The System Architecture Playbook

9 components every modern product needs to scale

AI can build your POC/MVP.
But architecture makes it work in the real world.

Why Architecture Matters

LLMs make coding easy. But without proper architecture:

- X Systems crash under load
- X Features break other features
- X Slow pages = lost revenue
- X Failures are not retried
- X One failure takes everything down

Demos are easy. Systems are hard.

The good news? You can prompt AI to add these components too.

1

Object Storage

WHAT IT DOES

Store and serve *files* (images, videos, documents) at scale

WHY IT MATTERS

Without it: files bloat your app server, and you can't scale horizontally without copying files everywhere

WHEN TO USE

User uploads, media assets, backups, static files

EXAMPLE

Store user profile photos and product images — scalable, durable, cheap

PROMPT

"Store user-uploaded images in S3 and serve them via signed URLs"

Tech: S3 | GCS | Azure Blob | MinIO | Cloudflare R2

2

CDN / Edge Cache

WHAT IT DOES

A global network that serves content close to users

WHY IT MATTERS

Without it: international users wait seconds for content, and your servers get hammered

WHEN TO USE

You want your site/app fast *worldwide*

EXAMPLE

Images, videos, scripts load from the nearest edge location — plus DDoS protection

PROMPT

"Serve product images and static assets via CDN for faster page loads worldwide"

Tech: Cloudflare | Fastly | Akamai | CloudFront

3

Application Cache

WHAT IT DOES

A fast in-memory layer for frequently used data

WHY IT MATTERS

Without it: every request hits the database, pages load slowly, and DB costs spike

WHEN TO USE

When your app repeatedly reads the same data

EXAMPLE

Product lists load instantly from memory — millisecond speed, lower DB cost

PROMPT

"Cache the product catalog in Redis so listing pages load in under 100ms"

Tech: Redis | Memcached

4

Queue

Message Queue

WHAT IT DOES

Do work *later* or in the *background* so users don't wait

WHY IT MATTERS

Without it: users wait for slow tasks, failures aren't retried, and errors cascade

WHEN TO USE

Slow tasks, background jobs, retries, throttling

EXAMPLE

Send receipt email *after* checkout completes — user sees instant confirmation

PROMPT

"Send order confirmation emails in a background queue with retry on failure"

Tech: SQS | RabbitMQ | Redis Streams | Kafka

5

Load Balancer

WHAT IT DOES

Shares traffic across multiple servers

WHY IT MATTERS

Without it: one server gets overwhelmed, no failover, and your app goes down

WHEN TO USE

High traffic, multiple servers, need uptime

EXAMPLE

Flash sale: traffic automatically spreads across servers — auto-bypass failures

PROMPT

"Add a load balancer so the app stays up during flash sales and traffic spikes"

Tech: Nginx | HAProxy | AWS ALB | Cloudflare LB

6

Scheduler

Cron Jobs

WHAT IT DOES

Runs tasks automatically on a schedule

WHY IT MATTERS

Without it: someone forgets, tasks run late or never, and data becomes stale

WHEN TO USE

Daily/weekly tasks, cleanup, reporting

EXAMPLE

Send daily sales summary every morning at 7am — reliable, zero manual work

PROMPT

"Run a daily sales report at 7am and email it to the team automatically"

Tech: Cron | EventBridge | Kubernetes CronJob | Airflow

7

Containers + Orchestration

WHAT IT DOES

Containers = run anywhere

Orchestration = scale + self-heal automatically

WHY IT MATTERS

Without it: "works on my machine" bugs, manual scaling, and risky deployments

WHEN TO USE

Deploying at scale, multi-server environments

EXAMPLE

Launch new versions with zero downtime — auto-scaling, auto-restart, predictable deployments

PROMPT

"Containerize the app and auto-scale based on CPU usage with zero-downtime deploys"

Tech: Docker | Podman | Kubernetes | AWS ECS/EKS | GKE

8

API Gateway

WHAT IT DOES

One secure entry point to all backend services

WHY IT MATTERS

Without it: inconsistent security, no rate limiting, and APIs are hard to manage

WHEN TO USE

Multiple APIs, authentication, rate limits

EXAMPLE

Mobile app → Gateway → correct service (with API keys + routing)

PROMPT

"Route all mobile and web API requests through a gateway with JWT auth and rate limiting"

Tech: Kong | Traefik | Apigee | AWS API Gateway

9

Pub-Sub

Publish / Subscribe

WHAT IT DOES

1 event triggers many reactions

WHY IT MATTERS

Without it: hardcoded integrations, tight coupling, and adding features breaks things

WHEN TO USE

Multiple systems need to respond to the same action

EXAMPLE

User signs up → CRM, analytics, and marketing all update automatically

PROMPT

"When a user signs up, notify CRM, analytics, and welcome email services via pub-sub"

Tech: Kafka | SNS | Google Pub/Sub | NATS

Architecture Checklist

Before you ship, ask:

- ✓ Does slow work run in a queue?
- ✓ Does one event trigger multiple systems?
- ✓ Do we cache expensive operations?
- ✓ Do global users load content quickly?
- ✓ Can we survive a server failure?
- ✓ Do we have one secure API entry?
- ✓ Are files stored in object storage?
- ✓ Are recurring jobs automated?
- ✓ Can we scale without manual work?

The Bottom Line

These 9 components turn "code" into a scalable, reliable product.

- **Day 1:** Object Storage
- **Early:** CDN, Application Cache
- **Growth:** Queue, Load Balancer
- **Scale:** Scheduler, Containers + Orchestration
- **Maturity:** API Gateway, Pub-Sub

AI can code. Architecture makes it real.

Let's Connect

What component do you wish you'd added earlier?

> Follow for more system design insights

> Repost to help others

> Save this for future reference

#SystemDesign #Architecture #SoftwareEngineering #AI