The Evolution of Cloud Computing

Part 1

The Evolution of Cloud Computing

- From shared mainframes → virtualized data centers → cloud-native platforms
- Goal today: understand *why* cloud looks the way it does

Session Roadmap

- The “shared computing” lineage: mainframes and time-sharing
- The shift to distributed computing: PCs and client–server
- Why on-prem got hard (cost, scale, operations)
- The enablers: virtualization, commodity hardware, automation
- The outcomes: IaaS, containers, orchestration, cloud economics

Overview: Cloud as an Evolution, Not an Invention

- Cloud combines earlier ideas: sharing, abstraction, automation, metering
- Enabled by fast networks + cheap hardware + software-defined infrastructure
- Shifts the default: *buy capacity* → *rent capability*
- Changes how teams build: self-service, APIs, rapid iteration
- Sets up service models (IaaS/PaaS/SaaS) and deployment models (public/private/hybrid)

What People Mean by “Cloud” (In Practical Terms)

- On-demand resources: compute, storage, networking available quickly
- Elasticity: scale up/down with workload rather than fixed capacity
- Pay-for-use: cost tied to usage, not hardware ownership
- Standard building blocks: VMs/containers, object storage, managed services
- Operational shift: provider runs the platform; you run what you deploy on it

Why Start with History?

- “New” cloud ideas often have older ancestors (shared access, pooling, metering)
- Helps separate: what’s genuinely new vs. what’s repackaged with better tooling
- Explains design choices (multi-tenancy, automation, resiliency patterns)
- Creates a timeline to understand service/deployment model tradeoffs

Recurring Themes Across the Timeline

- **Sharing**: multiple users/applications share expensive resources efficiently
- **Abstraction**: hide hardware complexity behind simpler interfaces
- **Automation**: replace manual operations with repeatable processes/APIs
- **Standardization**: common hardware/software patterns enable scale
- **Economics**: cost, utilization, and speed push each major transition

Mainframes and Centralized Computing

- Computers were rare, expensive, and centrally managed
- Many users shared one machine; access via terminals
- IT operated as a controlled utility (queues, quotas, scheduling)
- Strong governance: performance, security, and change tightly managed
- Compute treated as scarce capacity that must be allocated carefully

Mainframe Era: What It Got Right (and What It Limited)

- **Strengths**: high utilization, centralized control, consistent performance
- **Operational model**: specialized staff, formal processes, slow change
- **Constraint**: limited flexibility for experimentation and rapid deployment
- **User experience**: “request resources” rather than “self-serve resources”
- Connection to cloud: central pooling and managed access show up again later

Time-Sharing as a Precursor to the Cloud

- Multi-user systems let many people interact with one computer “at once”
- Introduced fairness concepts: slicing CPU time, prioritizing workloads
- Encouraged interactive computing (not just batch processing)
- Created early expectations of “computing as a shared service”

Time-Sharing: Proto-Cloud Concepts

- **Multi-tenancy**: multiple users share the same underlying system
- **Metering/quotas**: track usage, allocate capacity, prevent abuse
- **Isolation**: keep users from interfering with each other's work
- **Service mindset**: reliability and availability become operational goals
- Discussion prompt: Which of these show up directly in modern public cloud?

The Personal Computer Revolution

- Computing moved from centralized machines to individual desktops
- Lower cost hardware empowered departments and individuals
- Innovation accelerated, but governance/control fragmented
- New norm: local compute + local software installation

Client–Server: The Next Step After PCs

- PCs became clients; shared services moved to servers (files, databases, apps)
- Benefits: shared data, centralized services, more powerful backends
- Costs: network dependence, server management, more moving parts
- Scaling challenge: as users grew, server fleets and ops complexity grew too
- Typical evolution: single server → multiple tiers → load balancing → clusters

The Cost and Complexity of On-Premises IT

- You own everything: hardware, networking, power, cooling, physical space
- You build processes: procurement, patching, monitoring, backups, security
- You handle reliability: redundancy, disaster recovery, capacity planning
- Slow feedback loops: buying hardware can take weeks/months

On-Prem Pain Points (Why Teams Looked for Alternatives)

- **CapEx vs OpEx**: large upfront purchases vs ongoing usage-based costs
- **Overprovisioning**: buy for peak demand → idle capacity most of the time
- **Underprovisioning**: buy too little → outages/performance issues
- **Environment drift**: dev/test/prod differences create deployment risk
- **Ops bottlenecks**: tickets and handoffs slow delivery

The Evolution of Cloud Computing

Part 2

Virtualization Changes the Economics

- Runs multiple virtual machines on one physical server
- Improves utilization: consolidate workloads onto fewer machines
- Decouples software from hardware: easier to move/replicate environments
- Makes “resource pooling” practical in modern data centers

Virtualization: Why It's a Cloud Enabler

- **Isolation**: workloads separated even when sharing hardware
- **Encapsulation**: VM images capture a system configuration
- **Flexibility**: faster provisioning than racking physical servers
- **Mobility**: migrate workloads between hosts for maintenance/failover
- Bridge to cloud: standard units of compute that can be created on demand

Commodity Hardware at Massive Scale

- Instead of specialized machines: lots of relatively inexpensive servers
- Scale-out approach: add more nodes rather than “buy a bigger box”
- Standardization simplifies procurement, replacement, and automation
- Enables massive data centers built from repeatable building blocks

Hyperscale Operations: Designing for Failure

- At large scale, failures are normal (disks, servers, networks)
- Reliability shifts from “perfect hardware” to “resilient software”
- Use redundancy, replication, and automation to recover quickly
- Operational focus: monitoring, incident response, and continuous improvement
- Key idea: resilience becomes an architectural requirement, not an afterthought

2006: Infrastructure Becomes a Service (IaaS)

- Compute and storage offered as on-demand building blocks
- Self-service replaces procurement cycles for many workloads
- Resources become programmable (APIs), not just managed by tickets
- Start of public cloud as a mainstream model for infrastructure consumption

What IaaS Actually Provides (and What It Doesn't)

- Provides: VMs/compute, networking primitives, storage options
- You still design: availability, security configuration, scaling strategy
- Tradeoff: faster provisioning and flexibility, but new operational skills needed
- Introduce "shared responsibility" idea (preview): provider vs customer duties
- Example prompt: "What stays the same from on-prem? What fundamentally changes?"

API-Driven and Automated Infrastructure

- Infrastructure becomes software-controlled: create/modify via APIs
- Repeatability: the same action can be executed consistently every time
- Enables self-service portals and automation pipelines
- Reduces manual configuration (and the errors that come with it)

Infrastructure as Code (IaC) and the Ops Workflow Shift

- Treat infrastructure definitions like source code (versioned, reviewed)
- CI/CD concepts extend beyond apps to environments
- Benefits: reproducibility, auditability, faster recovery and scaling
- Cultural change: Dev + Ops collaboration and shared ownership
- Common outcome: faster delivery with guardrails (policy, templates, standards)

Containers: Lighter-Weight Abstraction

- Package an application with its dependencies into a consistent unit
- Shares the host OS kernel → often faster startup and higher density than VMs
- Supports immutable deployment patterns (replace instead of patch-in-place)
- Helps standardize “it runs the same everywhere” across environments

Containers vs. VMs (How to Explain the Difference)

- VM: virtualizes hardware; each VM has its own guest OS
- Container: virtualizes at the OS level; isolates processes and dependencies
- Practical impact: speed, density, and portability tradeoffs
- Typical pattern: VMs as the substrate + containers on top (especially in public cloud)

Orchestration and Cloud-Native Scale

- As container counts grow, you need a “control plane” for operations
- Orchestration handles scheduling, restarts, scaling, and placement
- Moves deployments from “pets” to “cattle” (replaceable units)
- Enables higher-level application patterns (services, rolling updates)

What Orchestrators Actually Do for You

- **Scheduling**: decide where workloads run based on resource needs
- **Desired state**: keep the system matching declared configuration
- **Service discovery & networking**: connect components reliably
- **Autoscaling**: adjust replicas/resources based on demand signals
- **Observability hooks**: health checks, logs, metrics, rollout status

Economic Drivers of Cloud Adoption

- Elasticity: match capacity to demand (avoid over/underprovisioning)
- Efficiency: higher utilization via pooling and standardization
- Speed: faster experimentation and deployment → faster time-to-value
- Risk tradeoff: shift some operational burden to providers, but increase dependency on good architecture/governance

Key Takeaways (and a Quick Check)

- Cloud is the convergence of sharing + abstraction + automation + scale economics
- Mainframes/time-sharing established core ideas (pooling, metering, managed access)
- PCs/client-server increased flexibility but made operations and scaling harder
- Virtualization + commodity hardware + APIs enabled IaaS and cloud-native platforms
- Quick check questions:
  - What's one "old" idea that cloud makes practical at global scale?
  - What's one operational responsibility that doesn't disappear in the cloud?