THE 'DAY 0' DEVOPS GUIDE

Master the basics before jumping into tools

Govardhana Miriyala Kannaiah

I'm Govardhana Miriyala Kannaiah, a DevOps and Multi-Cloud Architect with over 17 years of IT experience.

Over the past year, I've built a LinkedIn community of 88k+ followers and launched NeuVeu, a consulting firm that has served 16+ clients—from enterprises like Hearst and Stanford University to small-scale companies like MapDigital.

Throughout my journey, I've noticed that many people believe DevOps is all about tools, but there's a crucial layer beneath that.

I call this '**DevOps Day 0**'—the foundational aspects that answer **why** we do things, not just how.

This guide focuses on those key fundamentals to set you up for long-term DevOps success.

https://www.techopsexamples.com/

Table of Contents

Introduction

This guide is designed to provide the fundamentals needed for a DevOps role, basically a pre-state of DevOps. Whether you are just starting out or looking to enhance your existing skills, this guide will provide a clear and structured path to set you up for long-term success.

Target Audience

This guide is for:

- **Beginners** who want to understand the needed fundamentals before getting into DevOps tools.
- **Experienced individuals** looking to improve their skills and fill any gaps in their knowledge.

Resources

To discover real-world use cases, tech updates, and learning resources, check out:

- **Full Editions:** https://www.techopsexamples.com/

Linux Fundamentals

Linux is the backbone of most servers you’ll work with as a DevOps engineer. Whether setting up infrastructure or managing applications, Linux will be at the core. Knowing your way around the command line, especially with Bash, is crucial for handling tasks efficiently and automating processes.

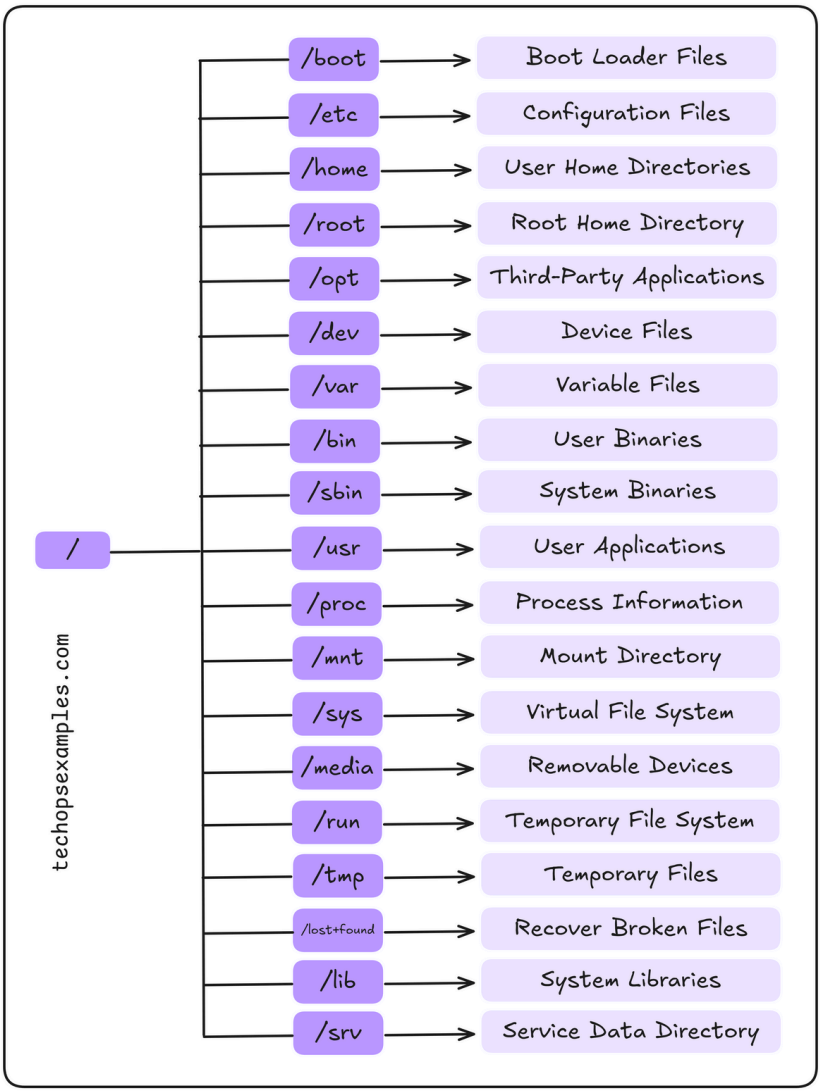
Key Concepts:

- Basic Linux commands (ls, cp, mv, rm, etc.)
- File system hierarchy (/, /home, /etc, /var, etc.)
- Permissions and ownership (chmod, chown etc.)
- Processes and signals (ps, top, kill etc.)
- Package management (apt, yum)
- Networking commands (ifconfig, ping, netstat, curl, wget etc.)
- Disk management (df, du, fdisk, mount, umount etc.)
- User and group management (adduser, deluser, usermod, groupadd)

Basic Linux Commands

Command /Concept	Description	Typically used when
ls	Lists files and directories in the current directory.	Checking available files in a directory.
cp	Copies files or directories.	Duplicating files or backing up data.
mv	Moves or renames files and directories.	Renaming or relocating files.
rm	Removes files or directories.	Deleting files or directories.
cat	Displays the contents of a file.	Quickly viewing file contents.
touch	Creates an empty file or updates the timestamp of an existing file.	Creating files or modifying timestamps.
mkdir	Creates a new directory.	Organizing files into directories.
rmdir	Removes an empty directory.	Cleaning up empty directories.

Linux File system hierarchy



Linux File system hierarchy

1. Root directory (/)
The top-level directory from which all other directories branch. It contains the entire file system and serves as the starting point for navigating through the system hierarchy.

2. Boot Loader Files (/boot)
Contains essential files needed to boot the system, such as the bootloader (e.g., GRUB), Linux kernels, and other static files required at startup.

3. Configuration Files (/etc)
Holds system-wide configuration files and shell scripts that control system settings and services.

For example, `/etc/passwd` for user information and `/etc/fstab` for disk mounts.

4. User Home Directories (/home)
Houses directories for individual users, where personal files, preferences, and configuration files reside. Each user has a dedicated folder, such as `/home/username`.

5. Root Home Directory (/root)
The home directory of the root (superuser), containing configuration and environment files specific to root. Only the root user has access to this directory.

Linux File system hierarchy

6. Third-Party Applications (/opt)
Contains add-on software packages and third-party applications that are not part of the default system.

For example, large software installations like Java or other proprietary programs can be placed here.

7. Device Files (/dev)
Contains device nodes, which represent hardware devices like disks, terminals, and USB devices. Special files like `/dev/null` (discards data) and `/dev/sda` (first hard drive) are found here.

8. Variable Files (/var)
Stores files that are expected to grow in size, such as logs, caches, mail, and temporary files. Common subdirectories include `/var/log` (system logs) and `/var/spool` (queued mail or print jobs).

9. User Binaries (/bin)
Contains essential user command binaries that are required for basic system operation and are available to all users, such as `ls`, `cp`, `mv`, and `rm`. These are crucial commands for interacting with the system.

Linux File system hierarchy

10. System Binaries (/sbin)
Stores essential system binaries used for system administration tasks. These commands often require root privileges, such as `shutdown`, `reboot`, `fdisk`, and `ifconfig`.

11. User Applications (/usr)
The largest directory, containing user applications, libraries, documentation, and source code.

Subdirectories like `/usr/bin` store binaries, and `/usr/lib` holds libraries used by user-installed programs.

12. Process Information (/proc)
A virtual file system that provides a view of kernel and process information. It dynamically updates and contains directories named after process IDs (PIDs).

For example, `/proc/cpuinfo` gives information about the CPU.

13. Mount Directory (/mnt)
A temporary location where external storage devices (e.g., additional hard drives, USB devices) are mounted for access. Administrators often manually mount devices here for maintenance or temporary use.

Linux File system hierarchy

14. Virtual File System (/sys)
Similar to `/proc`, this virtual file system provides information about the kernel, hardware devices, and drivers. It exposes device and driver attributes, and it reflects the current state of hardware.

15. Removable Devices (/media)
Automounts removable storage media such as USB drives, CDs, or DVDs. When you plug in a removable device, a directory is automatically created here for accessing the device.

16. Temporary File System (/run)
Stores runtime data used by system processes since the last boot. This data includes PID files and lock files, which are essential for managing currently running processes.

17. Temporary Files (/tmp)
Used by applications to store temporary files that are automatically deleted when the system is rebooted. These files are typically used for short-term storage during program execution.

Linux File system hierarchy

18. Recover Broken Files (/lost+found)
Created during the recovery process after a system crash or improper shutdown. It contains recovered files that were part of the file system but became corrupted or lost.

19. System Libraries (/lib)
Stores shared libraries (similar to Windows DLL files) needed by the essential binaries in `/bin` and `/sbin`. These libraries are critical for the basic functioning of both user and system programs.

20. Service Data Directory (/srv)
Contains data for services offered by the system, such as web server data for Apache or FTP data for FTP servers. This directory holds files related to server operations.

Permissions and ownership

Understanding Permissions (r, w, x) and Numeric Representation (e.g., 777)

In Linux, each file and directory has three types of permissions for three categories of users:

1. **Owner:** The user who owns the file.
2. **Group:** The group that owns the file.
3. **Others:** All other users who are not the owner or part of the group.

The three basic permissions are:

- **r (read):** Allows viewing the contents of a file or listing a directory’s contents.
- **w (write):** Allows modifying or deleting the file. For directories, allows creating or removing files.
- **x (execute):** Allows executing a file (if it’s a program/script). For directories, allows accessing and traversing the directory.

These permissions can be represented numerically:

- r = 4
- w = 2
- x = 1

Permissions and ownership

For each user category (Owner, Group, Others), permissions are represented as a three-digit number:

- **777:** Full permissions (rwx) for Owner, Group, and Others.
- **755:** Full permissions (rwx) for the Owner, and read/execute (r-x) for Group and Others.
- **644:** Read/write (rw-) for the Owner, and read-only (r--) for Group and Others.

Example Breakdown of 777

- Owner (7): Read (4), Write (2), Execute (1) = 7 (rwx)
- Group (7): Read (4), Write (2), Execute (1) = 7 (rwx)
- Others (7): Read (4), Write (2), Execute (1) = 7 (rwx)

Permissions and ownership

Command	Description	Example Usage
chmod	Changes file or directory permissions (r, w, x).	chmod 755 file.txt
chown	Changes file or directory ownership.	chown user:group file.txt
chgrp	Changes group ownership of a file or directory.	chgrp group file.txt
ls -l	Lists files with permissions and ownership info.	ls -l /home
umask	Sets default permissions for new files or directories.	umask 022
setfacl	Sets access control lists (ACLs) for detailed permissions.	setfacl -m u:user:rwx file.txt
getfacl	Displays ACLs for files or directories.	getfacl file.txt

Processes and signals

In a Linux system, processes represent running programs or tasks.

Every process is assigned a unique Process ID (PID) and has associated resource usage, priority, and state.

Managing these processes is essential for system performance, debugging, and ensuring stability in production environments.

For DevOps engineers, understanding and managing processes is crucial when optimizing resource usage, ensuring applications are running smoothly, or troubleshooting issues.

Signals allow you to communicate with processes, for instance, to terminate or modify their behavior, making it easier to automate tasks, manage workloads, and maintain system health.

Processes and signals

Command	Description	Example Usage
ps	Displays a snapshot of running processes.	ps aux
top	Provides real-time system process monitoring.	top
kill	Sends a signal to terminate a process by its process ID (PID).	kill 1234
killall	Terminates all processes with a specific name.	killall apache2
htop	An interactive process viewer with more features than top.	htop
nice	Changes the priority of a running process.	nice -n 10 command
renice	Alters the priority of an already running process.	renice 5 -p 1234

Package management (apt, yum)

Linux package managers like **apt** (for Debian-based systems) and **yum** (for RedHat-based systems) handle software installation, updates, and removal.

Action	apt Command	yum Command
Install	sudo apt install nginx	sudo yum install httpd
Update	sudo apt update && sudo apt upgrade	sudo yum update
Remove	sudo apt remove nginx	sudo yum remove httpd

Networking commands

Networking commands are crucial for troubleshooting connectivity issues, testing network performance, and interacting with remote servers.

These commands help monitor network interfaces, check server availability, and perform tasks such as downloading files or interacting with APIs.

Command	Description	Example Usage
ifconfig	Displays or configures network interfaces, such as IP addresses.	ifconfig eth0 (shows interface info)
ping	Sends packets to a remote host to test connectivity and latency.	ping google.com (tests connection to Google)
netstat	Displays network connections, routing tables, and listening ports.	netstat -an (shows active connections)

Networking commands

Command	Description	Example Usage
curl	Transfers data from or to a server, often used to test APIs.	curl https://api.example.com (gets API response)
wget	Downloads files from the web, commonly used in scripts.	wget https://example.com/file.zip (downloads a file)
nslookup	Queries DNS to find the IP address of a domain or vice versa.	nslookup example.com (finds IP of domain)
traceroute	Traces the route packets take to a network host.	traceroute example.com (shows the hops to the destination)

Disk management

Command	Description	Example Usage
df	Displays the available disk space on file systems.	df -h (human-readable format)
du	Estimates file space usage of directories and files.	du -sh /var/log
fdisk	Manipulates disk partitions on a storage device.	fdisk /dev/sda
mount	Mounts a file system or storage device to the system.	mount /dev/sdb1 /mnt
umount	Unmounts a file system or storage device, safely detaching it.	umount /mnt
lsblk	Lists information about all block devices (disks, partitions).	lsblk

User and group management

Command	Description	Example Usage
adduser	Adds a new user to the system.	sudo adduser john
deluser	Deletes a user from the system.	sudo deluser john
usermod	Modifies an existing user account.	sudo usermod -aG sudo john (adds "john" to the sudo group)
groupadd	Creates a new group on the system.	sudo groupadd developers
passwd	Changes a user's password.	sudo passwd john
groups	Displays group memberships of a user.	groups john
id	Displays user and group IDs.	id john

Networking Concepts

In any DevOps role, a strong understanding of networking is critical. Networks are the backbone of communication between servers, applications, and end-users. Whether you're setting up servers, troubleshooting connectivity issues, or ensuring secure communication, these networking concepts are essential for seamless operations.

Key Concepts:

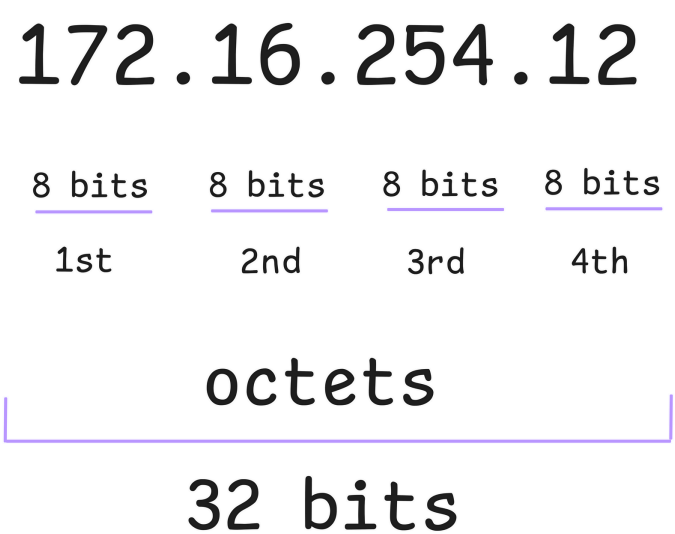
- IP addressing and subnetting (IPv4, IPv6)
- DNS and DHCP
- Load Balancers
- Network protocols (HTTP, HTTPS, FTP, SSH, etc.)

IP addressing and subnetting (IPv4, IPv6)

IP Addressing:
An IP (Internet Protocol) address is a unique identifier for devices on a network. It allows computers, servers, and other devices to communicate over a network.

IPv4 (Internet Protocol version 4)

- Format: IPv4 addresses are written in four groups of numbers separated by periods. Each group (called an octet) is a number between 0 and 255.
- Example: 172.16.254.12



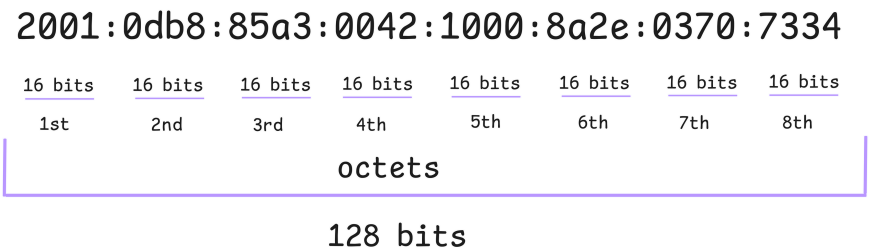
IP addressing and subnetting (IPv4, IPv6)

- Address Space: IPv4 has around 3.7 billion public IP addresses, which are running out due to the growing number of internet-connected devices.

- **Private vs Public IP:**
  - Private IPs: Used within a local network (e.g., home Wi-Fi, corporate networks). Not routable on the internet.
  - Public IPs: Unique across the internet and routable on the global internet.

IPv6 (Internet Protocol version 6)

- Format: IPv6 uses a longer address format with eight groups of four hexadecimal digits separated by colons
- Example: 2001:0db8:85a3:0000:0000:8a2e:0370:7334



IP addressing and subnetting (IPv4, IPv6)

- Address Space: IPv6 provides 340 undecillion IP addresses, solving the problem of IPv4 exhaustion. It’s essential for the Internet of Things (IoT) and future networks.
- IPv6 Compression: IPv6 addresses can be shortened by omitting groups of zeros using ::. For example, 2001:db8:0:0:0:0:2:1 can be written as 2001:db8::2:1.

Subnetting

- Subnetting divides a large network into smaller, more manageable sub-networks (subnets). It improves efficiency, enhances security, and helps with IP address management.

Subnet Mask (IPv4)

- A subnet mask helps determine which part of the IP address is the network and which part is for devices. It looks like an IP address but only uses 255 or 0.

- Example: 255.255.255.0 means the first three numbers (octets) represent the network, and the last number identifies the specific device.

IP addressing and subnetting (IPv4, IPv6)

CIDR Notation

- Classless Inter-Domain Routing (CIDR) is a method for assigning IP addresses and specifying subnet masks more flexibly.

- Example: 192.168.1.0/24 means the first 24 bits (the first three octets) represent the network, and the remaining bits are for device addresses.

IPv6 Subnetting

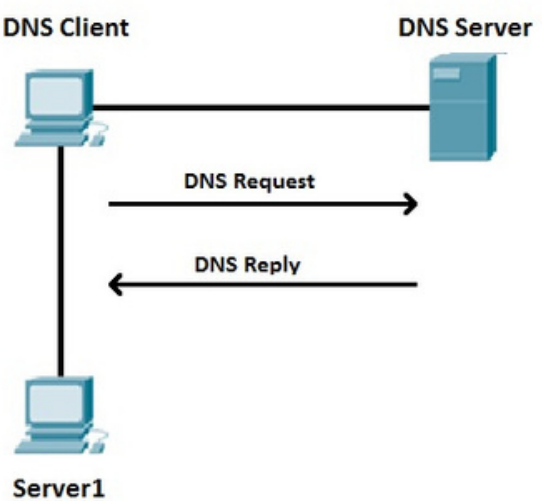
- IPv6 also uses subnetting, but due to its vast address space, subnets are much larger. CIDR notation works similarly, but with IPv6, the most common subnet size is /64.

DNS and DHCP

DNS (Domain Name System): DNS translates human-friendly domain names (e.g., www.google.com) into IP addresses. It’s not required for network connections but makes it easier for users. When a user types a hostname, the DNS server maps it to an IP address.

DNS Process:

1. The DNS client requests the IP address of a hostname (e.g., Server1).
2. The DNS server replies with the corresponding IP address.



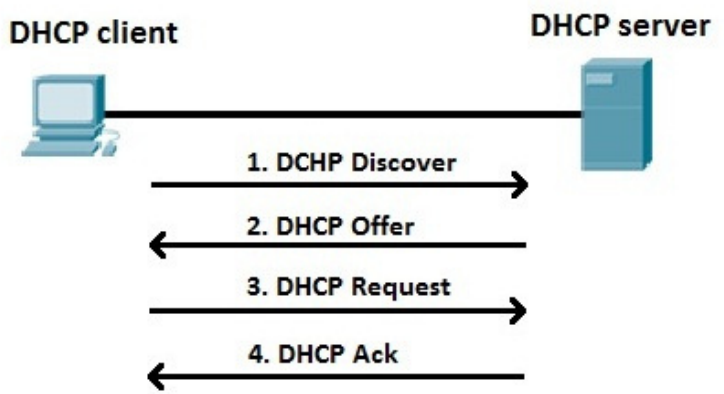
credit: CCNA

DNS and DHCP

DHCP automatically assigns network parameters (like IP addresses) to devices, simplifying network management. It operates as a client-server protocol, where the DHCP server assigns IP addresses and other details (e.g., subnet mask, gateway, DNS server) to DHCP clients.

DHCP Process:

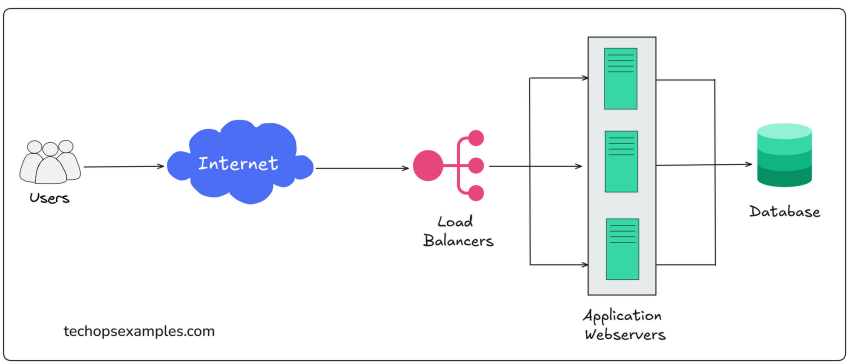
1. The client sends a DHCP Discover message to find DHCP servers.
2. Servers respond with DHCP Offer messages.
3. The client accepts the first offer and sends a DHCP Request.
4. The server confirms with a DHCP Acknowledgment (including lease info).



credit: CCNA

Load Balancers

A load balancer distributes network or application traffic across multiple servers to enhance the capacity, performance, and reliability of applications. It improves scalability, security, and user experience by managing traffic efficiently.



Load balancers operate at different layers:

- **Layer 4:** Works at the network layer, managing traffic based on IP addresses and ports.
- **Layer 7:** Works at the application layer, distributing traffic based on data in application protocols such as HTTP.

Load Balancers

Load balancers use various algorithms to determine how traffic is distributed:

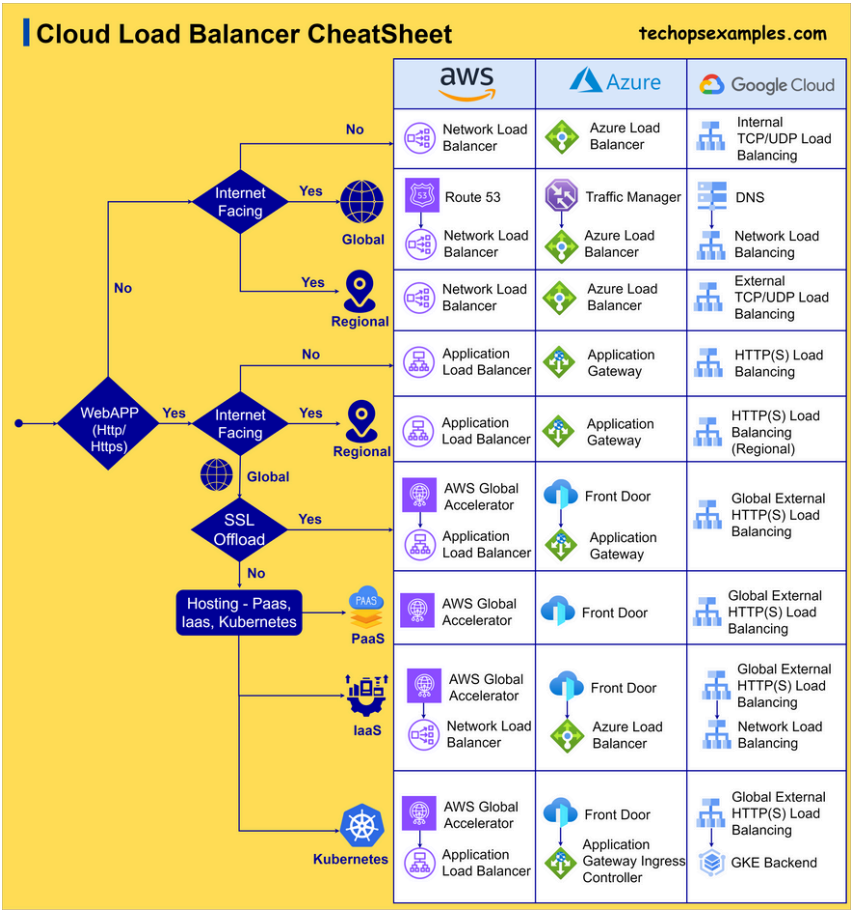
- **Round Robin:** Distributes requests equally across all servers in sequence.
- **Weighted Round-Robin:** Assigns more requests to servers with higher capacity, distributing based on server strength.
- **Least Connections:** Directs traffic to the server with the fewest active connections.
- **Least Response Time:** Sends requests to the server that has the fastest response time.

Advanced Layer 7 Features

Layer 7 load balancers can further distribute traffic based on application-specific data, such as:

- HTTP headers
- Cookies
- Data within the application message (e.g., the value of a specific parameter)

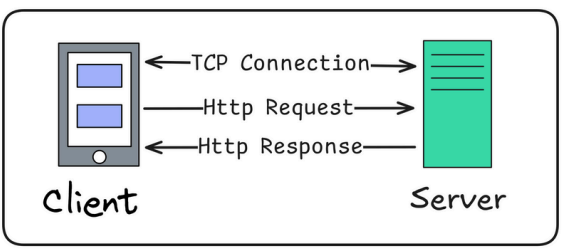
Load Balancers



Network protocols

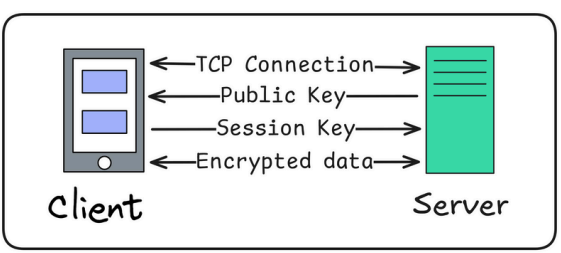
1. HTTP: Establishes a TCP connection, then sends HTTP requests and receives HTTP responses.

Use Cases: Web browsing, API communication



2. HTTPS: Similar to HTTP but adds encryption through public key and session key for secure data transfer.

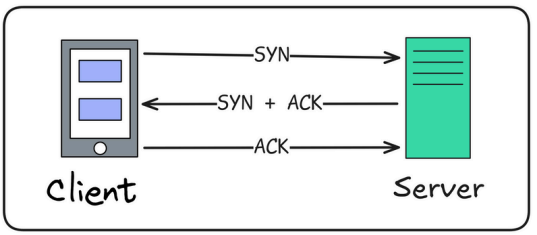
Use Cases: Secure web browsing, e-commerce



Network protocols

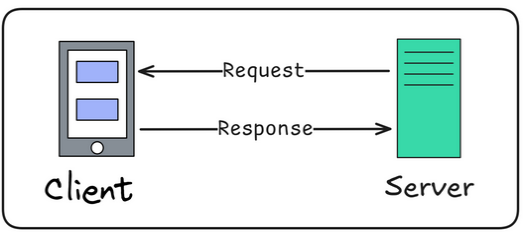
3. TCP: Establishes a connection with a 3-way handshake (SYN, SYN-ACK, ACK) and ensures reliable data transfer.

Use Cases: Web browsing, email, file transfers



4. UDP: Sends data without establishing a connection, prioritizing speed over reliability.

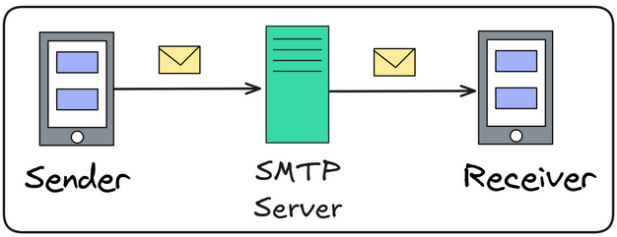
Use Cases: Video conferencing, gaming, DNS queries



Network protocols

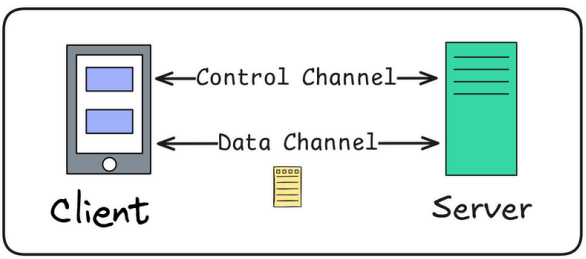
5. SMTP: Transfers emails by sending them from the client to the SMTP server and then to the receiver.

Use Cases: Sending and receiving emails



6. FTP: Uses separate control and data channels to transfer files between devices over a network.

Use Cases: Uploading and downloading files



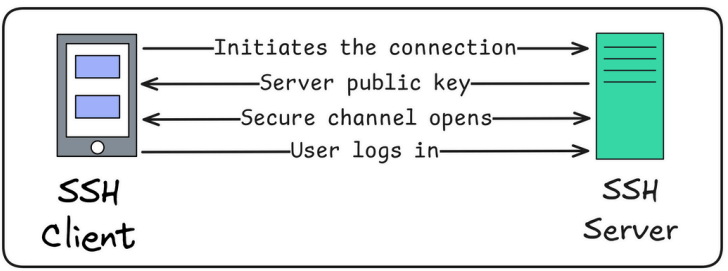
Network protocols

7. SSH: Establishes a secure connection over TCP by using public and private key pairs for authentication and encryption.

The private key stays with the user (client), while the public key is placed on the server. During connection, the server verifies the private key, allowing secure access.

Once connected, all data between the client and server is encrypted, ensuring that sensitive information remains secure during communication.

Use Cases: Remote server management, file transfers, and secure tunneling



Database Concepts

Understanding how databases work is crucial for managing data, ensuring availability, and optimizing performance. Databases are the backbone of data-driven applications, providing the means to store, retrieve, and scale information across various environments. Whether you're working with relational or non-relational databases, these concepts are essential for seamless operations.

Key Concepts:

- ACID Properties
- Scalability
- Data Modeling
- SQL vs. NoSQL

Database Concepts

ACID Properties:
ACID stands for Atomicity, Consistency, Isolation, and Durability.

These properties ensure reliable transactions in relational databases:

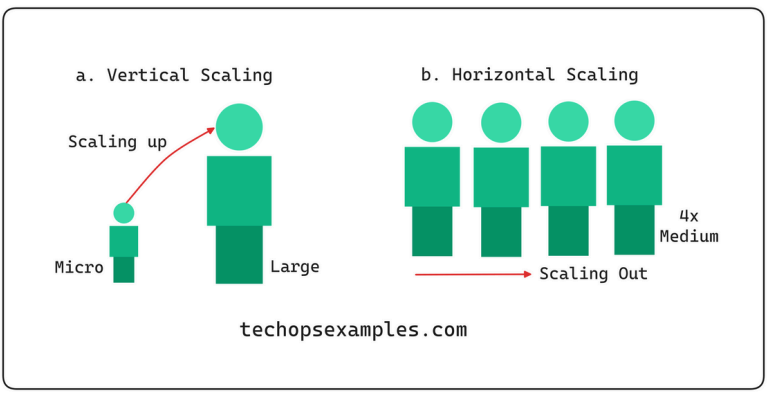
- **Atomicity:** Transactions are all or nothing. If one part fails, the entire transaction fails, ensuring no partial updates.
- **Consistency:** Each transaction brings the database from one valid state to another, maintaining data integrity.
- **Isolation:** Transactions occur independently of one another, preventing them from affecting each other’s results.
- **Durability:** Once a transaction is committed, it remains in the system, even in the event of a crash or failure.

Database Concepts

Scalability refers to the ability of a database to handle increasing workloads efficiently.

There are two types of scalability:

- **Vertical Scalability:** Scaling up by upgrading the hardware (more CPU, memory, storage).
- **Horizontal Scalability:** Scaling out by distributing the data across multiple servers or nodes (sharding). NoSQL databases are often more horizontally scalable than SQL databases.

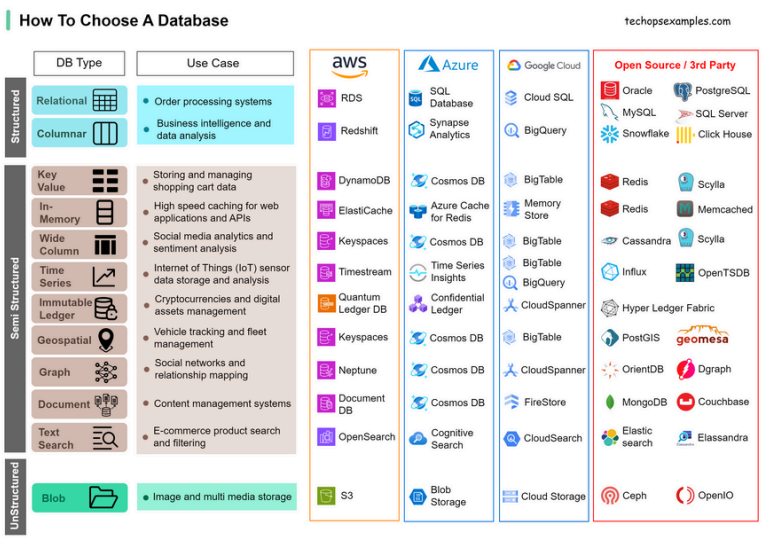


Database Concepts

Data modeling is the process of designing the structure of the database to store data efficiently and logically.

This includes defining the relationships between data points and structuring tables in SQL or documents, key-value pairs, etc., in NoSQL databases. Proper data modeling ensures optimized performance and storage.

- **In SQL:** Data is modeled using tables, rows, and columns with strict relationships and constraints.
- **In NoSQL:** Data models are flexible, with formats like key-value, document, wide-column, or graph structures.



Database Concepts

SQL vs NoSQL

Aspect	SQL	NoSQL
Data Model	Structured tables (rows and columns)	Flexible, semi-structured or unstructured data (e.g., JSON)
Data Model	Fixed schema, strict relationships	Dynamic schema, allows flexible data formats
Transactions	Supports ACID properties	Supports BASE (Basically Available, Soft-state, Eventual consistency) transactions
Scalability	Vertically scalable	Horizontally scalable
Use Case	Best for OLTP	Best for big data, unstructured data
Examples	MySQL, PostgreSQL, Oracle	MongoDB, Cassandra, Redis

Database Concepts

When to Choose SQL vs NoSQL

Choose SQL:

- When your data has a well-defined structure (rows and columns).
- When transactions need ACID compliance, ensuring data reliability.
- For applications that involve complex queries, like relational data analysis (e.g., banking, ERP systems).

Choose NoSQL:

- When dealing with large volumes of unstructured or semi-structured data.
- When you need high scalability and availability across distributed systems.
- For use cases where flexible schemas or rapid development cycles are needed (e.g., social media platforms, IoT applications).

Security Concepts

security is a critical aspect of managing systems and infrastructure. Ensuring the protection of data, applications, and networks from potential threats is essential for maintaining trust and preventing breaches. Whether it’s about securing communication, enforcing access controls, or ensuring compliance, these concepts form the foundation of secure operations in any environment.

Key Concepts:

- Encryption
- Authentication
- Authorization
- OWASP Top 10

Security Concepts

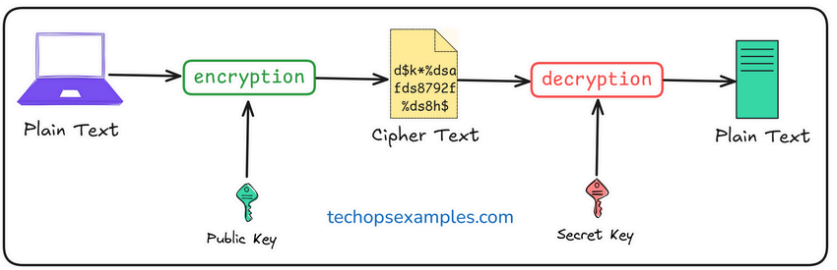
Encryption

Encryption is the process of converting data into a coded format to prevent unauthorized access. It ensures that only authorized parties can decode and access the original information.

How Encryption Works:

1. **Encryption:** Data in plain text is transformed into unreadable cipher text using a public key.
2. **Cipher Text:** This encrypted data can only be read if decrypted by someone with the correct key.
3. **Decryption:** The cipher text is converted back into plain text using the corresponding secret (private) key.

This process ensures that even if data is intercepted, it remains unreadable without the appropriate decryption key.



Security Concepts

Why Encryption is Important

- **Data privacy:** Protecting sensitive personal or business information.
- **Compliance:** Meeting legal and regulatory requirements for data protection.
- **Preventing breaches:** Safeguarding data in case of hacking attempts or physical theft.

Encryption at Rest protects stored data by converting it into unreadable cipher text. This ensures that data on storage devices (such as hard drives, databases, or cloud storage) remains secure even if accessed by unauthorized parties.

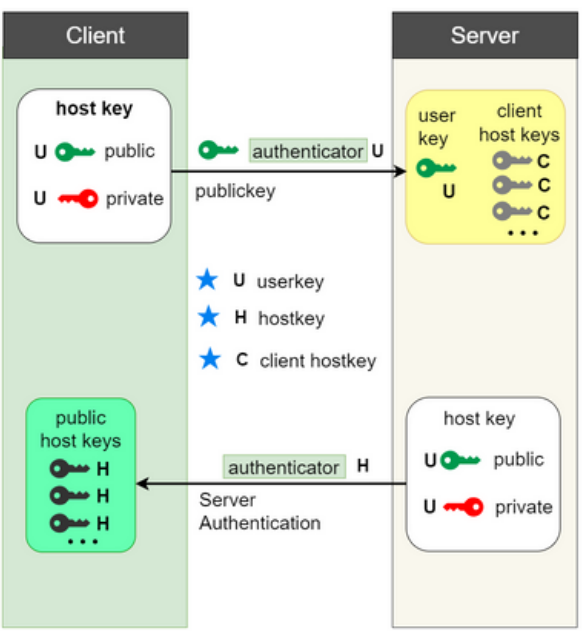
Encryption in Transit protects data as it is transferred across networks. It ensures that data remains secure and unreadable while being transmitted between systems, services, or users.

Security Concepts

Authentication is the process of verifying the identity of a user or system before granting access. It ensures only authorized users can access sensitive data or systems, preventing unauthorized breaches.

1. SSH Keys

SSH keys use public-private key pairs to authenticate a client with a server. The client uses the private key, and the server verifies it using the public key stored on the server. This method provides secure, passwordless login for remote access.

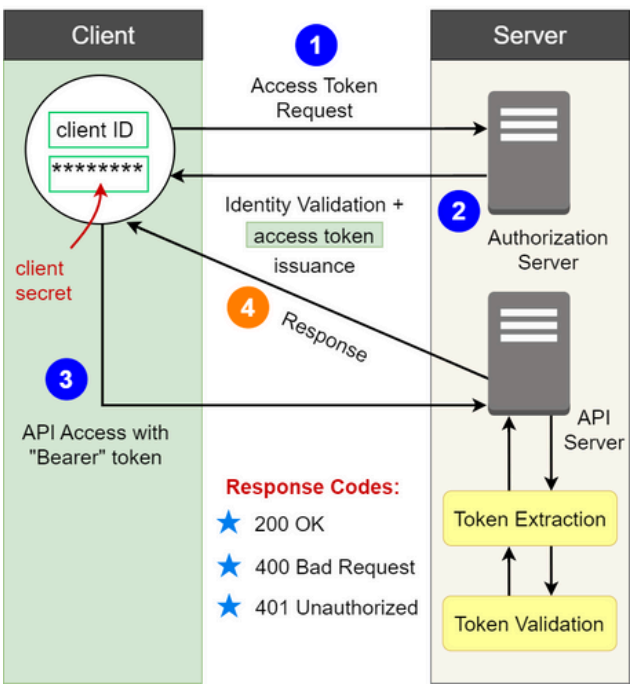


Security Concepts

2. OAuth Tokens

OAuth is an authorization protocol where a client requests an access token from an authorization server using its client ID and client secret.

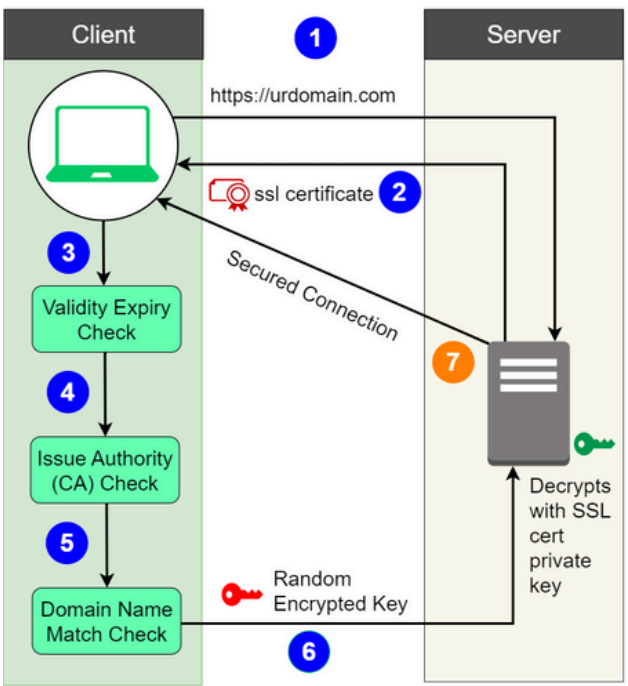
The token is used for API access and is validated by the server for authorization. OAuth is commonly used for third-party access without exposing user credentials.



Security Concepts

3. SSL Certificates

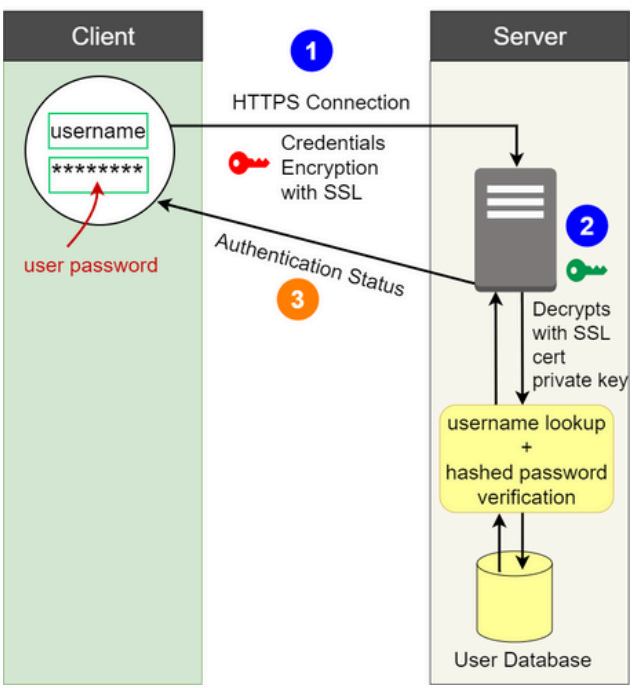
SSL certificates are used to establish a secure connection between a client and a server. The client receives the server's SSL certificate and verifies its validity, authority, and domain. Once validated, the client and server exchange encrypted keys for secure communication.



Security Concepts

4. Credentials

Traditional username-password authentication. The client submits credentials (username and password) over an encrypted HTTPS connection, and the server verifies the credentials by looking them up in the user database. If matched, access is granted.



Security Concepts

Authorization is the process of determining what actions or resources an authenticated user is permitted to access. It is essential for security as it ensures users can only perform actions or view data they are allowed to, protecting sensitive information and preventing misuse.

Aspect	Authentication	Authorization
Definition	Verifying the identity of a user or system	Determining what resources or actions a user can access
Purpose	Confirms the user is who they claim to be	Controls what the user is allowed to do
Occurs When	First step in the security process	Happens after authentication
Examples	Logging in with a password or SSH key	Granting permissions to read, write, or modify data
Involves	Credentials (passwords, keys, tokens)	Permissions and roles

Security Concepts

OWASP (Open Web Application Security Project) is a global non-profit organization focused on improving web application security. It provides free, accessible resources like documentation, tools, and the well-known OWASP Top 10 list, which highlights the most critical security risks in web applications.

Latest OWASP Top 10:

- Broken Access Control
- Cryptographic Failures
- Injection
- Insecure Design
- Security Misconfiguration
- Vulnerable and Outdated Components
- Identification and Authentication Failures
- Software and Data Integrity Failures
- Security Logging and Monitoring Failures
- Server-Side Request Forgery (SSRF)

For more details, check here.

Storage Concepts

Storage plays a pivotal role in handling data and infrastructure. Guaranteeing the efficiency, scalability, and dependability of storage systems is crucial for ensuring high performance and effectively managing large datasets. Whether it involves choosing the appropriate storage type, optimizing storage solutions, or safeguarding data integrity, these principles serve as the backbone of reliable data operations in any environment.

Key Concepts:

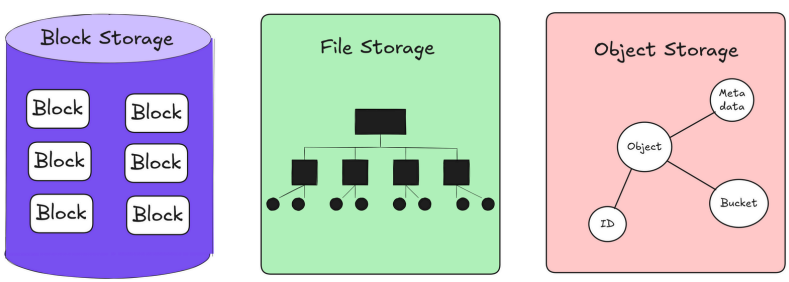
- Block Storage
- Object Storage
- File Storage
- SSD vs. HDD

Storage Concepts

Block Storage divides data into fixed-size blocks. Each block operates independently and can be stored across different environments. This storage type is typically used for databases, virtual machines, and high-performance applications.

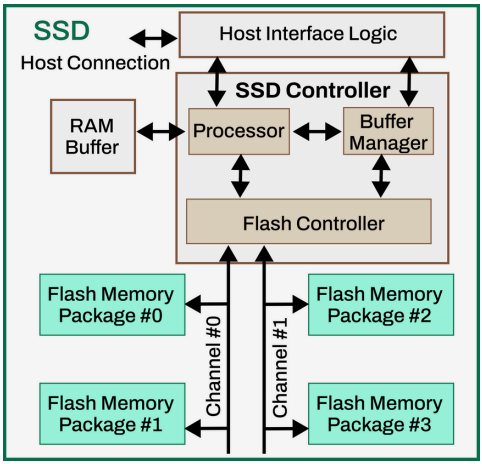
Object Storage stores data as discrete units called "objects," which include the data itself, metadata, and a unique identifier. It’s well-suited for unstructured data like images, videos, and backups, allowing for easy scaling and access.

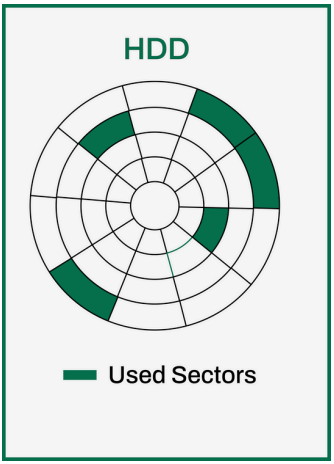
File Storage organizes data in a hierarchical structure, using directories and folders. It’s ideal for shared environments where multiple users and applications need access to the same data.



Storage Concepts

SSD vs HDD





SSD (Solid State Drive):
SSDs are crucial for performance-heavy applications, offering faster read/write speeds and reducing latency. They’re perfect for running virtual machines, containerized apps, or high-traffic databases, where speed and reliability are essential.

HDD (Hard Disk Drive):
Though slower, are often used for long-term storage or archiving. They’re more affordable for storing large volumes of logs, backups, or data that doesn't require frequent access or high speed.

Disaster Recovery Strategies

Disaster recovery strategies are vital for ensuring business continuity and minimizing downtime during unexpected failures. A strong approach enables quick restoration of services, protection of critical data, and uninterrupted operations. This involves automated backups, failover systems, and regular testing of recovery procedures, keeping infrastructure resilient and reducing the impact of outages or data loss on applications and customers.

Key Concepts:

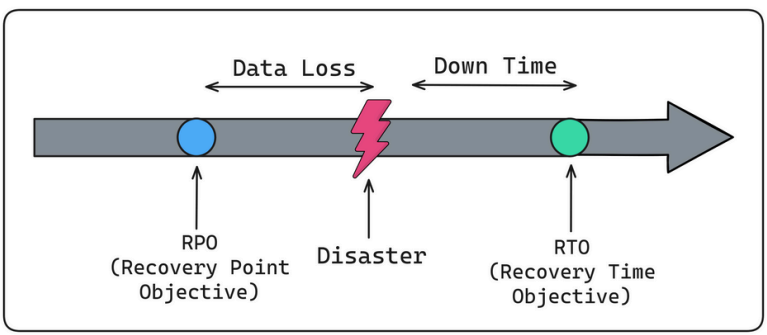
- RTO (Recovery Time Objective)
- RPO (Recovery Point Objective)
- Backup and Restore
- Pilot Light
- Warm Standby
- Multi-site

Disaster Recovery Strategies

Any DR strategy starts with finalizing:

1. RTO (Recovery Time Objective):
How much downtime can one accept?

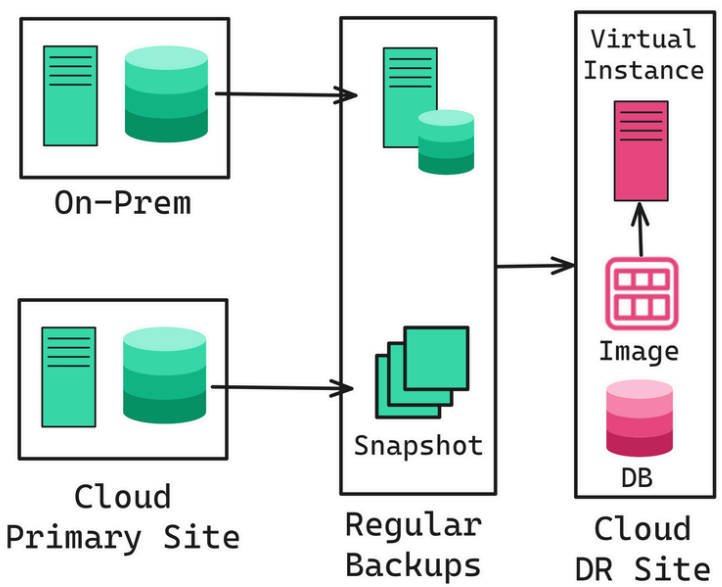
2. RPO (Recovery Point Objective):
How much data loss can one accept?



Without arriving at these objectives, it's impossible to design an effective disaster recovery plan. They define the thresholds for how quickly systems must recover and how much data can be lost without major impact, guiding every decision in your disaster recovery strategy—from backup frequency to the choice of recovery architecture.

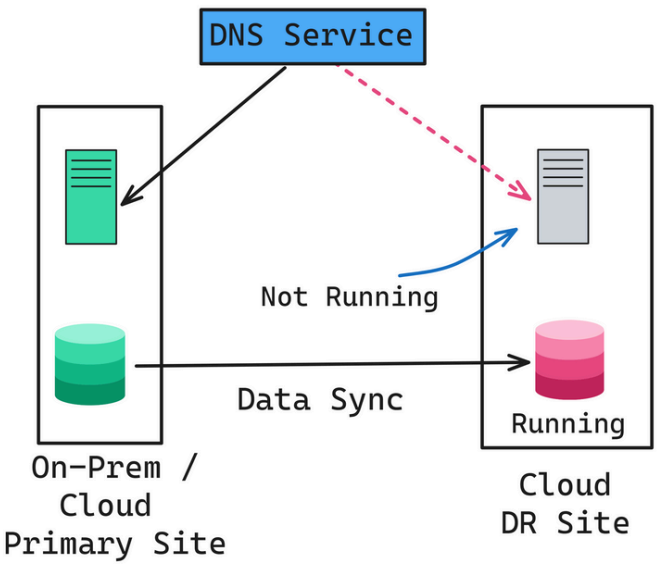
Disaster Recovery Strategies

1. Backup and Restore:
This is the simplest strategy, where regular backups are taken from the primary site and stored in a DR (Disaster Recovery) site. After a disaster, data and applications are restored from these backups, though it may involve significant downtime and data loss depending on the backup frequency.



Disaster Recovery Strategies

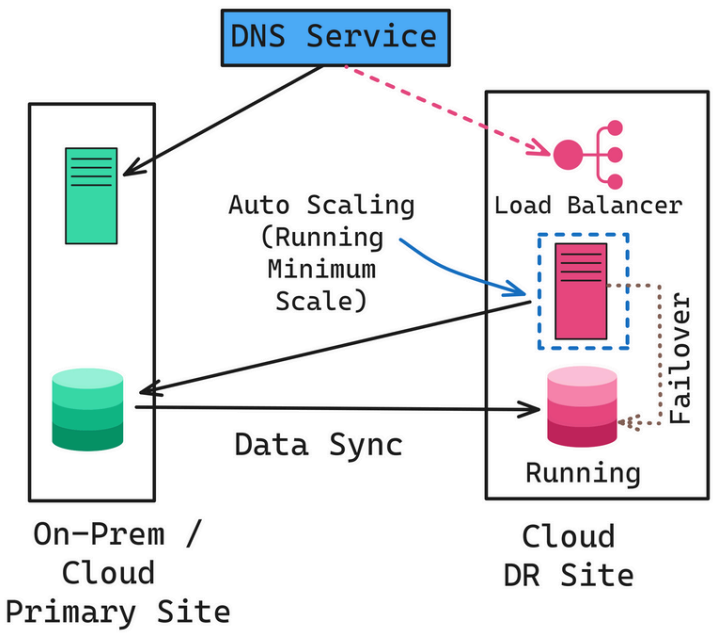
2. Pilot Light:
A minimal version of your environment is always running in the DR site. Data is continuously synced, but critical services remain inactive until a disaster occurs. When needed, the DR site can be quickly scaled up to take over from the primary site, reducing both downtime and data loss.



techopsexamples.com

Disaster Recovery Strategies

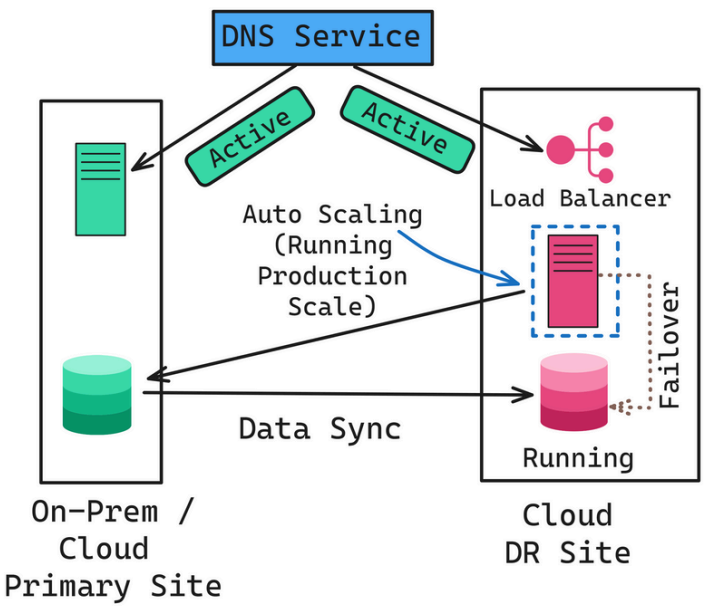
3. Warm Standby:
The DR site runs at a reduced capacity, with minimal resources active and synchronized. In case of failure, it can scale up quickly to full capacity, providing faster recovery compared to the pilot light, with lower downtime and data loss.



techopsexamples.com

Disaster Recovery Strategies

4. Multi-Site:
Both the primary and DR sites are fully active, with real-time data synchronization and load balancing between them. This ensures minimal downtime and data loss, as either site can take over instantly in case of a disaster.



Understanding Cache

Efficient caching can be the difference between a sluggish system and a high-performing infrastructure. For DevOps engineers, knowing how to leverage cache optimally means faster deployments, reduced latency, and less strain on resources. Whether dealing with data storage, request handling, or system updates, mastering cache strategies helps keep services responsive and scalable even under heavy loads.

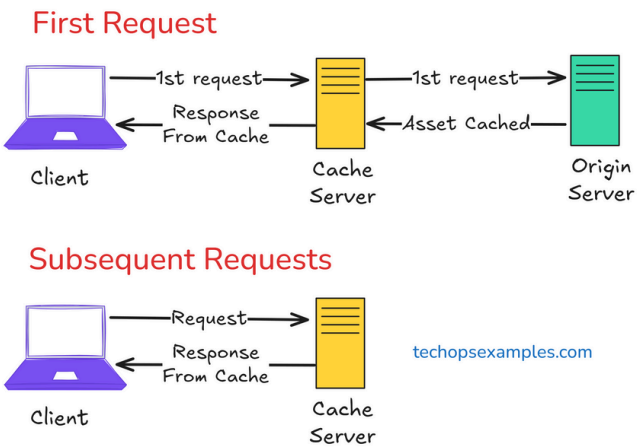
Key Concepts:

- How Cache works?
- Cache Hit Vs Cache Miss
- In-memory Caches (Redis, Memcached)

Understanding Cache

WHow Cache works?

A cache is high-speed memory that temporarily stores data, allowing faster access to content, such as web pages or database queries, on subsequent visits. It improves load times and enhances user experience by reducing the need to retrieve the same data repeatedly.



A cache hit occurs when requested data is found in the cache, speeding up retrieval.

A cache miss happens when data isn't cached, forcing the system to fetch it from the original source, which takes longer. More cache hits lead to better performance, while frequent cache misses slow down system response times.

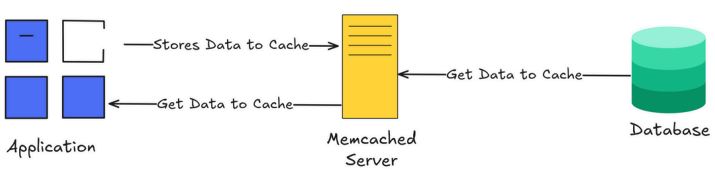
Understanding Cache

I**n-memory caches** store data in the system's RAM, offering rapid access to frequently used data, which significantly boosts performance. They're essential for applications requiring low-latency data retrieval.

Memcached is a distributed memory caching system used to enhance the performance and scalability of web applications by reducing the load on databases.

Here’s a breakdown of its role in system design:

- **In-Memory Storage:** Memcached stores data in RAM, which significantly speeds up access compared to disk-based storage.
- **Distributed Architecture:** It can run on multiple servers, distributing the cache across them to balance load and scale horizontally.
- **Key-Value Storage:** Data is stored as key-value pairs, enabling simple and efficient data retrieval.
- **Volatile Storage:** Memcached is non-persistent, meaning data is lost on server restart or when eviction occurs due to memory limitations.



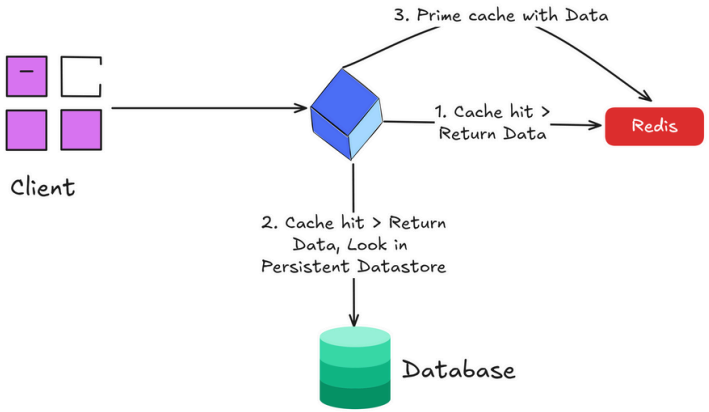
Understanding Cache

Redis is an in-memory data structure store that not only offers caching but also supports advanced data manipulation operations. Redis is known for its flexibility and persistence options, making it a powerful choice for more complex use cases. Here’s what makes Redis stand out:

- **Advanced Data Structures:** Redis supports a variety of data types like strings, lists, sets, and hashes, allowing more complex operations beyond simple key-value caching.
- **Persistence Options:** Unlike Memcached, Redis offers persistence through snapshotting and AOF (Append Only File), ensuring data survives restarts.
- **Replication and Clustering:** Redis supports master-slave replication and automatic failover through clustering, making it a good fit for high availability.
- **Pub/Sub and Scripting:** Redis also supports pub/sub messaging and Lua scripting, making it highly versatile for use cases like real-time analytics, leaderboards, or distributed locking.

Understanding Cache

How Redis Typically Used ?



Here's a breakdown of the process:

1. **Cache Hit:** When the client requests data, Redis first checks if the data exists in the cache. If it's a cache hit, Redis returns the data directly, ensuring quick access.
2. **Cache Miss:** If the requested data is not in the cache (a cache miss), Redis retrieves the data from the persistent database, which usually takes more time.
3. **Priming the Cache:** After retrieving data from the database, Redis stores (or "primes") the cache with this data so that subsequent requests can be served faster, reducing the need to query the database again.

Mastering DevOps is a marathon, not a sprint. This guide is your foundation—build on it, stay patient through challenges, and everything will fall into place.

— Govardhana Miriyala Kannaiah

Give a shout-out on how this guide helped you on my LinkedIn and Twitter (X), I'm listening.

LinkedIn link

Twitter (X) link

