DevOps Basics

[Course Details 2](#_Toc193568590)

[AWS Certifications 2](#_Toc193568591)

[Client Server Architecture 2](#_Toc193568592)

[1-Tier Architecture (Single Layer) 4](#_Toc193568593)

[2-Tier Architecture 5](#_Toc193568594)

[3-Tier Architecture 7](#_Toc193568595)

[Network Infrastructure and DNS 10](#_Toc193568596)

[HTTP & HTTPS Concepts 12](#_Toc193568597)

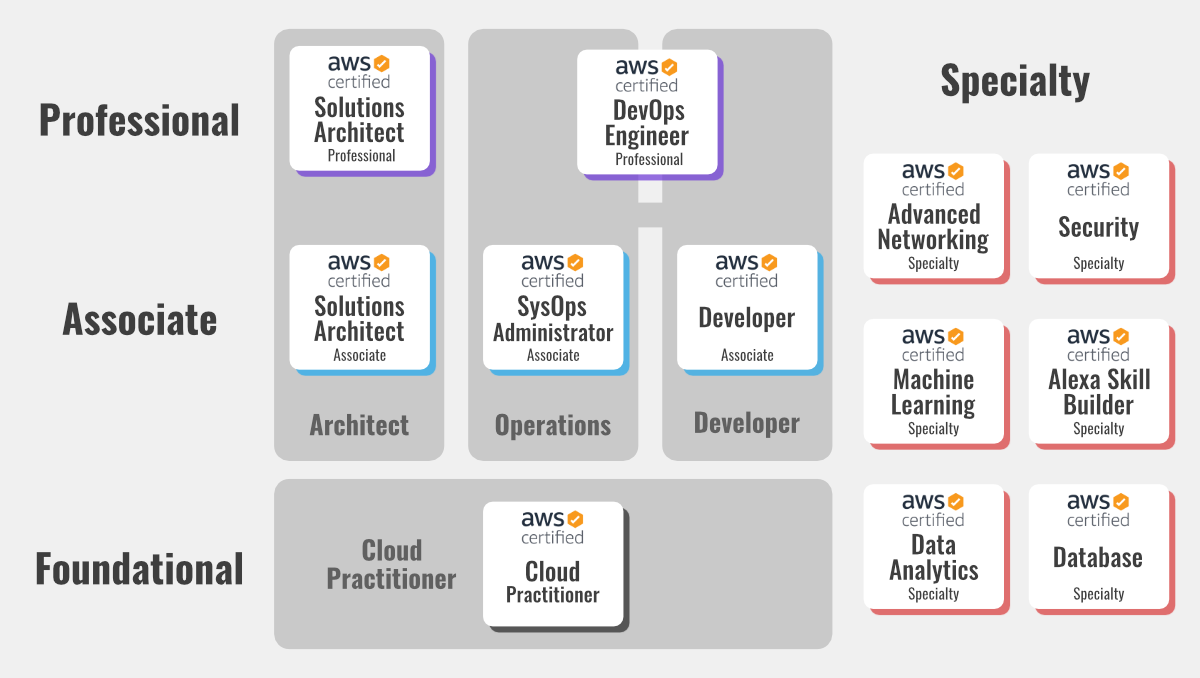
[OSI Model (Open Systems Interconnection) 14](#_Toc193568598)

# Course Details

|  |  |
| --- | --- |
| Section | Details |
| Course Strategy |  |
| Part 1: AWS - Manual | Focus on understanding and manually working with AWS core services. |
| Part 2: DevOps - Automation | Learn DevOps tools and practices for automating workflows and deployments. |
| Part 3: AWS with DevOps | Combine AWS services with DevOps practices for end-to-end cloud solutions. |

# AWS Certifications

|  |  |  |
| --- | --- | --- |
| Level | AWS Certifications | Description |
| Foundational | Cloud Practitioner | Covers AWS basics and is designed for individuals starting their cloud journey. |
| Associate | Developer Associate | Focuses on using AWS for application development and maintenance. |
| SysOps Admin Associate | Centers on deploying, managing, and operating scalable and fault-tolerant systems on AWS. |
| Solutions Architect Associate | Targets those designing distributed systems on AWS. |
| Professional | DevOps Engineer | Combines advanced AWS skills with DevOps principles for managing and automating infrastructure. |
| Solutions Architect Professional | Focuses on complex AWS architectures and advanced solution designs. |



# Client Server Architecture

**Fundamentals**

* **Resource:** Information or data that a client requests from a server.
* **Client:** A device or program that requests resources or services from a server. (e.g., Web browser, email client)
* **Server:** A device or program that provides resources or services to clients. (e.g., Web server, email server)
* **Network:** The communication pathway that connects clients and servers. (e.g., Internet, local network)

**Key Concepts**

* **Request-Response Model:**
  + Clients initiate communication by sending requests to the server.
  + Servers process requests and send back responses to the client.
* **Role Flexibility:** Devices can act as both a client and a server depending on the situation.

**Example: Web Browsing**

1. You type "google.com" into your web browser (client).
2. The browser sends a request over the internet to the Google server.
3. The Google server receives the request and processes it.
4. The server sends the requested webpage (resource) back to your browser.
5. Your browser displays the webpage.

**Advantages of Client-Server Architecture**

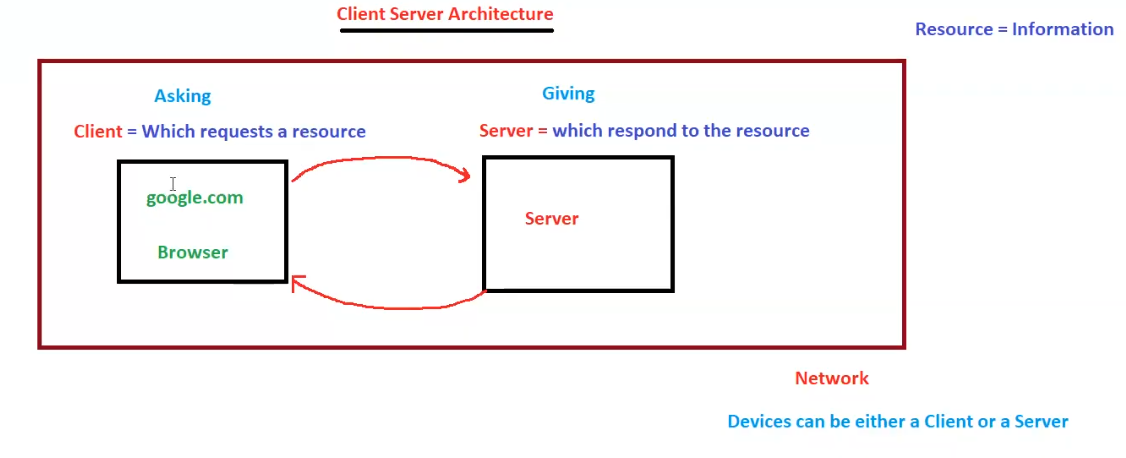
* **Centralized Data Management:** Data is stored and managed on the server, making it easier to control access, maintain data integrity, and perform backups.
* **Scalability:** Servers can be upgraded or more servers can be added to handle increasing client demands.
* **Security:** Servers can implement security measures to protect data from unauthorized access.
* **Flexibility:** Clients and servers can be different types of devices running different operating systems.

**Disadvantages of Client-Server Architecture**

* **Single Point of Failure:** If the server fails, the entire system can be affected.
* **Network Dependency:** The system relies on a stable network connection.
* **Cost:** Servers can be expensive to purchase and maintain.

**Additional Notes:**

* Client-server architecture is a fundamental concept in networking and distributed computing.
* Many applications and services we use daily rely on this model (e.g., the web, email, online gaming).



## 1-Tier Architecture (Single Layer)

**Fundamentals**

* **Single Layer:** All components (presentation, business logic, and data storage) reside on a single machine, often the user's local machine or a standalone server.
* **Localhost:** A hostname that refers to the current computer on which the code is running.
* **Client/Server on Same Machine:** The application acts as both the client and the server, running on the same system.
* **Limited Scalability:** Difficult to scale as all processing occurs on a single machine.
* **Suitable for Simple Applications:** Ideal for basic applications with low user traffic and minimal resource requirements.

**Example: Local Development Environment**

* A developer (like Ganesh in the diagram) builds and tests a website on their laptop.
* The website files, database, and browser all reside on the same machine.
* The developer accesses the website using "localhost" in the browser's address bar.

**Advantages of 1-Tier Architecture**

* **Simplicity:** Easy to set up and maintain due to its single-layer structure.
* **Cost-Effective:** No need for separate servers or network infrastructure.
* **Suitable for Offline Use:** Can function without a network connection.

**Disadvantages of 1-Tier Architecture**

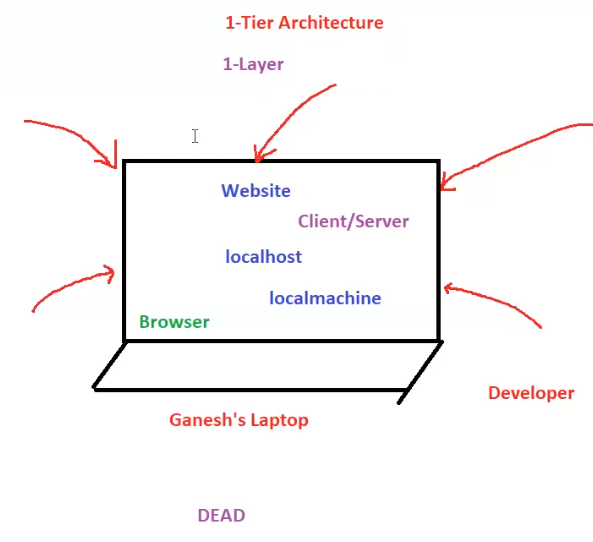
* **Limited Scalability:** Cannot handle large numbers of users or complex applications.
* **Security Concerns:** Data might be more vulnerable as it resides on the same machine as the client.
* **Performance Bottlenecks:** All processing occurs on a single machine, which can lead to performance issues.

**Additional Notes**

* 1-tier architecture is often used for personal projects, small-scale applications, or during the initial development phase of larger projects.
* It is less common in production environments due to its limitations.

**DEAD**

* The "DEAD" label likely signifies that this architecture is not suitable for large-scale or complex applications due to its limitations in scalability and performance.



## 2-Tier Architecture

**Fundamentals**

* **Two Layers:**
  + **Client Layer:** Handles user interface and user interactions.
  + **Server Layer:** Houses the application logic and the database.
* **Direct Communication:** Clients communicate directly with the server.
* **Suitable for Moderate-Scale Applications:** Can handle more users and complexity than 1-tier but has limitations compared to 3-tier.

**Components**

* **Clients:** Various devices (e.g., computers, mobiles) running client applications.
* **Server:** A dedicated machine running the application and database.
* **Application:** Processes client requests, performs business logic, and interacts with the database. (e.g., written in Java, .NET, Python)
* **Database:** Stores and manages application data. (e.g., MySQL, Oracle, MS SQL)

**Advantages of 2-Tier Architecture**

* **Improved Performance:** Compared to 1-tier, separate server handles processing, potentially improving performance.
* **Easier Development:** Simpler to develop and maintain than more complex architectures.
* **Cost-Effective:** Less expensive than 3-tier as it requires fewer servers.

**Disadvantages of 2-Tier Architecture**

* **Scalability Challenges:** Can become a bottleneck as the number of users increases.
* **Security Concerns:** Direct client-server communication can pose security risks.
* **Maintenance Overhead:** Updates and modifications can be more complex as the application and database are tightly coupled.

**"BIG NO FOR PROD"**

* The image emphasizes that 2-tier architecture is generally not recommended for production environments, especially for large-scale or critical applications.
* This is due to its limitations in scalability, security, and maintainability.

**Potential Issues Illustrated**

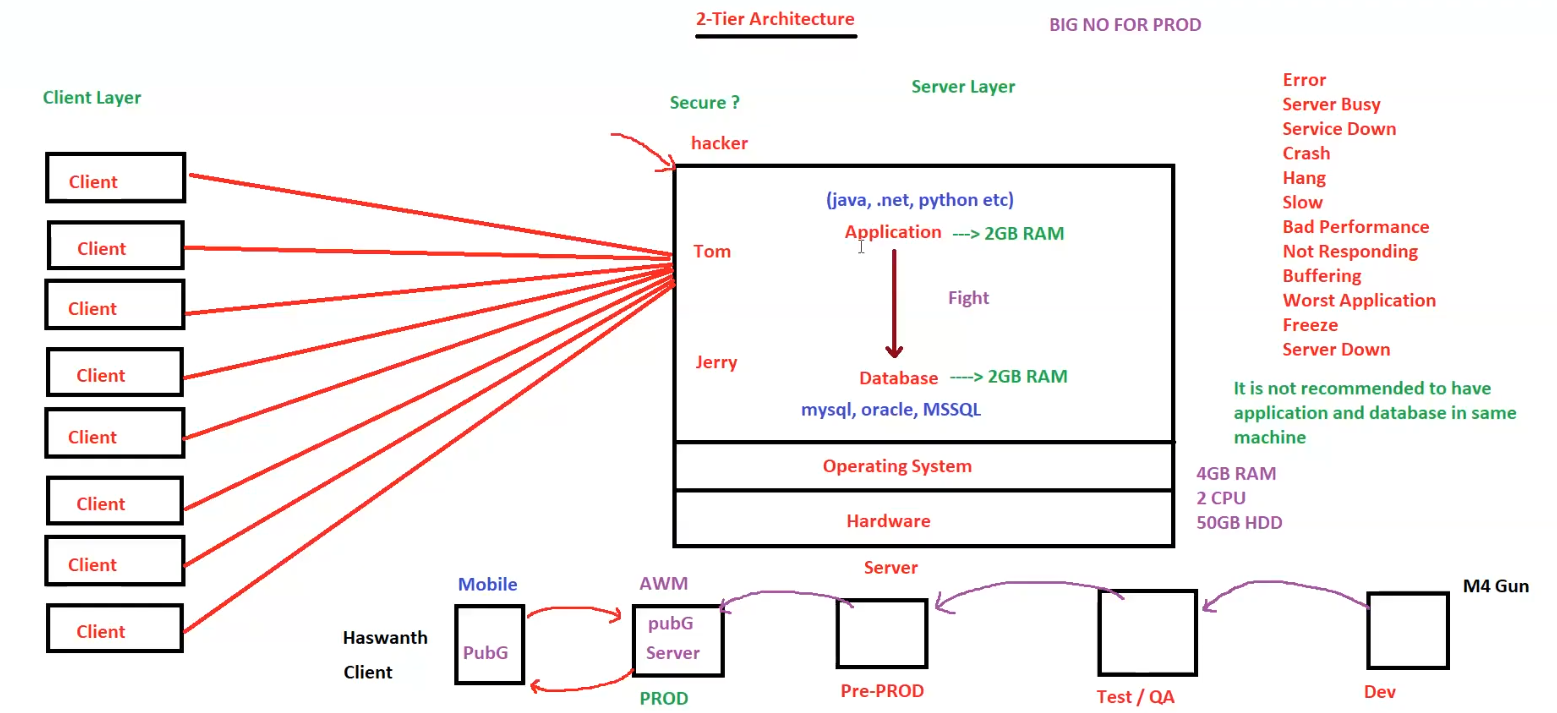
* **Server Overload:** Multiple clients accessing the server simultaneously can lead to errors, slowdowns, crashes, and poor performance.
* **Security Vulnerability:** A "hacker" is shown, highlighting the potential security risks of direct client-server communication.
* **Resource Constraints:** The server having limited resources (2GB RAM) for both the application and database can further impact performance.

**Example: "Tom and Jerry" Analogy**

* "Tom" (application) and "Jerry" (database) constantly fighting represents the potential conflicts and performance issues when both reside on the same server with limited resources.

**Additional Notes**

* 2-tier architecture was more common in the early days of client-server computing.
* While not ideal for large production systems, it can still be suitable for certain applications with moderate user loads and less demanding requirements.



## 3-Tier Architecture

**3-Tier Architecture**

**Fundamentals**

* **Three Layers:**
  + **Client Layer:** Presentation tier (user interface) - interacts with the user.
  + **Application Layer:** Logic tier (business logic, application code) - processes data and handles business rules.
  + **Database Layer:** Data tier (data storage) - stores and manages data.
* **Indirect Communication:** Clients do not directly access the database; they interact with the application server.
* **Enhanced Scalability and Security:** Each tier can be scaled independently, and the database is protected behind the application server.
* **Suitable for Complex Applications:** Ideal for large-scale applications with high user traffic and complex requirements.

**Components**

* **Clients:** Devices like web browsers accessing the application.
* **Web Server:** Handles client requests and redirects them to the application server (e.g., Nginx, Apache, httpd).
* **Application Server:** Hosts the application logic and processes client requests (e.g., Tomcat, IIS).
* **Database Server:** Stores and manages the application's data (e.g., MySQL).

**Communication Flow**

1. Client sends a request to the web server.
2. Web server redirects the request to the application server.
3. Application server processes the request, interacts with the database server if needed, and sends the response back to the web server.
4. Web server forwards the response to the client.

**Network Security**

* **Public Network:** The web server is accessible from the public internet.
* **Private Network:** The application server and database server are in a private network, protected from direct access from the internet. This enhances security.

**N-Tier Architecture**

* **Extends 3-Tier:** Can have more than three tiers, with additional layers for tasks like security, caching, load balancing, etc.
* **Increased Flexibility:** Offers greater flexibility and scalability for complex applications.

**Advantages of 3-Tier/N-Tier Architecture**

* **Scalability:** Each tier can be scaled independently to handle increased load.
* **Maintainability:** Easier to maintain and update as changes in one tier don't necessarily affect others.
* **Security:** Database is protected behind the application server.
* **Reusability:** Components can be reused across multiple applications.

**Disadvantages of 3-Tier/N-Tier Architecture**

* **Complexity:** More complex to design and implement than 2-tier.
* **Performance:** Can be slower than 2-tier if not designed and optimized properly.
* **Cost:** May require more hardware and resources.

**Additional Notes**

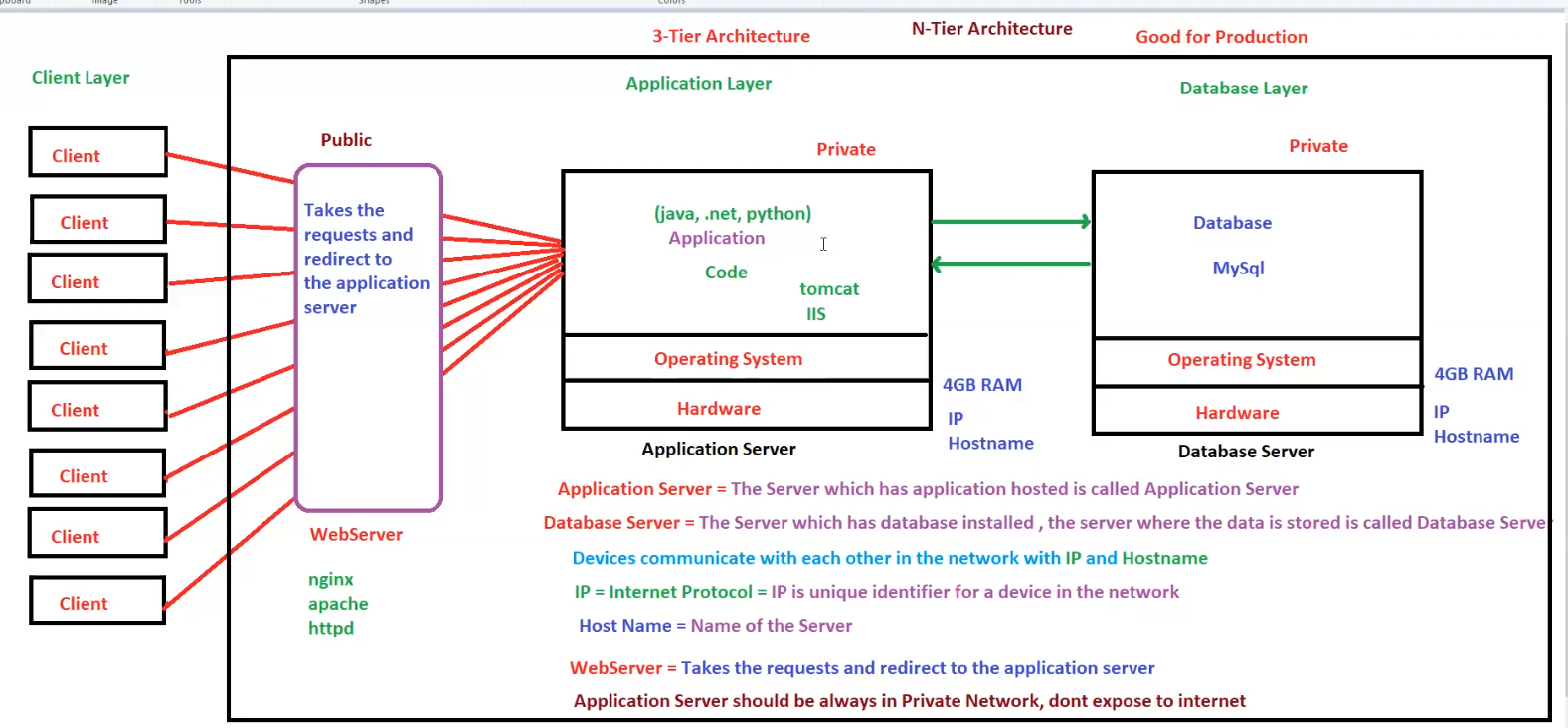
* 3-tier and N-tier architectures are widely used for enterprise-level applications and web services.
* They provide a robust and scalable solution for handling complex applications and large user bases.

**Diagram Explanation**

The diagram illustrates the different tiers, components, and communication flow in a 3-tier architecture. It also highlights the importance of keeping the application server and database server in a private network for security reasons.

**Key Terms**

* **IP Address:** A unique identifier for a device on a network.
* **Hostname:** A human-readable name for a server.



**Tier 1 vs Tier 2 vs Tier 3**

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | 1-Tier Architecture | 2-Tier Architecture | 3-Tier Architecture |
| Layers | Single | Two (Client, Server) | Three (Client, Application, Database) |
| Components | All components on one machine | Client and server | Client, web server, application server, database server |
| Communication | N/A (all on one machine) | Direct client-server | Client <-> Web Server <-> Application Server <-> Database Server |
| Scalability | Very limited | Moderate | High |
| Security | Low | Moderate | High |
| Complexity | Simple | Moderate | Complex |
| Performance | Limited by single machine | Can be improved over 1-tier | Potentially high with proper optimization |
| Cost | Low | Moderate | High |
| Typical Use Cases | - Personal projects  - Small-scale applications  - Local development | - Applications with moderate user load | - Enterprise applications  - High-traffic web services  - Complex systems |
| Suitable for Production? | No | Generally, not recommended | Yes |
| Advantages | - Simplicity  - Cost-effectiveness | - Improved performance over 1-tier  - Easier development than 3-tier | - High scalability  - Maintainability  - Security  - Reusability |
| Disadvantages | - Limited scalability  - Security concerns  - Performance bottlenecks | - Scalability challenges  - Security concerns  - Maintenance overhead | - Complexity  - Potential performance overhead  - Higher cost |

# Network Infrastructure and DNS

**1. Domain Name System (DNS)**

* **Function:** Translates human-readable domain names (e.g., google.com) into machine-readable IP addresses.
* **Hierarchy:**
  + **Root Name Servers (RNS):** The top of the hierarchy, responsible for delegating to Top-Level Domain (TLD) servers.
  + **Top-Level Domains (TLDs):** .com, .org, .in, etc.
  + **Name Servers:** Authoritative servers for specific domains, holding the actual IP address records (A records).
  + **Local DNS:** Cached records on a local network (e.g., your ISP's DNS server or a DNS server within your organization).

**2. Network Components**

* **Firewall:** A security device that controls network traffic based on rules (allow/deny), protecting against unauthorized access.
* **Load Balancer (LB):** Distributes incoming network traffic across multiple servers (e.g., web servers) to ensure high availability and performance.

**3. Web Communication**

* **Browser:** The client software used to access websites.
* **Web Server:** Hosts the website's files and serves them to users.
* **Application Server:** Handles the logic and processing of web applications.
* **Database Server:** Stores the data used by the application.

**4. Protocols**

* **HTTP (Port 80):** Unsecured protocol for transferring web content.
* **HTTPS (Port 443):** Secure protocol using SSL/TLS encryption for safe communication.
* **SSH (Port 22):** Secure protocol for remote access to Linux machines.
* **RDP (Port 3389):** Protocol for remote access to Windows machines.

**5. Network Packet Flow**

* **Browser:** Sends a DNS request for the hostname (e.g., google.com) to the Local DNS.
* **Local DNS:** Checks its cache. If not found, forwards the request up the DNS hierarchy.
* **DNS Resolution:** The request travels up to the Root Name Servers, then to the TLD server, and finally to the authoritative Name Server for the domain, which returns the IP address.
* **Browser:** Receives the IP address and sends an HTTP or HTTPS request to the web server at that address.
* **Firewall:** Checks the request against its rules.
* **Load Balancer:** If present, directs the request to an appropriate web server.
* **Web Server:** Responds with the requested web page.

**6. Other Concepts**

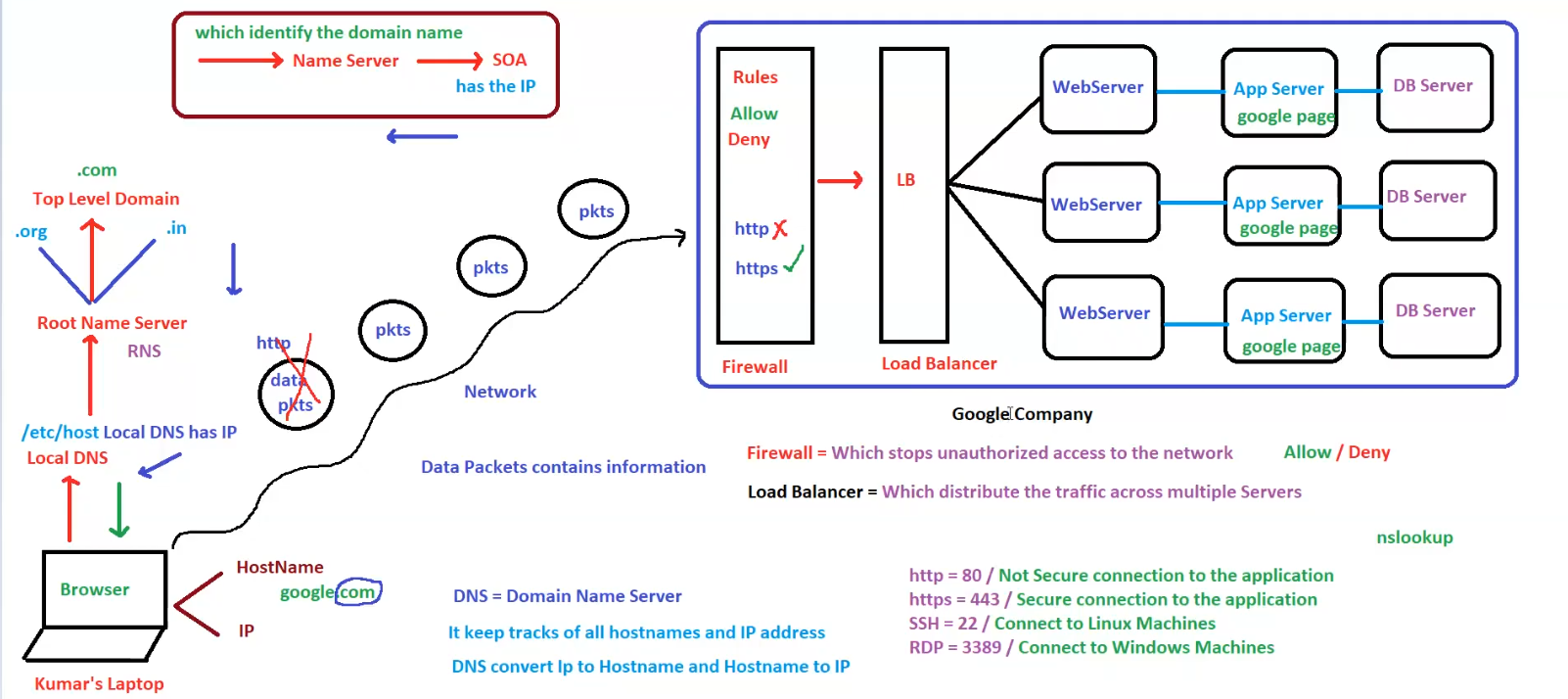
* **/etc/hosts:** A local file on a computer that can be used to override DNS mappings.
* **nslookup:** A command-line tool used to query DNS servers.

**Remember:**

* DNS is essential for navigating the internet.
* Firewalls and Load Balancers are critical components for network security and performance.
* Understanding protocols like HTTP, HTTPS, SSH, and RDP is important for secure communication.

**Additional Notes:**

* The diagram shows a simplified network architecture. Real-world networks can be much more complex.
* The "Data Packets contain information" note highlights that network traffic carries the actual data being exchanged.

****

# HTTP & HTTPS Concepts

**HTTP & HTTPS Concepts**

* **HTTP (HyperText Transfer Protocol)**
  + Default port: **80**
  + Used to transfer data between browsers and servers.
  + **Not secure** as data is transferred in **plain text**.
* **HTTPS (HyperText Transfer Protocol Secure)**
  + Default port: **443**
  + Uses **SSL/TLS certificates** for encryption.
  + **Secure** as data is encrypted.

**HTTP Status Codes**

* **404** – Page not found
* **500** – Internal Server Error
* **503** – Service Unavailable
* **200** – Page Found / Success

**Protocols & Ports**

* **Protocol Format:** protocol://domainname:portnumber
* **Examples:**
  + http://192.168.10.20:80
  + http://192.168.10.20:8080
  + http://192.168.10.20
* **Port Customization:**
  + Ports can be customized at the application level.
  + For customers, standard ports (**80 or 443**) should be used.

**Transmission Control Protocol (TCP)**

* Establishes connection between two hosts.
* Works as a **bridge** for data transfer.
* More **reliable** than UDP.

**User Datagram Protocol (UDP)**

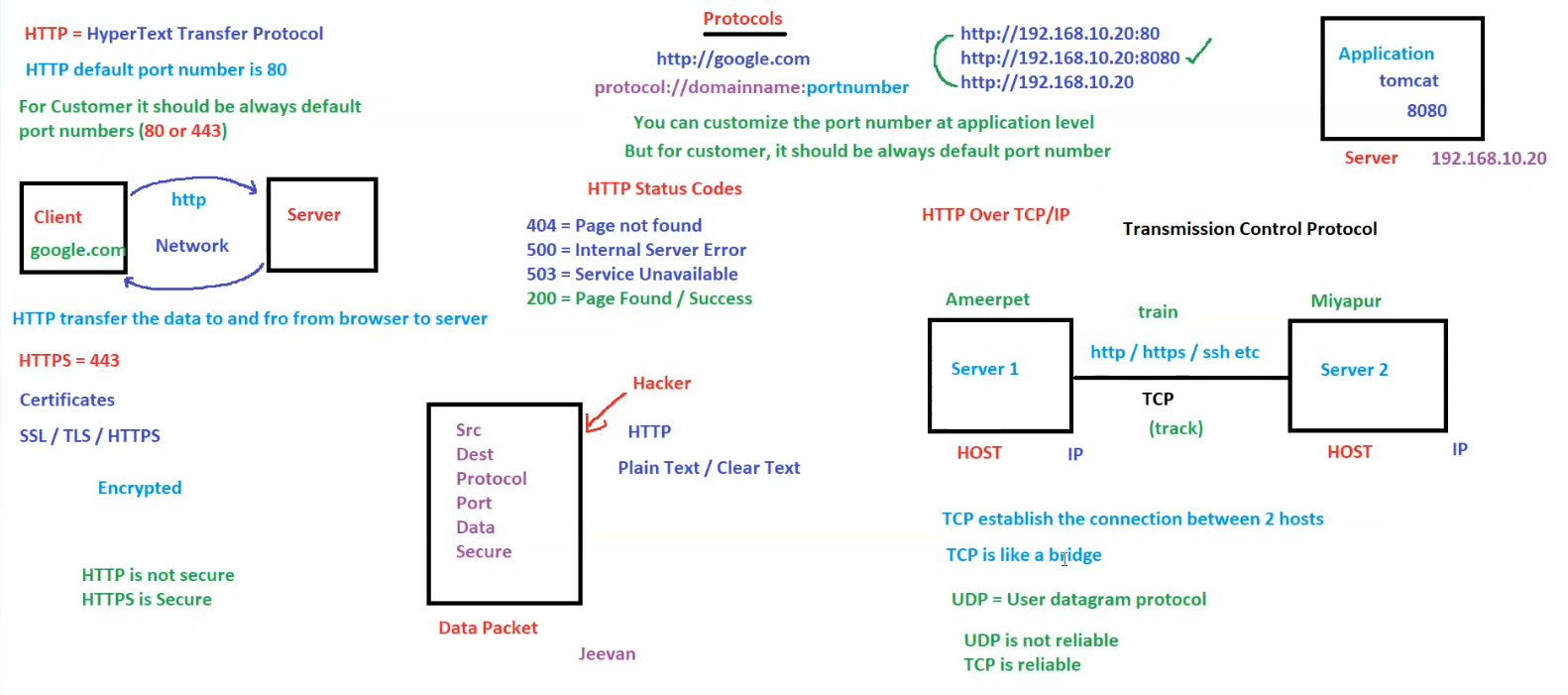
* Used for faster data transmission.
* **Not reliable** compared to TCP.

**Network Communication**

* **Client (google.com) ↔ Server** via HTTP/HTTPS.
* **HTTP over TCP/IP** ensures communication between servers.
* **Example:**
  + Server 1 (Ameerpet) ↔ Server 2 (Miyapur) using TCP.

**Security Concerns**

* **HTTP is vulnerable to hacking** as data is transferred in **plain text**.
* **Hackers can intercept data packets**.
* **HTTPS ensures encryption** and secures data.



# OSI Model (Open Systems Interconnection)

**OSI Model (Open Systems Interconnection)**

* The OSI model consists of **7 layers**, each responsible for a specific aspect of network communication.

**7 Layers of OSI Model**

1. **Application Layer**
   * Provides network services to end-users.
   * Examples: **HTTP, HTTPS**
2. **Presentation Layer**
   * Ensures data is in a readable format.
   * Examples: **SSH, RDP**
3. **Session Layer**
   * Manages and controls connections between computers.
4. **Transport Layer**
   * Ensures reliable data transfer between devices.
   * Examples: **TCP, UDP**
5. **Network Layer**
   * Handles addressing and routing of data.
   * Example: **IP (Internet Protocol)**
6. **Data Link Layer**
   * Manages data transfer between directly connected nodes.
7. **Physical Layer**
   * Concerned with the physical transmission of data (cables, signals, etc.).

**Network Communication Example**

* **Host 1 (Boy) ↔ Host 2 (Girl)**
  + Connected through **TCP or UDP** for communication.
  + This represents two devices communicating over a network.

**Key Protocols**

* **TCP (Transmission Control Protocol)**
  + Reliable, connection-oriented communication.
* **UDP (User Datagram Protocol)**
  + Faster but **unreliable** communication.

