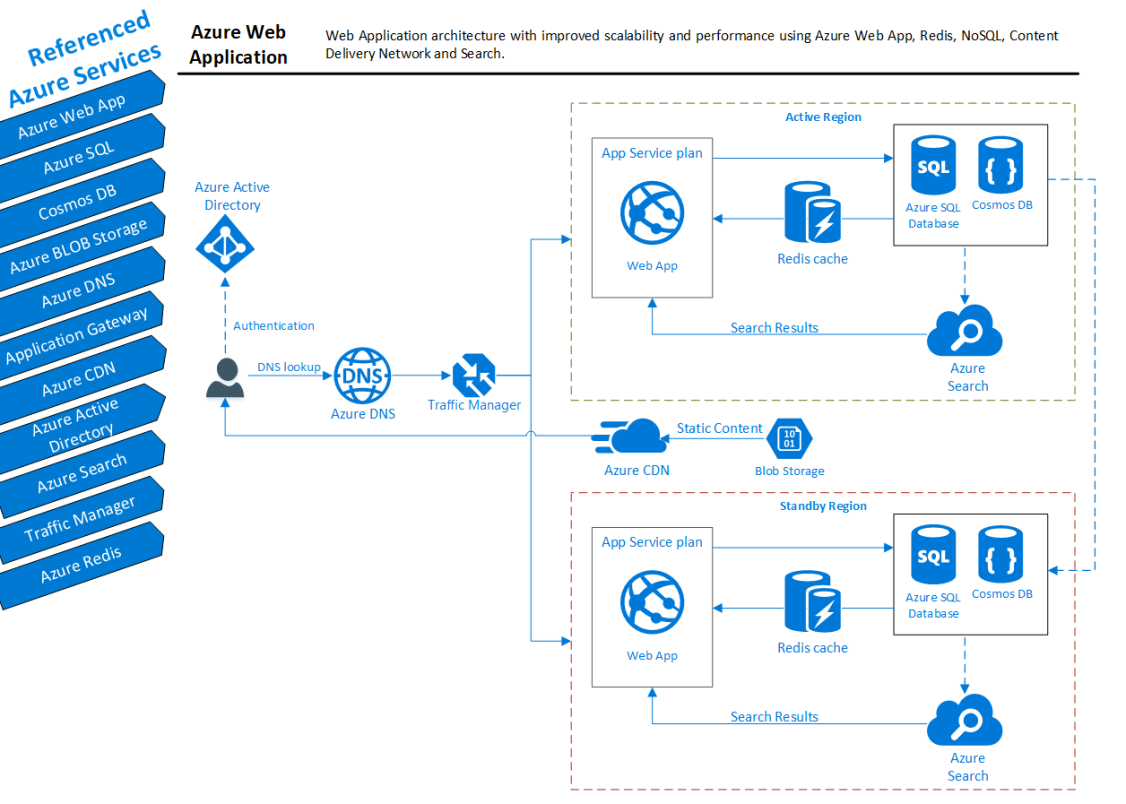
**System Design Blueprint**

**https://blog.devgenius.io/system-design-blueprint-the-ultimate-guide-e27b914bf8f1**

Developing a robust, scalable, and efficient system (DNS, load balancing, API Gateway, Caching on Application, Database)





|  |  |
| --- | --- |
| Principles of System Design | |
| Modularization | Dividing the system into smaller, manageable modules help reduce complexity, improve maintainability, and increase reusability. |
| Abstraction | Hiding the implementation details and showing only the essential features helps simplify complex systems and promote modularity. |
| Layering | Organizing the system into layers, each layer providing a specific set of functionalities promotes the separation of concerns and enhances maintainability. |
| Scalability | Design systems to handle the increased load by adding more resources (horizontal scaling) or optimizing the system’s capacity (vertical scaling). |
| Performance | Optimizing the system’s response time, throughput, and resource utilization is crucial for a successful design. |
| Security | Ensure the system’s confidentiality, integrity, and availability by implementing proper security measures and practices. |
| Fault Tolerance and Resilience | Design systems to withstand failures and recover gracefully from errors, ensuring reliability and availability. |

Key Component of system design

|  |  |
| --- | --- |
| DNS (Domain Name System) | It translates human-readable domain names (e.g., [www.example.com](http://www.example.com/)) into IP addresses, allowing users to access websites and services more efficiently. |
| Load Balancing | distributing network traffic across multiple servers |
| API Gateway | API Gateway is a server that acts as an intermediary between clients and microservices in a distributed system. It manages and routes requests, enforces security policies, and may provide additional features such as caching, logging, and monitoring. |
| Content Delivery Network (CDN) | CDNs cache content on edge servers close to end-users, improving the system’s performance and reducing the load on origin servers. |
| Message Queue | Message queues facilitate communication between distributed system components by temporarily storing messages in a queue. They enable asynchronous processing and help decouple components, improving the system’s scalability and fault tolerance |
| Communication Protocols | Different communication protocols are used in system design, such as HTTP/HTTPS, WebSocket, and gRPC. |
| Cache | Caching is a temporary technique used to store copies of data, allowing for faster retrieval in future requests. It helps reduce latency, server load, and bandwidth consumption. Popular caching mechanisms include in-memory caching, distributed caching, and browser caching. |
| DatabaseChoosing the appropriate database for a system depends on data structure, scalability, consistency, and latency. | ommon database types include relational databases (e.g., MySQL, PostgreSQL), NoSQL databases (e.g., MongoDB, Cassandra), and NewSQL databases (e.g., Cockroach DB, Google Spanner). |
| Replication Techniques | Replication is maintaining multiple copies of data across different nodes for increased reliability, availability, and fault tolerance. Standard replication techniques include synchronous replication, asynchronous replication, and semi-synchronous replication. |
| Distributed Unique ID Generation | Creating unique identifiers in a distributed system can be challenging but is essential for maintaining data consistency and integrity. |

# Common Components in System Design

|  |  |
| --- | --- |
| Payment Service | Payment services handle transactions between customers and businesses. Integrating a reliable payment service is crucial for e-commerce and subscription-based platforms. Popular payment service providers include Stripe, PayPal, and Square. These services usually provide APIs to facilitate secure transactions and manage recurring payments, refunds, etc |
| Notification | Notification services keep users informed about updates, alerts, and important information. These services can deliver notifications through various channels, such as email, SMS, and push notifications. Examples of notification service providers include Firebase Cloud Messaging (FCM), Amazon Simple Notification Service (SNS), and Twilio. |
| Search | Integrating a powerful search component is essential for systems with large amounts of data or content. A search service should provide fast, relevant, and scalable search capabilities. Elasticsearch, Apache Solr, and Amazon CloudSearch are popular choices for implementing search functionality. These services typically support full-text search, faceted search, and filtering, enabling users to find the information they’re looking for quickly and efficiently. |

# Best Practices for System Design

|  |  |
| --- | --- |
| Requirement Gathering |  |
| Design Patterns |  |
| Documentation | Document your design decisions, assumptions, and rationale to ensure better communication and maintainability. |
| Iterative Design | Refine your Design through multiple iterations and feedback, allowing it to evolve and improve. |
| Testing and Validation | Validate your Design against the requirements and conduct testing to identify and address potential issues. |

## 