Cognixia

Scaling Systems for Growth

By: Ahmad

1

Cognixia

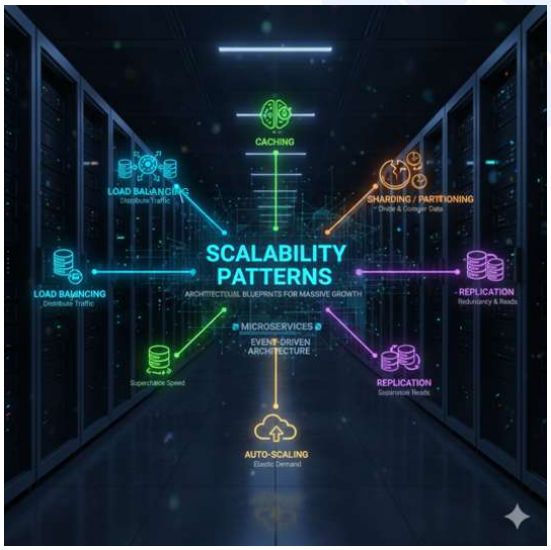
Scalability Pattern

www.cognixia.com

2

Scalability Patterns

- **Scalability Patterns** are **reusable design methods** that help a **system handle more users**, **more traffic**, and **more data** without **slowing down** or **failing.**
- They are **proven ways** to make software grow smoothly as demand increases.
- **Why do we need them?**
  - Because **when systems get bigger**, they **face**:
    - Performance issues
    - High traffic spikes
    - Database overload
    - Slow response times
- Scalability patterns **help architects** solve these problems **efficiently** and **predictably**.



www.cognixia.com

3

Core Scalability Patterns

- **Scalability patterns focus** on **distributing load** and **decoupling system components**.
- Here are the **most fundamental ones**:
  - **Load Balancing (Traffic Distribution)**
    - Using a **specialized device** or **software** (*Load Balancer*) **to distribute incoming network traffic** across **multiple application servers**.
    - Benefit
      - ✓ **Prevents any single server** from **becoming overwhelmed**, **ensuring maximum throughput** and **high availability**.
      - ✓ If **one server fails**, the load balancer **routes traffic to the healthy ones**.
  - **Caching (Speed and Database Offload)**
    - **Storing** the **results** of **frequently requested** or **expensive operations** (*like a complex database query or a rendered webpage*) in a **faster**, temporary **storage layer** (*like Redis or Memcached*).
    - Benefit
      - ✓ **Reduces latency** for **users** and **significantly reduces** the **load** on the slower primary data store (database).
  - **Sharding or Partitioning (Data Distribution)**
    - The **process of splitting** a **single, large logical database** or dataset **into multiple**, smaller, **independent databases**, **called shards or partition**s, each running on its own server.
    - Benefit
      - ✓ **Allows** the **database to scale horizontally**, **overcoming** the storage and **processing lim**its of a **single machine**.

www.cognixia.com

4

Core Scalability Patterns … continue

Cognixia

- **Replication (Redundancy and Read Scaling)**
  - Creating and maintaining **multiple copies** of **data** or services **across different servers**.
  - Benefit
    - ✓ **Improves fault tolerance** (*if one copy fails, others exist*) and read scalability by **allowing read requests** to be **served** by the **copies** (*known as read replicas*) **instead** of **overloading** the primary **server**.

- **Auto-Scaling (Elasticity)**
  - Automatically **adjusting** the **number of computing resources** (*servers*) in use based on the current load or demand.
  - Benefit
    - ✓ **Ensures optimal performance during peak hours** by scaling up and saves cost during quiet hours by scaling down.

5

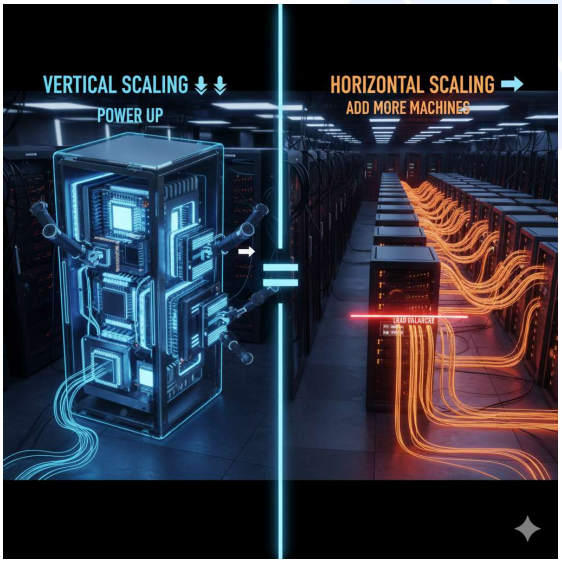
Cognixia

Scaling

6

What is Scaling

- Scaling is the ability of an IT system or business process to **handle an increasing amount of work** (more users, data, or transactions) by adding resources efficiently.
- It ensures that as demand grows, the system remains fast, available, and reliable.
- Auto scaling involves **two main concepts**:
  - **Scaling Types** (the direction of change)
  - **Scaling Methods** (the trigger for the change).
- There are **two main ways** to increase a system's capacity in a technical context:
  - **Vertical** (Scaling Up)
  - **Horizontal** (Scaling Out)



www.cognixia.com

7

Vertical Scaling (Scale-Up)

- Vertical scaling, or **scaling up**, means **increasing the capacity** of a **single machine**.
- You **add more resources**—like CPU, RAM, or faster storage (SSD)—to an **existing server**.
- **Limit**: It's **limited** by the **maximum hardware specifications** available for a **single physical machine**.
- **Availability**: **Creates** a **Single Point of Failure** (SPOF).
- **Simplicity**: It is generally **simpler to manage** since you only have one machine to maintain and don't need complex distributed logic.
- **Example**
  - **Upgrading your laptop** from **8 GB** → **32 GB** RAM to run more apps.
- **When to Use Vertical Scaling**
  - **Application** is **monolithic** or tightly coupled
  - You **cannot add multiple servers**

www.cognixia.com

8

Horizontal Scaling (Scaling Out)

Cognixia

- Horizontal scaling, or **scaling out**, means **increasing capacity** by **adding more machines** to the system.
- You **add more servers** (*or virtual instances/containers*) and **distribute** the **workload among them**.
- **Limit**: **Virtually limitless** (*you can theoretically keep adding servers*).
- **Availability/Fault Tolerance**: If **one server fails**, the load balancer **automatically directs traffic** to the **remaining healthy** servers, meaning no downtime for users.
- **Complexity**: **Requires architectural changes** to ensure the **application is stateless** (*so any server can handle any user request*) and requires managing **data consistency** across **multiple nodes**.
- **When to Use Horizontal Scaling**
  - Microservices-based architecture
  - **Stateless applications**
  - **High traffic web and mobile apps**
  - **Dynamic workloads**

www.cognixia.com

9

Auto-Scaling

Cognixia

- Its **main purpose** is to ensure that **applications maintain consistent** p**erformance** and **availability** by always having the necessary resources, while **simultaneously optimizing costs** by **avoiding** the o**ver-provisioning** of resources **during low-demand periods**.
- The goal is to **always match capacity with demand**:
  - **When traffic spikes**: **Automatically add resources** to prevent slowdowns.
  - **When traffic drops**: **Automatically remove resources** to save money.
- In a **public cloud** (like AWS or Azure), **auto scaling** is a **service they provide and manage**.
- **On-premises** auto scaling is when you **build this same capability** in your **own data center**.
- It's **more complex** because you have to manage all the moving parts.
- The **most common way** to do this today is by using **Kubernetes** (*often called K8s*), which is the software platform used to manage application containers.

www.cognixia.com

10

Cognixia

Performance Analysis

11

What is Performance Analysis

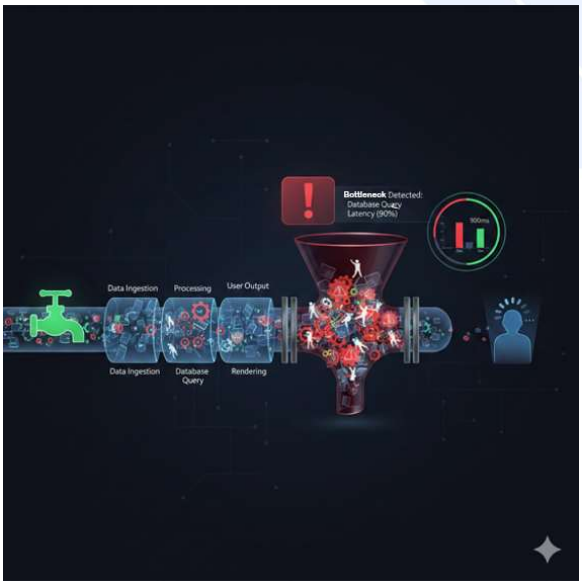
Cognixia

- Performance Analysis is the **process** of **measuring how well** a **system performs** and finding **what slows it down**.
- Its **primary goal** is to **ensure** the **system meets required performance objectives** (*like fast response times*) and to **identify** and **resolve** the **bottlenecks** that limit its capacity.
- It helps identify:
  - Bottlenecks
  - Latency issues
  - Slow queries
  - High CPU/RAM usage
  - Network delays
  - Scaling problems
- Goal: Make the **system faster**, **stable**, and **efficient** under **real load**.

12

Bottlenecks Analysis

- The core objective of this **analysis** is to find and **eliminate bottlenecks**—the **components** that **constrain** the **overall throughput** or **latency of the system**.
- A **bottleneck** is any component that slows down the entire system — like the slowest person in an assembly line.



www.cognixia.com

13

Detecting Bottlenecks

Layer / Component	Symptoms (What You See)	How to Detect (Tools & Techniques)
Application Layer	**Slow APIs**, **high CPU**, hangs	**APM tools** (Datadog, NewRelic), **CPU/Memory** profiling, **Logs**, **Traces**
Database Layer	**Slow queries**, locks, DB CPU 80–100%	Query execution plans, **DB monitoring** dashboards, **slow query logs**
Storage/I/O Layer	**File uploads slow**, high I/O wait	**Disk IOPS metrics**, CloudWatch, Storage performance insights
Network Layer	**High latency**, packet drops	**Ping tests**, **traceroute**, VPC **Flow Logs**, Network dashboards
Messaging Queue	**Queue backlog increasing**	**Queue** depth **metrics**, consumer lag monitoring, CloudWatch/Prometheus
Third-Party APIs	**External call latency** spikes	**API Gateway logs**, circuit-breaker metrics, distributed tracing
Front-End/UI	**Slow page load**, high Time-to-First-Byte	**Web Vitals**, Chrome DevTools, Lighthouse, RUM tools

www.cognixia.com

14

Fixing Bottlenecks

Layer / Component	Fix / Optimization Actions
Application Layer	**Add caching** (Redis), optimize code loops, async processing, **add Auto Scaling**, refactor heavy logic
Database Layer	**Add indexes**, optimize queries, **add read replicas**, increase connection pools, **sharding/partitioning**
Storage/I/O Layer	**Move to SSD**, **increase provisioned IOPS**, use object storage (S3), enable **caching/CDN**
Network Layer	**Reduce cross-region calls**, use private links, **increase bandwidth**, enable API caching
Messaging Queue	**Add more consumers**, increase batch size, change visibility timeout, auto-scale workers
Third-Party APIs	**Add circuit breaker**, retry with backoff, **local caching**, queue requests asynchronously
Front-End/UI	**Compress images**, minify JS/CSS, enable CDN, lazy load assets, optimize bundle size
Microservices	**Reduce network hops**, optimize service boundaries, tune gRPC/HTTP calls, add bulkheads

15

Optimizing Delivery

www.cognixia.com

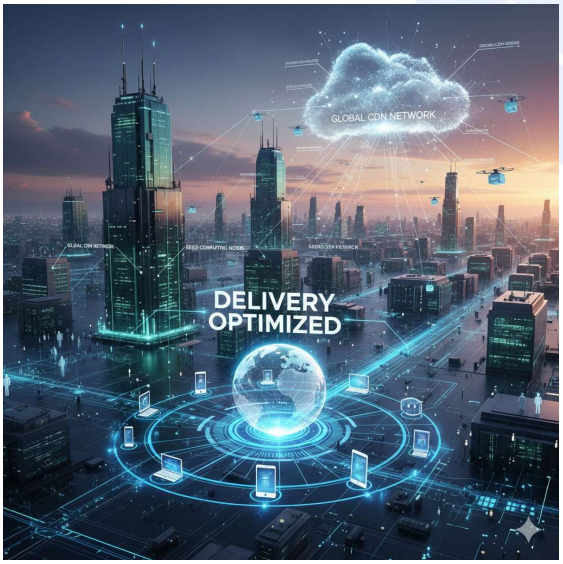
16

Optimizing Delivery

Cognixia

- Optimizing delivery with **Content Delivery Networks (CDNs)** and **Edge Computing** involves **moving data** and **computation geographically closer to the end-users** to dramatically reduce latency, decrease network congestion, and improve the overall user experience.

- **Content Delivery Network (CDN)**
  - A CDN is a **geographically distributed network of proxy servers** and data centers.
  - It focuses primarily on the **delivery of content**.



www.cognixia.com

17

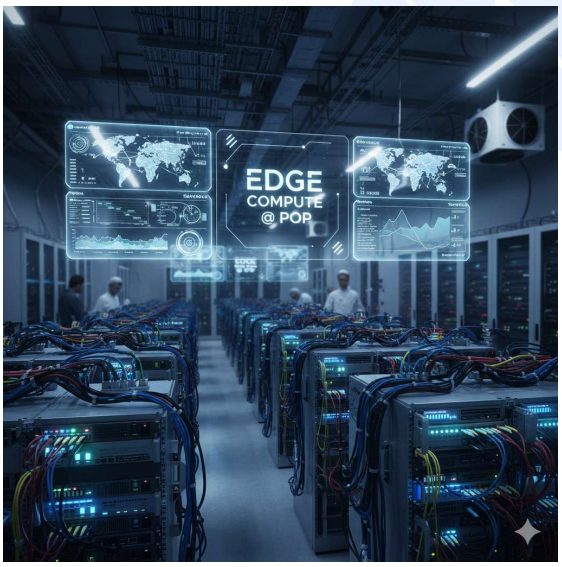
Edge Computing

Cognixia

- Edge Computing is an **architectural approach** that **extends** the concept of a **CDN** by **moving application logic and computation—not just cached content**—to the **edge** of the **network**, even **closer to** the **user** or the data source.

- **Computation at the Edge**
  - Unlike a traditional CDN which primarily caches static files, **Edge Computing runs small applications** or functions (*known as "serverless functions" or "Edge functions"*) **on the PoPs**.

- **Preprocessing Data**
  - It can **handle tasks** that **require low latency**, such as data validation, basic authentication, formatting data, or simple business logic, before the request ever reaches the main cloud data center.



www.cognixia.com

18

Edge Computing … continue

- **Key Benefits**
  - **Ultra-Low Latency**
    - **Processing** is done immediately **at the edge**, making it vital for **time-sensitive applications**.
  - **Bandwidth Efficiency**
    - Edge devices can filter, **aggregate**, and **analyze data locally**, sending only relevant information back to the central cloud, saving bandwidth.
  - **Enhanced Security**
    - Edge functions can be used to **scrub incoming requests** for **malicious traffic** before they reach the core infrastructure.

- **Industry Example**
  - **Industrial IoT (Internet of Things) Manufacturing**
    - **Sensors** on a **factory floor generate massive amounts** of **data** in **real-time**.
    - **Instead of sending all** that **raw data** to a **central cloud** (which would be slow and expensive), an **edge gateway** in the **factory processes** the **data instantly** to detect equipment anomalies.
    - This immediate, **local processing allows** for **rapid shut-down** of a **malfunctioning machine** (critical action) long **before** the **central cloud could ever receive** and **analyze the alert**.

www.cognixia.com

19

20