System Design Engineering:

How Senior Engineers Architect Large Scale Applications

Subtitle: Learn how experienced engineers design scalable, reliable, and maintainable systems used by millions of users worldwide.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Most beginner developers focus on:
• writing features
• learning frameworks
• building projects

Senior engineers think differently.

They focus on:
• scalability
• architecture
• reliability
• performance
• maintainability

At small scale:
• almost any architecture works

At large scale:
• poor design destroys systems quickly

This is why:

System Design Engineering

became one of the most valuable skills in modern software engineering.

Large companies evaluate system design heavily during:
• senior interviews
• backend engineering roles
• infrastructure positions

because designing scalable systems is far harder than writing basic code.

This PDF explains:

• how system design actually works
• how engineers think about scalability
• how large applications are architected
• what separates beginner thinking from engineering thinking

Chapter 1: What System Design Actually Means

System design means:

planning how software systems will operate at scale.

System Design Includes

• architecture decisions
• infrastructure planning
• database design
• scaling strategies
• communication systems
• reliability engineering

Important Truth

System design is mostly:
• problem solving through tradeoffs

Chapter 2: Why Architecture Matters

Small applications tolerate poor design.

Large applications do not.

Poor architecture causes

• downtime
• bottlenecks
• slow performance
• deployment problems
• scaling failures

Good architecture enables

• scalability
• maintainability
• operational stability

Chapter 3: Functional vs Non Functional Requirements

System design starts with requirements.

Functional Requirements

What the system should do.

Examples:
• upload videos
• send messages
• process payments

Non Functional Requirements

How the system should behave.

Examples:
• scalability
• reliability
• latency
• security

Senior engineers think heavily about:

• non functional requirements

Chapter 4: Scalability Basics

Scalability means:

handling increasing workload efficiently

Example

Application handles:
• 1,000 users today
• 10 million users later

Important Principle

Systems must scale without complete redesigns.

Chapter 5: Vertical vs Horizontal Scaling

Vertical Scaling

Increase power of one server.

Example:
• more RAM
• stronger CPU

Problem

Hardware limits eventually appear.

Horizontal Scaling

Add more servers.

Modern internet systems heavily rely on:

• horizontal scaling

Chapter 6: Load Balancers

Large systems distribute traffic using:

load balancers

Purpose

Distribute requests across servers evenly.

Benefits

• reliability
• scalability
• fault tolerance

Example Flow

Users
↓
Load Balancer
↓
Multiple Servers

Chapter 7: Databases Become Bottlenecks

At scale:
• databases often become the slowest component

Problems Include

• slow queries
• heavy traffic
• write overload
• replication delays

Engineers optimize using

- indexing
- caching
- replication
- sharding

Chapter 8: Caching Systems

Caching stores frequently accessed data temporarily.

Benefits

- lower latency
- faster responses
- reduced database load

Popular Caching Technologies

- Redis
- Memcached

Example

Popular posts served from cache instead of database repeatedly.

Chapter 9: CDN Systems

CDN means:

Content Delivery Network

Purpose

Deliver content closer to users geographically.

Benefits

- faster loading
- reduced latency
- reduced infrastructure load

Used For

- images
- videos
- static assets

Chapter 10: Database Replication

Replication copies database data across servers.

Benefits

• fault tolerance
• read scalability
• backup redundancy

Common Setup

Primary Database:
• handles writes

Replica Databases:
• handle reads

Chapter 11: Database Sharding

Sharding splits databases into smaller pieces.

Example

Users A–F → Shard 1
Users G–M → Shard 2

Benefits

• horizontal scalability
• reduced server pressure

Challenge

Distributed queries become more complex.

Chapter 12: Monolith vs Microservices

Monolith

Single application containing all logic.

Benefits

• simpler architecture
• easier debugging

Microservices

Independent services communicating together.

Benefits

• independent scaling
• team autonomy
• fault isolation

Important Engineering Truth

Microservices increase:
• operational complexity

Chapter 13: API Design in System Architecture

APIs connect system components.

Good APIs Must Be

• scalable
• secure
• maintainable
• predictable

Poor APIs create:

• long term architectural problems

Chapter 14: Asynchronous Systems

Large systems use asynchronous communication heavily.

Example

User uploads video.

System processes:
• transcoding
• thumbnails
• notifications
later asynchronously.

Benefits

• scalability
• resilience
• reduced blocking

Chapter 15: Message Queues

Message queues manage asynchronous workloads.

Popular Technologies

- Kafka
- RabbitMQ
- SQS

Use Cases

- notifications
- analytics
- background jobs
- event systems

Chapter 16: Reliability Engineering

Large systems must survive failures.

Failures Are Normal

- server crashes
- network outages
- deployment failures

Reliable systems use

- redundancy
- failover systems
- retries
- replication

Chapter 17: CAP Theorem

Distributed systems involve tradeoffs.

CAP Components

- consistency
- availability
- partition tolerance

Important Principle

No distributed system perfectly maximizes all three simultaneously during failures.

Chapter 18: Monitoring and Observability

Production systems require constant monitoring.

Engineers monitor

• latency
• traffic
• failures
• infrastructure health
• database performance

Without observability:

• debugging becomes extremely difficult

Chapter 19: Security in System Design

Security must be part of architecture from the beginning.

Common Areas

• authentication
• authorization
• encryption
• API protection
• secret management

Poor security architecture creates:

• catastrophic risks

Chapter 20: Rate Limiting

Public systems require traffic control.

Purpose

Prevent:
• abuse
• overload
• DDoS attacks

Example

Limit:
• requests per user per minute

Chapter 21: Fault Tolerance

Fault tolerance means:

systems continue functioning despite failures

Examples

• server redundancy
• database replication
• multi region infrastructure

Goal

Avoid:
• complete outages

Chapter 22: Real Time Systems

Some applications require real time communication.

Examples

• chat apps
• multiplayer games
• trading systems
• live analytics

Technologies Used

• WebSockets
• streaming systems
• event driven infrastructure

Chapter 23: Storage Systems

Large applications manage enormous amounts of data.

Types of Storage

• relational databases
• object storage
• distributed file systems
• caches

Engineering challenge:

• balancing performance and cost

Chapter 24: System Design Interviews

Large companies test:

architectural thinking

not just coding ability.

Interview Focus Areas

• scalability
• tradeoffs
• bottleneck analysis
• reliability
• communication skills

Important Insight

Interviewers evaluate:
• thinking process

more than:
• perfect answers

Chapter 25: Beginner vs Senior Engineering Thinking

Beginner

• “Feature works.”

Senior Engineer

• “Will this scale?”
• “What happens during failures?”
• “Where are bottlenecks?”
• “How maintainable is this?”

Chapter 26: The Biggest Beginner Mistake

Many beginners memorize:
• system design diagrams

without understanding:
• engineering tradeoffs

Real system design is not memorization.

It is:
• structured engineering thinking

Chapter 27: Real World Architecture Example

Client
↓
CDN
↓
Load Balancer
↓
API Gateway
↓
Microservices
↓
Caches
↓
Databases
↓
Monitoring Systems

This architecture supports

• scalability
• reliability
• fault isolation
• global traffic

Chapter 28: The Most Important System Design Principle

Every architecture decision creates:
• advantages
and:
• tradeoffs

Great engineers understand:

• no architecture is perfect

Engineering is about:

• choosing the right compromises

Chapter 29: Final Engineering Insight

System design separates:
• developers
from:
• engineers

Developers write features.

Engineers design systems that:
• survive scale
• survive failures
• evolve safely over time

The larger the system becomes:

• the more architecture matters

Key Takeaways

• System design focuses on scalability and reliability
• Large systems require thoughtful architecture decisions
• Databases and APIs become bottlenecks at scale
• Caching and CDNs improve performance significantly
• Distributed systems involve tradeoffs and operational complexity
• Monitoring and observability are critical in production systems
• Great architecture balances scalability with maintainability
• Senior engineers think about long term system evolution

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app