory.
17. **wc** - Count lines, words, and characters in a file.
    - **wc -l <file>** - Count lines only.
18. **head** - Display the first few lines of a file.
    - **head -n 20 <file>** - Display first 20 lines.
19. **tail** - Display the last few lines of a file.
    - **tail -n 20 <file>** - Display last 20 lines.
    - **tail -f <file>** - Follow the file and output new lines in real-time.
20. **sort** - Sort the contents of a file.
21. **uniq** - Remove duplicate lines from a file (requires sorted input).
    - **sort file.txt | uniq -c** - Count unique lines.
22. **diff** - Compare two files line by line.
23. **tar** - Archive files into a tarball.
    - **tar -cvf archive.tar /path/to/dir** - Create.
    - **tar -xvf archive.tar** - Extract.
    - **tar -czvf archive.tar.gz /path/to/dir** - Create and gzip.
    - **tar -xzvf archive.tar.gz** - Extract gzipped archive.
24. **zip/unzip** - Compress and extract ZIP files.

25. **df** - Display disk space usage.
    - **df -h** - Human-readable format.
26. **du** - Display directory size.
    - **du -sh /path/to/dir** - Human-readable summary of a directory's total size.
27. **top** - Monitor system processes in real time.
28. **ps** - Display active processes.
    - **ps aux** - Show all processes from all users.
29. **kill** - Terminate a process by its PID.
    - **kill <PID>** - Graceful shutdown (SIGTERM).
    - **kill -9 <PID>** - Force kill (SIGKILL).
30. **ping** - Check network connectivity.
31. **wget** - Download files from the internet.
32. **curl** - Transfer data from or to a server.
    - **curl -L <url>** - Follow redirects.
    - **curl -o <file> <url>** - Download to a file.
33. **scp** - Securely copy files between systems.
    - **scp <local_file> <user>@<remote_host>:/path/**
34. **rsync** - Synchronize files and directories.
    - **rsync -avz <source_dir>/ <user>@<remote_host>:/path/**

Advanced Linux Commands

35. **awk** - Text processing and pattern scanning.
36. **sed** - Stream editor for filtering and transforming text.
    - **sed 's/foo/bar/g' file.txt** - Replace "foo" with "bar" globally in a file.
37. **cut** - Remove sections from each line of a file.
38. **tr** - Translate or delete characters.
39. **xargs** - Build and execute command lines from standard input.
40. **ln** - Create symbolic or hard links.
    - **ln -s /path/to/original /path/to/link** - Create a symbolic link.
41. **df -h** - Display disk usage in human-readable format. (Duplicate from intermediate)
42. **free** - Display memory usage.
    - **free -h** - Human-readable.
43. **iostat** - Display CPU and I/O statistics.
44. **netstat** - Network statistics.
    - **netstat -tulnp** - Show listening TCP/UDP ports and the programs using them.
45. **ifconfig/ip** - Configure network interfaces.
    - **ip addr show** - (Modern) Show IP addresses.
46. **iptables** - Configure firewall rules.
47. **systemctl** - Control the systemd system and service manager.
    - **systemctl start <service>**
    - **systemctl stop <service>**
    - **systemctl restart <service>**
    - **systemctl status <service>**

- **systemctl enable <service>** - Start on boot.
- **systemctl disable <service>** - Don't start on boot.

48. **journalctl** - View system logs (from systemd).
    - **journalctl -u <service>** - View logs for a specific service.
    - **journalctl -f** - Follow all logs.
49. **crontab** - Schedule recurring tasks.
    - **crontab -e** - Edit user's crontab.
50. **at** - Schedule tasks for a specific time.
51. **uptime** - Display system uptime.
52. **whoami** - Display the current user.
53. **users** - List all users currently logged in.
54. **hostname** - Display or set the system hostname.
55. **env** - Display environment variables.
56. **export** - Set environment variables.

Networking Commands

57. **ip addr** - Display or configure IP addresses.
58. **ip route** - Show or manipulate routing tables.
59. **traceroute** - Trace the route packets take to a host.
60. **nslookup** - Query DNS records.
61. **dig** - Query DNS servers (more detailed than nslookup).
62. **ssh** - Connect to a remote server via SSH.
    - **ssh <user>@<host>**
    - **ssh -i /path/to/key.pem <user>@<host>** - Use an identity file.
63. **ftp** - Transfer files using the FTP protocol.
64. **nmap** - Network scanning and discovery.
65. **telnet** - Communicate with remote hosts.
66. **netcat (nc)** - Read/write data over networks.

File Management and Search

67. **locate** - Find files quickly using a database.
68. **stat** - Display detailed information about a file.
69. **tree** - Display directories as a tree.
70. **file** - Determine a file's type.
71. **basename** - Extract the filename from a path.
72. **dirname** - Extract the directory part of a path.

System Monitoring

73. **vmstat** - Display virtual memory statistics.
74. **htop** - Interactive process viewer (alternative to top).
75. **lsof** - List open files.
    - **lsof -i :<port>** - Find which process is using a specific port.

76. **dmesg** - Print kernel ring buffer messages.
77. **uptime** - Show how long the system has been running. (Duplicate)
78. **iotop** - Display real-time disk I/O by processes.

Package Management

79. **apt** (Debian/Ubuntu) - Package manager.
    - **apt update**
    - **apt upgrade**
    - **apt install <package>**
    - **apt remove <package>**
80. **yum/dnf** (RHEL/CentOS/Fedora) - Package manager.
    - **dnf install <package>**
81. **snap** - Manage snap packages.
82. **rpm** - Manage RPM packages.

Disk and Filesystem

83. **mount/umount** - Mount or unmount filesystems.
84. **fsck** - Check and repair filesystems.
85. **mkfs** - Create a new filesystem.
86. **blkid** - Display information about block devices.
87. **lsblk** - List information about block devices.
88. **parted** - Manage partitions interactively.

Scripting and Automation

89. **bash** - Command interpreter and scripting shell.
90. **sh** - Legacy shell interpreter.
91. **cron** - Automate tasks. (Duplicate)
92. **alias** - Create shortcuts for commands.
93. **source** - Execute commands from a file in the current shell.

Development and Debugging

94. **gcc** - Compile C programs.
95. **make** - Build and manage projects.
96. **strace** - Trace system calls and signals.
97. **gdb** - Debug programs.
98. **git** - Version control system.
99. **vim/nano** - Text editors for scripting and editing.

Other Useful Commands

100. **uptime** - Display system uptime. (Duplicate)
101. **date** - Display or set the system date and time.
102. **cal** - Display a calendar.

103. **man** - Display the manual for a command.
104. **history** - Show previously executed commands.
105. **alias** - Create custom shortcuts for commands. (Duplicate)

Linux Playbook Scenarios & Scripts

Common Scenarios

- **Find which process is using port 80:**
  ```
  sudo lsof -i :80
  # OR
  sudo netstat -tulnp | grep :80
  ```

- **Find the top 10 largest files/directories:**
  ```
  # Find top 10 largest directories in /
  sudo du -Sh / | sort -rh | head -n 10

  # Find top 10 largest files in /
  sudo find / -type f -exec du -Sh {} + | sort -rh | head -n 10
  ```

- **Live-watch a log file for a specific error:**
  ```
  tail -f /var/log/app.log | grep -i 'ERROR'
  ```

- **Find all files modified in the last 2 days:**
  ```
  find /path/to/search -mtime -2
  ```

- **Count unique IPs accessing a web server (from access log):**
  ```
  awk '{print $1}' /var/log/nginx/access.log | sort | uniq -c | sort -nr | head -n 20
  ```

Basic Bash Script Example (backup.sh)

A simple script to back up a directory.

```
#!/bin/bash

# Set variables
SRC_DIR="/var/www/html"
DEST_DIR="/mnt/backups/web"
DATE=$(date +%Y-%m-%d-%H%M%S)
BACKUP_FILE="$DEST_DIR/web_backup_$DATE.tar.gz"

# Create backup directory if it doesn't exist
mkdir -p $DEST_DIR
```

```
# Create the compressed archive
echo "Starting backup of $SRC_DIR..."
tar -czvf $BACKUP_FILE $SRC_DIR

# Check if backup was successful
if [ $? -eq 0 ]; then
  echo "Backup successful! File: $BACKUP_FILE"
else
  echo "Backup FAILED."
fi

# Optional: Remove backups older than 7 days
find $DEST_DIR -name "web_backup_*.tar.gz" -mtime +7 -exec rm {} \;
echo "Old backups cleaned up."
```

To run it:

1. **chmod +x backup.sh**
2. **./backup.sh**

2. Git Version Control

Git is your code time machine. It tracks every change, enables team collaboration, and lets you undo mistakes.

Basic Git Commands

1. **git init** - Initializes a new Git repository.
2. **git clone <url>** - Copies a remote repository.
3. **git status** - Displays the state of the working directory.
4. **git add <file>** - Adds changes to the staging area.
   - **git add .** - Stage all changes.
5. **git commit -m "Message"** - Records changes to the repository.
6. **git config** - Configures user settings.
   - **git config --global user.name "Your Name"**
   - **git config --global user.email "you@example.com"**

Git Log and Diff

7. **git log** - Shows the commit history.
   - **git log --oneline --graph --decorate** - A cleaner, graphical view.
8. **git show <commit-hash>** - Displays details about a specific commit.
9. **git diff** - Shows changes between commits, staging, and working directory.
   - **git diff** - Changes in working dir vs. staging area.

    - **git diff --staged** - Changes in staging area vs. last commit.
10. **git reset** - Unstages changes or resets commits.
    - **git reset HEAD <file>** - Unstage a file.
    - **git reset --hard <commit-hash>** - **DANGEROUS:** Discard all changes back to a specific commit.

Branching and Merging

11. **git branch** - Lists branches or creates a new branch.
    - **git branch <branch-name>** - Create a new branch.
    - **git branch -d <branch-name>** - Delete a branch.
12. **git checkout <branch-name>** - Switches to a branch.
    - **git checkout -b <branch-name>** - Create a new branch and switch to it.
13. **git switch <branch-name>** - (Modern) Switches branches.
14. **git merge <branch-name>** - Combines changes from one branch into another.
15. **git rebase <branch-name>** - Moves or combines commits from one branch onto another.
16. **git cherry-pick <commit-hash>** - Applies specific commits from one branch to another.

Remote Repositories

17. **git remote** - Manages remote repository connections.
    - **git remote add <name> <url>** (e.g., git remote add origin ...)
18. **git push <remote> <branch>** - Sends changes to a remote repository.
    - **git push -u origin <branch-name>** - Push and set upstream.
19. **git pull <remote> <branch>** - Fetches and merges changes from a remote.
20. **git fetch <remote>** - Downloads changes from a remote without merging.
21. **git remote -v** - Lists the URLs of remote repositories.

Stashing and Cleaning

22. **git stash** - Temporarily saves changes not yet committed.
23. **git stash pop** - Applies stashed changes and removes them from the stash list.
24. **git stash list** - Lists all stashes.
25. **git clean -f** - Removes untracked files from the working directory.
    - **git clean -fd** - Also remove untracked directories.

Tagging

26. **git tag -a <tag-name> -m "Message"** - Creates an annotated tag.
27. **git tag -d <tag-name>** - Deletes a tag.
28. **git push --tags** - Pushes tags to a remote repository.

Advanced Git Commands

29. **git bisect** - Finds the commit that introduced a bug.
30. **git blame <file>** - Shows which commit and author modified each line.

31. **git reflog** - Shows a log of changes to HEAD (good for recovering lost commits).
32. **git submodule** - Manages external repositories as submodules.
33. **git archive** - Creates an archive of the repository files.
34. **git gc** - Cleans up unnecessary files and optimizes the repository.

GitHub-Specific Commands (using gh CLI)

35. **gh auth login** - Logs into GitHub via the command line.
36. **gh repo clone <user/repo>** - Clones a GitHub repository.
37. **gh issue list** - Lists issues in a GitHub repository.
38. **gh pr create** - Creates a pull request on GitHub.
39. **gh repo create** - Creates a new GitHub repository.

Git Playbook Workflows

Workflow 1: Starting a New Feature (GitHub Flow)

1. **Sync your main branch:**
   ```
   git switch main
   git pull origin main
   ```
2. **Create your feature branch:**
   ```
   git switch -b feature/my-new-thing
   ```
3. **Do your work (edit files, etc.).**
4. **Add and commit your changes:**
   ```
   git add .
   git commit -m "Feat: Add component for my-new-thing"
   ```
5. **Push your branch to the remote:**
   ```
   git push -u origin feature/my-new-thing
   ```
6. **Create a Pull Request:** Go to GitHub to open a PR from your branch into main.

Workflow 2: Undoing a Mistake

- **Case A: You just committed, but want to change the message.**
  ```
  git commit --amend -m "A better commit message"
  ```
- **Case B: You want to add more files to the last commit.**
  ```
  git add new-file.txt
  git commit --amend --no-edit
  ```
- **Case C: You want to completely undo the last commit (and keep changes).**
  ```
  git reset --soft HEAD~1
  # Your files are unchanged, commit is undone.
  ```

- **Case D: You want to undo a commit that is already public (pushed).**
    - **Do not use git reset!** Use git revert.

```
git revert <commit-hash-to-undo>
# This creates a *new* commit that is the inverse of the bad one.
git push
```

3. Docker Containerization

Docker packages applications into portable containers. These commands help build, ship, and run applications consistently.

Basic Docker Commands

1. **docker --version** - Displays the installed Docker version.
2. **docker info** - Shows system-wide information about Docker.
3. **docker pull <image:tag>** - Downloads an image.
4. **docker images** - Lists all downloaded images.
5. **docker run <image>** - Creates and starts a new container.
    - **docker run -it <image> bash** - Run interactively with a shell.
    - **docker run -d -p 8080:80 <image>** - Run detached, mapping port 8080 (host) to 80 (container).
6. **docker ps** - Lists running containers.
7. **docker ps -a** - Lists all containers (running and stopped).
8. **docker stop <container_id_or_name>** - Stops a running container.
9. **docker start <container_id_or_name>** - Starts a stopped container.
10. **docker rm <container_id_or_name>** - Removes a container.
    - **docker rm $(docker ps -aq)** - Remove all stopped containers.
11. **docker rmi <image_id_or_name>** - Removes an image.
12. **docker exec -it <container> <command>** - Runs a command inside a running container.

Intermediate Docker Commands

13. **docker build -t <image_name:tag> .** - Builds an image from a Dockerfile.
14. **docker commit <container> <new_image:tag>** - Creates an image from a container's changes.
15. **docker logs <container>** - Fetches logs from a container.
    - **docker logs -f <container>** - Follow logs.
16. **docker inspect <container_or_image>** - Returns detailed information.
17. **docker stats** - Displays live resource usage statistics.
18. **docker cp <src_path> <container>:<dest_path>** - Copies files.
    - **docker cp <container>:<src_path> <dest_path>**
19. **docker rename <old_name> <new_name>** - Renames a container.
20. **docker network ls** - Lists all Docker networks.

21. **docker network create <network_name>** - Creates a new network.
22. **docker network inspect <network_name>** - Shows details about a network.
23. **docker network connect <network> <container>** - Connects a container to a network.
24. **docker volume ls** - Lists all Docker volumes.
25. **docker volume create <volume_name>** - Creates a new volume.
26. **docker volume inspect <volume_name>** - Provides details about a volume.
27. **docker volume rm <volume_name>** - Removes a volume.

Advanced Docker Commands

28. **docker-compose up** - Starts services defined in docker-compose.yml.
    - **docker-compose up -d** - Run in detached mode.
29. **docker-compose down** - Stops and removes services.
    - **docker-compose down -v** - Also remove volumes.
30. **docker-compose logs** - Displays logs for services.
31. **docker-compose exec <service_name> <command>** - Runs a command in a service.
32. **docker save -o <file.tar> <image>** - Exports an image to a tar file.
33. **docker load -i <file.tar>** - Imports an image from a tar file.
34. **docker export <container> > <container.tar>** - Exports a container's filesystem.
35. **docker import <container.tar> <new_image>** - Creates an image from an exported container.
36. **docker system df** - Displays disk usage by Docker.
37. **docker system prune** - Cleans up unused resources.
    - **docker system prune -af** - Prune all (images, containers, volumes) without prompting.
38. **docker tag <old_image:tag> <new_image:tag>** - Assigns a new tag to an image.
39. **docker push <image:tag>** - Uploads an image to a Docker registry.
40. **docker login** - Logs into a Docker registry.
41. **docker logout** - Logs out of a Docker registry.
42. **docker swarm init** - Initializes a Docker Swarm.
43. **docker service create** - Creates a new service in Swarm.
44. **docker stack deploy -c <compose.yml> <stack_name>** - Deploys a stack.
45. **docker stack rm <stack_name>** - Removes a stack.
46. **docker checkpoint create <container> <checkpoint>** - Creates a checkpoint.
47. **docker checkpoint ls <container>** - Lists checkpoints.
48. **docker checkpoint rm <container> <checkpoint>** - Removes a checkpoint.

Docker Playbook Examples

Example 1: Dockerfile for a simple Node.js App

```
# Use an official Node.js runtime as a parent image
FROM node:18-alpine

# Set the working directory in the container
```

```
WORKDIR /usr/src/app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install any needed dependencies
RUN npm install

# Copy the rest of the application's source code
COPY . .

# Make port 3000 available to the world outside this container
EXPOSE 3000

# Define the command to run the app
CMD [ "node", "server.js" ]
```

Example 2: docker-compose.yml for a Web App + Database

This example starts a WordPress site and a MySQL database, connecting them with a network and persisting data with volumes.

```
version: '3.8'

services:
  # WordPress Service
  wordpress:
    image: wordpress:latest
    ports:
      - "8000:80"  # Map host port 8000 to container port 80
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: somepassword
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wordpress_data:/var/www/html  # Persist WordPress files
    networks:
      - app_network
    depends_on:
      - db
  # MySQL Database Service
  db:
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: somepassword
      MYSQL_ROOT_PASSWORD: rootpassword
    volumes:
      - db_data:/var/lib/mysql  # Persist database data
    networks:
      - app_network

# Define networks
networks:
  app_network:
    driver: bridge

# Define volumes
volumes:
  wordpress_data:
  db_data:
```

4. Kubernetes (K8s) Orchestration

Kubernetes automates deployment, scaling, and management of containerized applications.

Basic Kubernetes Commands (kubectl)

1. **kubectl version** - Displays Kubernetes client/server version.
2. **kubectl cluster-info** - Shows cluster information.
3. **kubectl get nodes** - Lists all nodes in the cluster.
4. **kubectl get pods** - Lists all pods in the default namespace.
   - **kubectl get pods -n <namespace>** - List pods in a specific namespace.
   - **kubectl get pods -A** - List pods in *all* namespaces.
   - **kubectl get pods -o wide** - Get more details (IP, node).
5. **kubectl get services** - Lists all services.
6. **kubectl get namespaces** - Lists all namespaces.
7. **kubectl describe pod <pod-name>** - Shows detailed information about a pod.
8. **kubectl logs <pod-name>** - Displays logs for a pod.
   - **kubectl logs -f <pod-name>** - Follow logs.

9. **kubectl create namespace <name>** - Creates a new namespace.
10. **kubectl delete pod <pod-name>** - Deletes a pod.

Intermediate Kubernetes Commands

11. **kubectl apply -f <file.yaml>** - Applies changes from a YAML file.
12. **kubectl delete -f <file.yaml>** - Deletes resources from a YAML file.
13. **kubectl scale deployment <name> --replicas=3** - Scales a deployment.
14. **kubectl expose deployment <name> --type=LoadBalancer --port=80** - Exposes a deployment.
15. **kubectl exec -it <pod-name> -- /bin/bash** - Executes a command in a pod.
16. **kubectl port-forward <pod-name> 8080:80** - Forwards a local port to a pod.
17. **kubectl get configmaps** - Lists all ConfigMaps.
18. **kubectl get secrets** - Lists all Secrets.
19. **kubectl edit <resource>/<name>** - Edits a resource definition.
20. **kubectl rollout status deployment/<name>** - Displays deployment rollout status.

Advanced Kubernetes Commands

21. **kubectl rollout undo deployment/<name>** - Rolls back a deployment.
22. **kubectl top nodes** - Shows resource usage for nodes.
23. **kubectl top pods** - Displays resource usage for pods.
24. **kubectl cordon <node-name>** - Marks a node as unschedulable.
25. **kubectl uncordon <node-name>** - Marks a node as schedulable.
26. **kubectl drain <node-name> --ignore-daemonsets** - Safely evicts all pods from a node.
27. **kubectl taint nodes <node-name> <key>=<value>:<effect>** - Adds a taint to a node.
28. **kubectl get events** - Lists all events.
29. **kubectl apply -k <dir>** - Applies resources from a kustomization directory.
30. **kubectl config view** - Displays the kubeconfig file.
31. **kubectl config use-context <cluster-name>** - Switches the active context.
32. **kubectl debug pod/<pod-name>** - Creates a debugging session for a pod.
33. **kubectl delete namespace <name>** - Deletes a namespace.
34. **kubectl patch <resource> <name> -p '{"spec": ...}'** - Updates a resource.
35. **kubectl rollout history deployment/<name>** - Shows deployment rollout history.
36. **kubectl autoscale deployment <name> --cpu-percent=50 --min=1 --max=10** - Creates a HorizontalPodAutoscaler.
37. **kubectl label pod <pod-name> <key>=<value>** - Adds or modifies a label.
38. **kubectl annotate pod <pod-name> <key>=<value>** - Adds or modifies an annotation.
39. **kubectl delete pv <pv-name>** - Deletes a PersistentVolume.
40. **kubectl get ingress** - Lists all Ingress resources.
41. **kubectl create configmap <name> --from-literal=<key>=<value>** - Creates a ConfigMap.
42. **kubectl create secret generic <name> --from-literal=<key>=<value>** - Creates a Secret.

43. **kubectl api-resources** - Lists all available API resources.
44. **kubectl api-versions** - Lists all API versions.
45. **kubectl get crds** - Lists all Custom Resource Definitions (CRDs).

Kubernetes Playbook Examples (YAML Manifests)

Example 1: deployment.yaml

This manifest creates a Deployment that runs 3 replicas of the Nginx container.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.23
        ports:
        - containerPort: 80
```

To apply: kubectl apply -f deployment.yaml

Example 2: service.yaml

This manifest creates a Service of type NodePort to expose the nginx-deployment outside the cluster.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  selector:
    app: nginx  # This MUST match the labels in the Deployment's template
  ports:
    - protocol: TCP
      port: 80       # Port the service is available on *inside* the cluster
      targetPort: 80 # Port the container is listening on
      # nodePort: 30080 # Optional: specify a port (30000-32767)
```

To apply: kubectl apply -f service.yaml

5. Helm (The K8s Package Manager)

Helm simplifies installing and managing complex Kubernetes applications using "charts".

Basic Helm Commands

1. **helm help** - Displays help.
2. **helm version** - Shows Helm client/server version.
3. **helm repo add <name> <url>** - Adds a new chart repository.
4. **helm repo update** - Updates all chart repositories.
5. **helm repo list** - Lists all added repositories.
6. **helm search hub <keyword>** - Searches for charts on Helm Hub.
7. **helm search repo <keyword>** - Searches charts in your repositories.
8. **helm show chart <repo/chart>** - Displays information about a chart.

Installing and Upgrading Charts

9. **helm install <release-name> <repo/chart>** - Installs a chart.
10. **helm upgrade <release-name> <repo/chart>** - Upgrades an existing release.
11. **helm upgrade --install <release-name> <repo/chart>** - Installs or upgrades.
12. **helm uninstall <release-name>** - Uninstalls a release.
13. **helm list** - Lists all installed releases.
    - **helm list -n <namespace>** or **helm list -A**
14. **helm status <release-name>** - Displays the status of a release.

Working with Helm Charts

15. **helm create <chart-name>** - Creates a new Helm chart directory.
16. **helm lint ./<chart-name>** - Lints a chart for errors.
17. **helm package ./<chart-name>** - Packages a chart into a .tgz file.
18. **helm template <release-name> ./<chart-name>** - Renders YAML files without installing.
19. **helm dependency update ./<chart-name>** - Updates dependencies.

Advanced Helm Commands

20. **helm rollback <release-name> <revision_number>** - Rolls back a release.
21. **helm history <release-name>** - Displays the history of a release.
22. **helm get all <release-name>** - Gets all information for a release.
23. **helm get values <release-name>** - Displays the values used in a release.
24. **helm test <release-name>** - Runs tests defined in a chart.

Helm Chart Repositories

25. **helm repo remove <name>** - Removes a chart repository.
26. **helm repo update** - (Duplicate) Updates local cache.
27. **helm repo index <dir>** - Creates an index file for a chart repository.

Helm Values and Customization

28. **helm install <name> <chart> --values <values.yaml>** - Installs with custom values.
29. **helm upgrade <name> <chart> -f <values.yaml>** - Upgrades with custom values.
30. **helm install <name> <chart> --set <key>=<value>** - Installs with a direct value.
31. **helm upgrade <name> <chart> --set <key>=<value>** - Upgrades with a direct value.

Helm Template and Debugging

32. **helm uninstall <release-name> --purge** - (Note: --purge is deprecated in Helm 3, uninstall does this by default).
33. **helm template <name> <chart> --debug** - Renders templates with debug output.
34. **helm install <name> <chart> --dry-run** - Simulates an install.
35. **helm upgrade <name> <chart> --dry-run** - Simulates an upgrade.

Helm and Kubernetes Integration

36. **helm list --namespace <ns>** - Lists releases in a namespace.
37. **helm uninstall <name> --namespace <ns>** - Uninstalls from a namespace.
38. **helm install <name> <chart> --namespace <ns>** - Installs into a namespace.
39. **helm upgrade <name> <chart> --namespace <ns>** - Upgrades in a namespace.

Helm Chart Development

40. **helm package --sign** - Packages and signs a chart.
41. **helm create --starter <path>** - Creates a chart from a starter template.
42. **helm push <chart.tgz> <repo_name>** - Pushes a chart to a repository.

Helm with Kubernetes CLI

43. **helm list -n <namespace>** - (Duplicate) Lists releases in a namespace.
44. **helm install <name> <chart> --kube-context <context>** - Installs to a specific cluster context.
45. **helm upgrade <name> <chart> --kube-context <context>** - Upgrades in a specific

context.

Helm Chart Dependencies

46. **helm dependency build ./<chart-name>** - Builds dependencies.
47. **helm dependency list ./<chart-name>** - Lists all dependencies.

Helm History and Rollbacks

48. **helm rollback <name> <revision> --recreate-pods** - Rolls back and recreates pods.
49. **helm history <name> --max <number>** - Limits history output.

Helm Playbook Workflow

Workflow: Install a Customized Prometheus Stack

1. **Add the Prometheus community repository:**
   ```
   helm repo add prometheus-community [https://prometheus-community.github.io/helm-charts](https://prometheus-community.github.io/helm-charts)
   helm repo update
   ```

2. **Search for the kube-prometheus-stack chart:**
   ```
   helm search repo prometheus-community/kube-prometheus-stack
   ```

3. **Get the default values and save them to a file:**
   ```
   helm show values prometheus-community/kube-prometheus-stack > prom-values.yaml
   ```

4. **Edit the prom-values.yaml file:**
   - For example, you might want to disable Grafana or set persistence.
   - **nano prom-values.yaml**
   - Find grafana: and set enabled: false
5. **Install the chart into a monitoring namespace:**
   ```
   kubectl create namespace monitoring
   helm install my-prometheus prometheus-community/kube-prometheus-stack \
     -n monitoring \
     -f prom-values.yaml
   ```

6. **Check the status:**
   ```
   helm status my-prometheus -n monitoring
   kubectl get pods -n monitoring
   ```

7. **Uninstall the release:**
   ```
   helm uninstall my-prometheus -n monitoring
   ```

6. Terraform (Infrastructure as Code)

Terraform lets you build, change, and version cloud and on-prem infrastructure safely and efficiently.

Basic Terraform Commands

50. **terraform --help** - Displays general help.
51. **terraform init** - Initializes the working directory (downloads providers).
52. **terraform validate** - Validates configuration files syntax.
53. **terraform plan** - Creates an execution plan.
54. **terraform apply** - Applies the changes.
    - **terraform apply -auto-approve** - Apply without interactive approval.
55. **terraform show** - Displays the current state.
56. **terraform output** - Displays output values.
57. **terraform destroy** - Destroys the infrastructure.
58. **terraform refresh** - Updates state file with real infrastructure.
59. **terraform taint <resource_address>** - Marks a resource for recreation.
60. **terraform untaint <resource_address>** - Unmarks a tainted resource.
61. **terraform state** - Manages state files.
62. **terraform import <resource_address> <resource_id>** - Imports existing infrastructure.
63. **terraform graph** - Generates a graphical representation.
64. **terraform providers** - Lists providers.
65. **terraform state list** - Lists all resources in the state.
66. **terraform backend** - Configures the state backend.
67. **terraform state mv <source> <destination>** - Moves an item in the state.
68. **terraform state rm <resource_address>** - Removes an item from the state.
69. **terraform workspace** - Manages workspaces.
70. **terraform workspace new <name>** - Creates a new workspace.
71. **terraform module** - Manages modules.
72. **terraform init -get-plugins=true** - (Note: This is default behavior in modern Terraform).
73. **TF_LOG=DEBUG** - Set log level via environment variable.
74. **TF_LOG_PATH=<path>** - Set log file path.
75. **terraform login** - Logs into Terraform Cloud/Enterprise.
76. **terraform remote** - (Legacy) Manages remote state.
77. **terraform push** - (Legacy) Pushes modules.

Terraform Playbook Example (HCL)

This example defines an AWS S3 bucket.

File: main.tf

```
# 1. Configure the AWS Provider
```

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
}

# 2. Define a variable for the bucket name
variable "bucket_name" {
  description = "The name for the S3 bucket"
  type        = string
  default     = "my-unique-tf-playbook-bucket-12345"
}

# 3. Create the S3 bucket resource
resource "aws_s3_bucket" "my_bucket" {
  bucket = var.bucket_name

  tags = {
    Name        = "My Terraform Bucket"
    Environment = "Dev"
  }
}

# 4. Output the bucket name
output "bucket_name" {
  value = aws_s3_bucket.my_bucket.bucket
}
```

Terraform Workflow:

1. **terraform init** - Initializes and downloads the AWS provider.
2. **terraform plan** - Shows that it will create 1 S3 bucket.
3. **terraform apply** - Prompts for approval, then creates the bucket.
4. **terraform output** - Displays the name of the created bucket.
5. **terraform destroy** - Prompts for approval, then deletes the bucket.

7. Ansible (Configuration Management) - *NEW SECTION*

Ansible is an open-source tool that automates software provisioning, configuration management, and application deployment. It is **agentless**, meaning it connects to servers over SSH.

Core Concepts

- **Inventory:** A file (like hosts.ini) that lists the servers Ansible manages.
- **Playbook:** A YAML file that defines a set of tasks to be executed on a server.
- **Task:** A single action to be performed (e.g., install a package, copy a file).
- **Module:** The code that Ansible runs for a task (e.g., apt, copy, systemd).
- **Role:** A collection of playbooks, templates, and variables to organize complex configurations.

Common Commands

- **ansible --version** - Check version.
- **ansible all -m ping -i inventory.ini** - Ping all hosts in the inventory (ad-hoc command).
- **ansible-playbook -i inventory.ini playbook.yml** - Run a playbook.
- **ansible-playbook -i inventory.ini playbook.yml --check** - Dry-run: see what would change.
- **ansible-galaxy install <role_name>** - Install a role from Ansible Galaxy.

Ansible Playbook Example (YAML)

This playbook installs and starts Nginx on a group of web servers.

File: inventory.ini

```
[webservers]
web1.example.com
web2.example.com
```

File: playbook.yml

```
---
- name: Configure Web Servers
  hosts: webservers  # This matches the group in the inventory
  become: yes      # This means "run as sudo"
  tasks:
    - name: Install nginx (Debian/Ubuntu)
      ansible.builtin.apt:
        name: nginx
        state: present
        update_cache: yes
      when: ansible_os_family == "Debian"

    - name: Install nginx (RHEL/CentOS)
      ansible.builtin.yum:
        name: nginx
        state: present
      when: ansible_os_family == "RedHat"

    - name: Start and enable nginx service
      ansible.builtin.systemd:
        name: nginx
        state: started
        enabled: yes

    - name: Copy custom index.html page
      ansible.builtin.template:
        src: index.html.j2  # A template file
        dest: /var/www/html/index.html
        mode: '0644'
```

File: index.html.j2 (This is a Jinja2 template)

```
<html>
<head><title>Welcome</title></head>
<body>
  <h1>This server is {{ ansible_hostname }}</h1>
  <p>Managed by Ansible.</p>
</body>
</html>
```

To run: ansible-playbook -i inventory.ini playbook.yml

8. CI/CD (Continuous Integration/Deployment) - *NEW SECTION*

CI/CD is a practice that automates the software build, test, and deployment pipeline. GitHub Actions is a popular tool built directly into GitHub.

Core Concepts (Using GitHub Actions)

- **Workflow:** An automated process defined in a YAML file in the .github/workflows/ directory.
- **Event:** The trigger for a workflow (e.g., on: push, on: pull_request).
- **Job:** A set of steps that run on a runner.
- **Step:** A single task (either a shell command or a pre-built action).
- **Action:** A reusable piece of code (e.g., actions/checkout@v3).
- **Runner:** The server (Linux, Windows, macOS) that executes the job.

CI/CD Playbook Example (GitHub Actions)

This workflow triggers on a push to the main branch. It builds a Node.js app, runs tests, builds a Docker image, and pushes it to Docker Hub.

File: .github/workflows/main.yml

```
name: CI/CD Pipeline

# 1. Trigger the workflow on push to the 'main' branch
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  # 2. Job to build and test the application
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - name: Check out repository code
        uses: actions/checkout@v4

      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'

      - name: Install dependencies
        run: npm install

      - name: Run tests
        run: npm test
```

```
 # 3. Job to build and push the Docker image
 build-and-push-docker:
   needs: build-and-test  # This job only runs if 'build-and-test' succeeds
   runs-on: ubuntu-latest
   if: github.ref == 'refs/heads/main' # Only run on push to main, not PRs
   steps:
     - name: Check out repository code
       uses: actions/checkout@v4

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

     - name: Build and push Docker image
       uses: docker/build-push-action@v5
       with:
         context: .
         file: ./Dockerfile
         push: true
         tags: your-dockerhub-username/my-app:latest
```

To use:

1. Add this file to your repository at .github/workflows/main.yml.
2. Add DOCKER_USERNAME and DOCKER_PASSWORD to your GitHub repository's **Settings > Secrets and variables > Actions** secrets.

9. Monitoring & Observability - *NEW SECTION*

- **Monitoring:** Tells you *if* something is wrong (e.g., CPU is at 90%).
- **Observability:** Tells you *why* something is wrong (e.g., a specific function is in a loop).

Core Tools

- **Prometheus:** A time-series database that *pulls* (scrapes) metrics from your applications.
- **Grafana:** A visualization tool that queries Prometheus (and other sources) to create dashboards.
- **ELK/EFK Stack:**
  - **E**lasticsearch: A database for storing logs.
  - **L**ogstash / **F**luentd: Tools for collecting and processing logs.
  - **K**ibana: A visualization tool for logs.

Prometheus & PromQL

Prometheus uses a powerful query language called PromQL.

Common PromQL Queries:

- **Get the per-second rate of HTTP requests over the last 5 minutes:**
  rate(http_requests_total[5m])

- **Get the 95th percentile request latency:**
  histogram_quantile(0.95, sum(rate(http_request_duration_seconds_bucket[5m])) by (le))

- **Show how many instances of each job are "up" (running):**
  sum(up) by (job)

- **Get free memory on nodes (using Node Exporter):**
  node_memory_MemFree_bytes

Grafana

Grafana is the visualization layer. You don't "run commands" in it, but you use it to:

1. Add Prometheus as a data source.
2. Create dashboards with panels.
3. Write PromQL queries in the panels to build graphs.
4. Set up alerts based on query thresholds.

Logging (ELK/EFK)

The goal is to centralize logs.

1. **Fluentd/Logstash** runs on your nodes (or as a sidecar in K8s).
2. It collects logs from files (/var/log/*.log) or container output.
3. It parses, enriches, and forwards these logs to **Elasticsearch**.
4. **Kibana** provides a web UI to search and visualize all your logs in one place.

Thank you. Lets Connect
www.linkedin.com/in/bhooshan-pattanashetti