Introduction: from Monoliths to Distributed Systems

Servicios y Aplicaciones Distribuidas

Welcome and Objectives

- Set expectations and outcomes for the course.
- Understand monoliths: strengths, weaknesses, and limits.
- Recognize motivations for distributed architectures.
- Preview the course structure and practical focus.

What Is a Monolith?

- Single deployable application containing UI, business logic, and data access.
- All parts share the same runtime and are released together.
- Optimized for simplicity and speed in early product stages.

Monolith Deployment Model

- All-or-nothing deployments: even small changes ship the whole app.
- Build, test, and release pipelines handle one artifact.
- Acceptable with compact teams and limited feature sets; problematic at scale.

Why Monoliths Work Early

- Rapid iteration: one repo, one CI/CD pipeline, one artifact.
- Simple onboarding and local development close to production.
- End-to-end tests run in a single environment.

Modular Monoliths

- Well-defined internal modules with clear interfaces.
- Single compilation and deployment unit for operational simplicity.
- Good stepping stone toward future extractions.

Stress Signals in Monoliths

- Eroding module boundaries and increasing merge conflicts.
- Longer builds and slower feedback loops.
- Coordination bottlenecks: unrelated changes delay releases.

Regression Testing Burden

- Small features trigger broad regression testing.
- Risk of breaking unrelated areas slows release cadence.
- Bundled, infrequent releases increase failure blast radius.

Single Artifact Operational Risk

- Incidents force rollbacks that revert unrelated features.
- Tight coupling across domains raises deployment risk.
- Discourages continuous delivery practices.

When Monoliths Still Fit

- Small teams and constrained domains.
- Low traffic or internal tools with modest SLAs.
- Prototypes or short-lived products.

Transition Triggers

- Sustained traffic growth and performance hotspots.
- Need for independent release cadence per domain.
- Rising incident frequency tied to deployment coupling.
- Long lead times and coordination overhead.

Distributed System: Definition

- Independent computers collaborate over a network.
- Present a single coherent system to users.
- Coordinate to share resources and tolerate partial failures.

Core Properties

- Concurrency and parallelism for throughput.
- Fault tolerance via replication and graceful degradation.
- Scalability through vertical and horizontal strategies.
- Transparency to hide distribution details from users.

Transparency in Practice

- Users should not care which node handled the request.
- Location, replication, and failure handling are invisible.
- Achieved with load balancers, caches, smart clients.

Resource Sharing & Elasticity

- Pool compute, storage, and bandwidth across nodes.
- Cloud platforms enable elastic scaling up and down.
- Optimize cost/performance by matching capacity to load.

Concurrency vs Parallelism

- Concurrency: multiple tasks progress at once (interleaving).
- Parallelism: tasks execute simultaneously on different cores/nodes.
- Introduce coordination: locks, optimistic control, idempotency.

Fault-Tolerance Mindset

- Design for failure: assume components will fail.
- Techniques: retries with backoff, circuit breakers, timeouts.
- Graceful degradation and redundancy to keep service usable.

Scalability Basics

- Vertical scaling: add CPU/RAM to a single node.
- Horizontal scaling: add more nodes/instances.
- Horizontal favored for resilience and elasticity; requires statelessness.

The Network Tax

- Distributed calls incur latency and serialization overhead.
- Potential for packet loss and retries.
- Mitigate with batching, efficient protocols, and careful API design.

Consistency Realities

- Replicated state causes conflicts and stale reads.
- Eventual consistency is common for availability and speed.
- User experience patterns: versioning, conflict resolution, read-your-writes.

Security Surface Area

- More services and endpoints increase attack surface.
- Strong authN/authZ, transport security, and secret management are mandatory.
- Adopt zero-trust networking principles.

Operational Complexity

- Requires mature CI/CD, automated testing, and IaC.
- Health checks, metrics, logs, and tracing for observability.
- On-call readiness and incident response playbooks.

Driver: Independent Releases

- Teams ship without waiting on unrelated components.
- Service boundaries map to business domains.
- Reduces coordination overhead and accelerates delivery.

Driver: Targeted Scaling

- Scale hotspots (e.g., checkout) independently of cold paths (e.g., admin).
- Aligns cost with actual demand.
- Avoids scaling the entire system unnecessarily.

Example Domain Split

- Orders, Payments, Products, Users, Notifications as separate domains.
- Each owns its storage, APIs, and scaling policies.
- Technology aligns with business boundaries.

Migration Path

- Start from modular monolith and identify seams.
- Extract the most independent or painful domains first.
- Establish platform capabilities: gateway, discovery, observability.

Trade-Offs to Acknowledge

- Agility and resilience vs added latency and coordination.
- Operational costs rise with more moving parts.
- Architecture is economics: pay costs to unlock benefits.

Key Takeaways (1)

- Monoliths are pragmatic and effective early on.
- They become a liability with growth and coordination bottlenecks.
- Recognize stress signals and plan ahead.

Key Takeaways (2)

- Distributed systems enable independence and scaling.
- They introduce latency, consistency, security, and operational challenges.
- Discipline and tooling are essential to succeed.

Common Misconceptions

- “Microservices are faster” — network overhead is real.
- “Use many languages” — polyglot increases ops and hiring costs.
- “More services is better” — follow domains and team boundaries.

Looking Ahead

- Next: define microservices precisely and contrast with distributed systems.
- Discuss organizational and technical implications.
- Identify when NOT to adopt microservices.

Discussion

- Where has a monolith been a bottleneck in your experience?
- Which domains would benefit most from independent deployments?

Closing

- You now have the vocabulary for why distributed systems exist.
- Keep trade-offs in mind—they guide the rest of the course.