**Architectural**

**DNS**

A computer with a globe and a computer with a server

AI-generated content may be incorrect.

POXY SERVER

A computer network diagram with a globe and a server

AI-generated content may be incorrect.

Latency

A green and black map

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Load balancer

A diagram of a load balancer

AI-generated content may be incorrect.

Top algorithm round robin

**Let's Learn Architectural Design Patterns in Simple, Non-Technical Terms**

Imagine you're building a **restaurant**. The way you design your kitchen, staff, service, and how food is served—that's your **architecture**.

In software, an **architectural design pattern** is like a **blueprint** or **strategy** for building your app—deciding how different parts will talk to each other and work together.

**🍰 1. Layered Pattern (Cake Layers)**

Think of a **cake with layers**:

* **Top layer** (Presentation): Looks good, for the customer (website/app screen)
* **Middle layer** (Logic): Decides what to do (cooking staff)
* **Bottom layer** (Data): Keeps the records (kitchen storage)

👉 Example: Your **online shopping app**

* Top layer: UI shows your products
* Middle layer: Business rules (discounts, stock)
* Bottom layer: Database stores product details

✅ Easy to manage, test, and update each layer separately.

**🍽️ 2. Client-Server Pattern (Waiter & Kitchen)**

**A screen shot of a computer screen

AI-generated content may be incorrect.**

* **Client** is you (the customer).
* **Server** is the kitchen.

You (client) give an order. The kitchen (server) prepares and serves it.

👉 Example: Using a mobile app to order pizza.

✅ This pattern is used in almost all web apps—your phone talks to a server.

**🧩 3. Microservices Pattern (Many Small Kitchens)**

Instead of one big kitchen, imagine:

* One kitchen for pasta 🍝
* One for desserts 🍰
* One for drinks 🥤

Each small kitchen (microservice) handles its own job and can work independently.

👉 Example: Amazon has:

* A **user** service
* An **order** service
* A **payment** service

✅ If the payment system crashes, the others still work!

**🛎️ 4. Event-Driven Architecture (Bell Notification)**

In this restaurant:

* The bell rings when food is ready 🍛🔔
* The waiter picks it up when the bell rings.

Here, the bell is the **event**, and the waiter **reacts**.

👉 Example: You get a text when your online order is out for delivery.

✅ Fast and automatic response to actions.

**🧑‍🍳 5. MVC – Model View Controller (Restaurant Staff)**

**A person pointing at a diagram

AI-generated content may be incorrect.**

* **Model** = The ingredients/fridge (data)
* **View** = The food on the plate (what customer sees)
* **Controller** = The chef (decides how to cook)

👉 Example: On Instagram:

* Data = your photo (Model)
* What you see = your feed (View)
* App logic = what happens when you click a button (Controller)

✅ Keeps things organized between logic, data, and display.

**💨 6. Serverless Architecture (Robot Kitchen)**

You don’t hire a chef. Instead, a **robot chef** comes **only when needed**, cooks, and disappears.

👉 Example: You upload a file, and a robot (code) resizes it automatically.

✅ You save money because you're not always running a kitchen!

**🧾 Summary (in real life terms):**

| **Pattern** | **Real-Life Example** | **Why It's Useful** |
| --- | --- | --- |
| Layered | Cake with layers | Organized and clean |
| Client-Server | Waiter and Kitchen | Central place to serve |
| Microservices | Many small kitchens | Independent and scalable |
| Event-Driven | Bell rings when food is ready | Fast reaction to events |
| MVC | Chef, Ingredients, Plate | Clean separation |
| Serverless | Robot chef on-demand | No maintenance needed |

11. What are some common security best practices you follow in your architecture?

**ChatGPT said:**

Below is the solution Roy...

**🔐 Common Security Best Practices:**

1. **🔑 Use HTTPS** – Encrypt all data in transit.
2. **🔐 Store secrets securely** – Use AWS Secrets Manager or environment variables.
3. **🧼 Validate inputs** – Prevent SQL/NoSQL injection, XSS, etc.
4. **🔒 Use JWT or OAuth** – For secure authentication and authorization.
5. **📦 Least privilege access** – Limit permissions for users/services.
6. **📊 Enable logging & monitoring** – Detect suspicious activity early.
7. **🚫 Disable unused services/routes** – Reduce attack surface.
8. **📅 Keep dependencies updated** – Patch known vulnerabilities.

**Top 20 Architectural Design Pattern Interview Questions with Answers (Non-Tech Friendly)**

These answers are simplified so even someone without a tech background can understand and explain them with confidence.

**🔟 Basic Conceptual Questions**

**1. What is an architectural design pattern?**  
👉 **Answer:**  
It’s like a blueprint for building apps. It helps organize parts of an app—like where to put the UI (what users see), the logic (rules), and data (storage)—so everything works smoothly together.

**2. What is the difference between a design pattern and an architectural pattern?**  
👉 **Answer:**

* **Design Pattern**: Deals with smaller parts of code (e.g., how one class talks to another).
* **Architectural Pattern**: Focuses on the whole system (like how different parts of an app connect).

**3. Explain Layered Architecture with an example.**  
👉 **Answer:**  
It’s like a 3-layer cake:

* Top = UI (what user sees)
* Middle = Logic (rules)
* Bottom = Data (database)

Example: In a banking app:

* Top shows account details
* Middle checks if your transaction is valid
* Bottom stores your account info

**4. What is MVC architecture and where is it used?**  
👉 **Answer:**  
MVC = Model, View, Controller

* **Model**: Data (like your photos)
* **View**: UI (what you see)
* **Controller**: Logic (what happens when you click)

Used in web apps like Instagram.

**5. Difference between Monolithic and Microservices architecture?**  
👉 **Answer:**

* **Monolithic**: One big app—hard to manage as it grows.
* **Microservices**: App is split into small parts (login, payment, search), each working independently.

**6. Advantages and Disadvantages of Microservices?**  
👉 **Answer:**  
**Pros**:

* Easy to scale
* Teams can work independently
* If one part fails, others still work

**Cons**:

* More complex to manage
* Needs good communication setup between parts

**7. What is Client-Server architecture?**  
👉 **Answer:**

* **Client** (like a mobile app) sends a request
* **Server** (backend system) responds

Example: Swiggy app (client) sends order → Server processes it → You get confirmation.

**8. What is Event-Driven Architecture?**  
👉 **Answer:**  
Something happens (event) → system reacts.

Example: You upload a photo (event) → It auto-resizes and sends a notification.

**9. Explain Service-Oriented Architecture (SOA).**  
👉 **Answer:**  
App is made of services (login, order, payment), which talk using a standard language. Older style than microservices but similar in idea.

**10. What is Serverless Architecture?**  
👉 **Answer:**  
You don’t manage any servers. Cloud provider runs your code when needed.

Example: You upload a file → It gets resized without a running server.

**🔟 Scenario-Based & Practical Questions**

**11. You're building a large e-commerce site. What architecture would you choose?**  
👉 **Answer:**  
**Microservices** – each service handles orders, users, payments separately. Easy to scale, maintain, and deploy.

**12. How would you break a monolithic app into microservices?**  
👉 **Answer:**

1. Identify key features (login, cart, checkout)
2. Separate them into small services
3. Make them talk via APIs

**13. How do microservices talk to each other?**  
👉 **Answer:**  
They use APIs (like REST or GraphQL) or messaging systems (like RabbitMQ or Kafka) to send/receive data.

**14. How to keep data consistent across microservices?**  
👉 **Answer:**

* Use **shared IDs** or transactions
* Use **event-driven** updates (when one service updates, it informs others)

**15. What are the challenges of Microservices?**  
👉 **Answer:**

* Harder to test
* Need good API design
* Need monitoring tools to track services
* Data sync between services

**16. How to design a scalable system?**  
👉 **Answer:**  
Use:

* **Microservices** (for modularity)
* **Caching** (to reduce load)
* **Load balancers** (to distribute requests)

**17. What is an API Gateway in Microservices?**  
👉 **Answer:**  
It’s like a receptionist. All requests go to the gateway, which forwards them to the right service (login, search, etc.).

**18. How does caching improve architecture?**  
👉 **Answer:**  
It stores frequent data temporarily, so the system doesn’t fetch it again and again. This makes apps faster and reduces load.

**19. Difference between horizontal and vertical scaling?**  
👉 **Answer:**

* **Vertical**: Add more power to 1 server (CPU, RAM)
* **Horizontal**: Add more servers

Microservices work better with **horizontal** scaling.

**20. How would you design WhatsApp or Instagram (basic idea)?**  
👉 **Answer:**

* **Client-Server** model for sending messages
* **Event-driven** for notifications
* **Microservices** for chat, media, and user management
* **Caching** for fast profile/photo loading
* **Serverless** for on-demand tasks like image processing

**Main Building Blocks of Cloud Architecture:**

| **Component** | **Role** | **Real-life Analogy** |
| --- | --- | --- |
| **Frontend** | What the user sees (like websites/apps) | The receptionist’s desk |
| **Backend** | Servers, databases, storage, etc. | The inner working of the office |
| **Cloud Storage** | Stores files, images, backups | Your filing cabinet in the cloud |
| **Database** | Stores structured data | A cloud-based Excel sheet |
| **Networking** | Connects all parts together | Internet cables and routers |
| **Cloud Services** | Tools like AI, email, notifications | Specialized departments |
| **Security** | Protects data, access | Security guards, locks, passwords |

SYSTEM DESIGN

Verical scalling and horizontal scalling

A screenshot of a computer

AI-generated content may be incorrect.