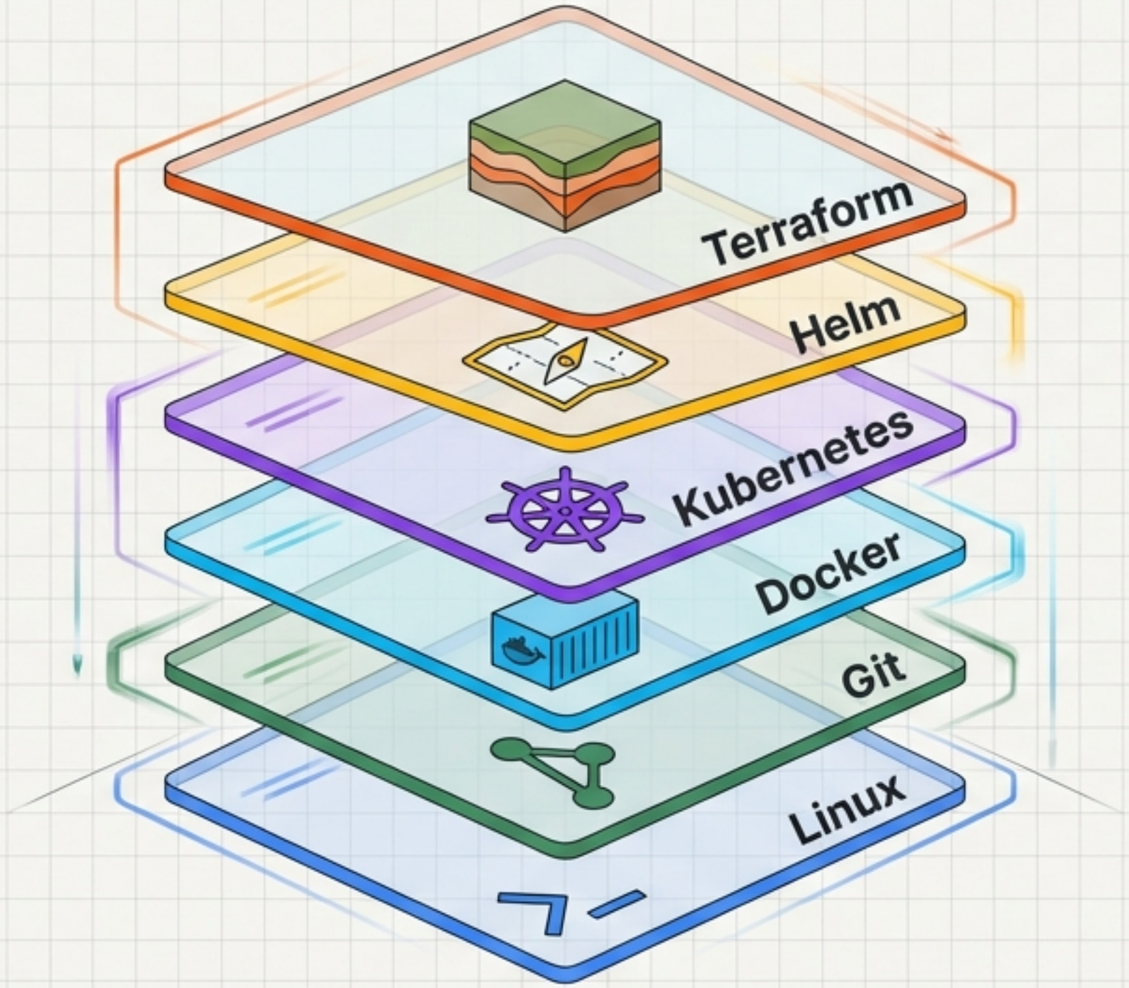
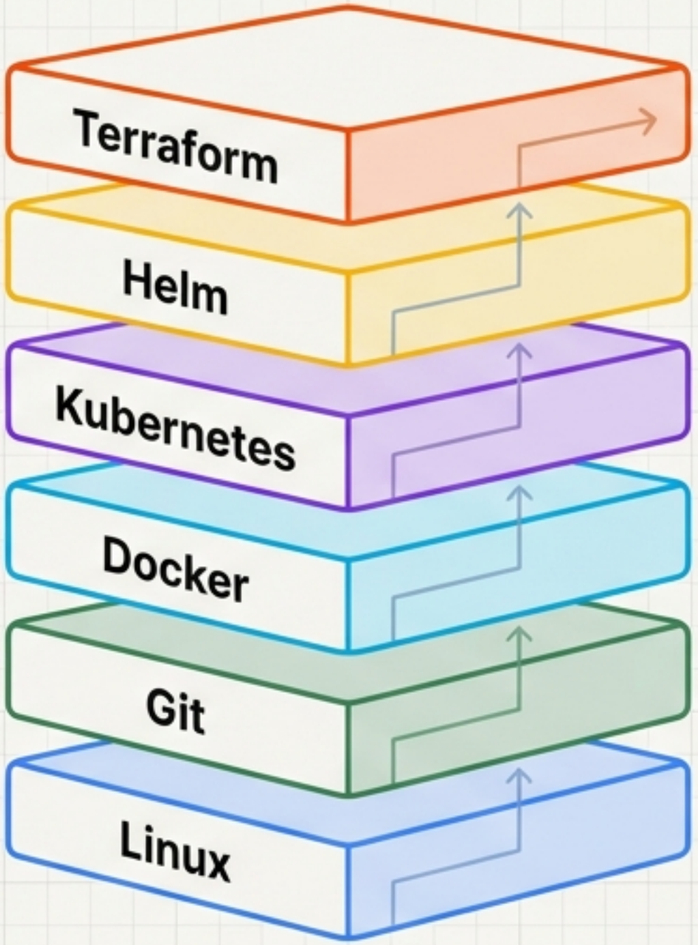
Building the Modern Stack, Command by Command

A DevOps Journey from the Command Line to the Cloud



The DevOps Toolchain: An Architectural Blueprint



Terraform: The Cloud Layer - Defining and provisioning infrastructure with code.

Helm: The Packaging Layer - Simplifying the deployment and management of Kubernetes applications.

Kubernetes: The Orchestration Layer - Conducting containerized applications at scale.

Docker: The Encapsulation Layer - Packaging software into universal, portable containers.

Git: The Versioning Layer - Tracking every change to our code and configuration.

Linux: The Foundation Layer - The operating system where everything runs.

Layer 1: The Foundation

Linux

Linux is the foundation of DevOps operations - it's like a Swiss Army knife for servers. These commands help you navigate systems, manage files, configure permissions, and automate tasks.

Linux Essentials: Filesystem & Navigation

Navigation & Inspection

`pwd`
Print the current working directory.

`ls`
List files and directories.

`cd`
Change directory.

`cat`
Display the content of a file.

`echo`
Display a line of text.

`clear`
Clear the terminal screen.

File & Directory Manipulation

`touch`
Create an empty file.

`mkdir`
Create a new directory.

`cp`
Copy files or directories.

`mv`
Move or rename files and directories.

`rm`
Remove files or directories.

`rmdir`
Remove empty directories.

Linux Power Tools: System & Process Management

Permissions & Search

`chmod`
Change file permissions.

`chown`
Change file ownership.

`find`
Search for files and directories.

`grep`
Search for text in a file.

System Monitoring

`ps`
Display active processes.

`top`
Monitor system processes in real time.

`df -h`
Display disk usage in human-readable format.

`du`
Display directory size.

Networking & Transfer

`ping`
Check network connectivity.

`ssh`
Connect to a remote server via SSH.

`scp`
Securely copy files between systems.

`wget`
Download files from the internet.

Layer 2: The Blueprint

Git is your code's time machine. It tracks every change, enables team collaboration without conflicts, and lets you undo mistakes.

Git Core: Managing Your Code's History

Starting a Project

`git init`

Initializes a new Git repository in the current directory.

```
git init
```

`git clone`

Copies a remote repository to the local machine.

```
git clone
https://github.com/user/repo.git
```

The Commit Cycle

`git status`

Displays the state of the working directory and staging area.

`git add`

Adds changes to the staging area.

```
git add file.txt
```

`git commit`

Records changes to the repository.

```
git commit -m "Initial commit"
```

Inspecting History

`git log`

Shows the commit history.

`git diff`

Shows changes between commits, the working directory, etc.

Git Collaboration: Branching and Remotes

Branching & Merging (Working in Parallel)

`git branch`

Lists branches or creates a new branch.

```
git branch feature-branch
```

`git checkout`

Switches between branches or restores files.

```
git checkout feature-branch
```

`git merge`

Combines changes from one branch into another.

```
git merge feature-branch
```

Remote Repositories (Sharing & Syncing)

`git remote`

Manages remote repository connections.

```
git remote add origin ...
```

`git push`

Sends changes to a remote repository.

```
git push origin main
```

`git pull`

Fetches and merges changes from a remote repository.

`git fetch`

Downloads changes from a remote repository without merging.

Layer 3: The Container

Docker

Docker packages applications into portable containers - like shipping containers for software. Build, ship, and run applications consistently across any environment.

Docker Lifecycle: Building & Running Containers

Image Management

`docker pull`

Downloads an image from a Docker registry.

```
docker pull ubuntu:latest
```

`docker images`

Lists all downloaded images.

`docker build`

Builds an image from a Dockerfile.

```
docker build -t my_image .
```

`docker push`

Uploads an image to a Docker registry.

Container Lifecycle

`docker run`

Creates and starts a new container from an image.

```
docker run -it ubuntu bash
```

`docker ps`

Lists running containers. (Use `` `ps -a` `` for all containers).

`docker stop`

Stops a running container.

`docker rm`

Removes a container.

Interaction & Debugging

`docker exec`

Runs a command inside a running container.

```
docker exec -it <container_name> bash
```

`docker logs`

Fetches logs from a container.

`docker inspect`

Returns detailed information on a container or image.

Layer 4: The Orchestra

Kubernetes

Kubernetes is the conductor of your container orchestra. It automates deployment, scaling, and management of containerized applications across server clusters.

Kubernetes Inspection: Understanding Your Cluster

Cluster & Node Info

`kubectl cluster-info`

Shows information about the Kubernetes cluster.

`kubectl get nodes`

Lists all nodes in the cluster.

`kubectl top nodes`

Shows resource usage for nodes.

Application Workloads

`kubectl get pods`

Lists all pods in the default namespace.

`kubectl get services`

Lists all services in the default namespace.

`kubectl get deployments`

Lists all deployments.

Deep Dive & Debug

`kubectl describe pod [pod-name]`

Shows detailed information about a specific pod.

`kubectl logs [pod-name]`

Displays logs for a specific pod.

Kubernetes Management: Applying Change

Declarative Management (The "Right" Way)

`kubectl apply -f [file.yaml]`

Applies changes defined in a YAML file.

`kubectl delete -f [file.yaml]`

Deletes resources defined in a YAML file.

Imperative Commands (For Quick Changes & Debugging)

`kubectl scale deployment [name] --replicas=3`

Scales a deployment to the desired number of replicas.

`kubectl expose deployment [name] --port=80`

Exposes a pod or deployment as a service.

`kubectl port-forward [pod-name] 8080:80`

Forwards a local port to a port on a pod.

`kubectl exec -it [pod-name] -- /bin/bash`

Executes a command in a running pod.

The Highest Abstractions: Helm & Terraform

Helm - The Kubernetes Package Manager

Think of it like apt-get for Kubernetes. Helm simplifies installing and managing complex applications using pre-packaged "charts".

Core Commands (A clean list)

`helm repo add / update` - Manage chart repositories.

`helm install [release] [chart]` - Installs a chart into a Kubernetes cluster.

`helm upgrade [release] [chart]` - Upgrades an existing release.

`helm list` - Lists all releases in the cluster.

`helm rollback [release] [revision]` Rolls back a release to a previous version.

Terraform – Infrastructure as Code

Instead of clicking buttons in a cloud console, Terraform lets you build, change, and version your entire infrastructure using configuration files.

Core Workflow Commands (A clean list)

`terraform init` - Initializes the working directory and downloads providers.

`terraform plan` - Creates an execution plan to preview changes.

`terraform apply` - Applies the changes required to reach the desired state.

`terraform destroy` - Destroys the infrastructure defined in the configuration.

The Complete Stack: From Foundation to Cloud

From a single command on a Linux server to deploying and managing a complex, orchestrated, infrastructure-defined application. Each layer builds upon the last, solving the challenges of scale and complexity. This is the blueprint of modern DevOps.