Cloud Service Models

Part 1

Cloud Service Models (Why They Matter)

- Cloud services differ by level of abstraction
- Each model shifts responsibility between provider and customer
- The trade-off space: control vs flexibility vs operational effort
- Model choice impacts architecture, cost, and security decisions

Roadmap and Learning Objectives

- Define “service model” and interpret the stack layers
- Understand IaaS responsibilities (what you run vs what the provider runs)
- Build a decision lens for choosing between IaaS/PaaS/SaaS/FaaS
- Preview: shared responsibility and security implications

What Is a Cloud Service Model?

- A service model defines what the provider manages vs what the user manages
- It organizes cloud offerings into layers
- Higher layers = less operational responsibility
- Lower layers = more control and customization

The Stack: From Hardware to Application

- Hardware + networking at the base
- OS, runtime, middleware in the middle
- Applications + data at the top
- Service models differ in how much of this stack you manage

A Practical Way to Think About Models

- Ask: “Which layer(s) am I responsible for operating day-to-day?”
- Separate *building* vs *running*: dev effort vs ops effort
- Identify the “undifferentiated heavy lifting” you can offload
- Decide where standardization is acceptable vs where customization is required

The Four Common Service Models (At a Glance)

- IaaS: provider gives compute/storage/network; you manage OS→apps
- PaaS: provider manages OS/runtime/middleware; you focus on code
- SaaS: provider delivers the whole application
- FaaS: event-driven functions; “no visible server management”

Infrastructure as a Service (IaaS): Definition

- Provider supplies compute, storage, and networking
- Customer manages OS, runtime, and applications
- Closest to traditional data centers
- High control, higher operational burden

IaaS: What You Get (Common Building Blocks)

- Virtual machines / instance types (CPU, memory, GPU)
- Virtual networking (VPC/VNet, subnets, routing, firewalls)
- Storage options (block, object, file) + snapshots/backup primitives
- Load balancing and basic autoscaling capabilities (provider tools; you configure)

What You Manage in IaaS (Expanded)

- OS installation and patching
- Security configuration and updates
- Scaling and availability planning
- Full responsibility for applications and data
- Operational necessities: monitoring, logging, incident response, runbooks

IaaS Availability: What “You Still Design”

- Redundancy across zones/regions (where required)
- Health checks + load balancing behavior (routing, failover)
- Backup/restore strategy and recovery objectives (RTO/RPO)
- Capacity planning vs autoscaling (and when each is appropriate)

IaaS Security: What Changes vs On-Prem

- You still own configuration risk (misconfigurations are a top failure mode)
- Identity and access design becomes foundational (least privilege, roles)
- Network segmentation is software-defined (security groups/NACLs)
- Patch cadence and vulnerability management remain your responsibility

IaaS Cost Drivers (Typical)

- “Always-on” compute (baseline instances) vs elastic scaling
- Storage growth + snapshots/backups + cross-region replication
- Data transfer and egress (especially multi-region and internet delivery)
- Operational overhead: time spent on OS/security/availability work

IaaS Fit: When It's Usually the Right Choice

- You need OS-level control, custom networking, or custom middleware
- Legacy workloads that aren't ready for platform constraints
- Specialized requirements (appliances, niche runtimes, bespoke controls)
- You have (or can build) the operational capability to run it reliably

Cloud Service Models

Part 2

Platform as a Service (PaaS): Definition

- Provider manages OS, runtime, and middleware
- Developers focus on application code
- Built-in scaling and availability (within platform options)
- Reduced operational complexity

What PaaS Typically Includes (So You Don't Have To)

- Managed runtimes (language/framework runtime versions)
- Managed middleware (web servers, app hosting, job schedulers)
- Managed data services (often as companions: SQL/NoSQL/queues/caches)
- Platform features: deployments, rollbacks, health checks, autoscaling hooks

PaaS Benefits: Why Teams Choose It

- Faster development and deployment
- Standardized environments reduce “works on my machine”
- Less time spent on patching and base OS hardening
- Easier scaling for typical web/API workloads (with platform constraints)

Trade-offs of PaaS (And How to Manage Them)

- Less control over environment details
- Opinionated platforms may limit customization
- Potential vendor lock-in
- Mitigations: portable runtimes, clear interfaces, data portability plans, exit strategy

SaaS: Definition

- Complete applications delivered over the internet
- Provider manages everything except user data and access
- No infrastructure or platform management for the customer
- Most familiar cloud model to end users

SaaS: What You Still Have to Operate

- Identity, access, and governance (roles, MFA/SSO, lifecycle)
- Data classification, retention, and eDiscovery requirements
- Integrations (APIs, webhooks, ETL), and their reliability
- Vendor management: SLAs, compliance evidence, incident communications

When SaaS Makes Sense (Use Cases + Signals)

- Standardized business functions (email, CRM, collaboration)
- Minimal IT staff required
- Rapid onboarding and updates
- Least flexibility, lowest operational burden

SaaS Selection Checklist (Architecture View)

- Data residency and compliance fit (where data lives, audit controls)
- Identity integration (SSO, SCIM, role models)
- Export/backup options and offboarding path (avoid “data hostage” risk)
- Extensibility: APIs, workflow automation, eventing
- Availability and support model aligned to business criticality

Function as a Service (FaaS): Definition

- Event-driven execution of small functions
- No server management visible to the user
- Automatic scaling to zero or high demand
- Example: AWS Lambda

How Serverless Works (Conceptually)

- Triggers: events (HTTP requests, queue messages, file uploads, schedules)
- Short-lived compute units: stateless by default; state externalized
- Concurrency scaling: many instances in parallel when demand spikes
- Pricing aligned to execution time/requests (vs always-on servers)

Serverless Trade-offs (And Typical Mitigations)

- Great for bursty or unpredictable workloads
- Pay only for execution time
- Limited execution time and environment control
- Debugging and observability can be harder
- Mitigations: structured logging, tracing, local emulation, clear event contracts

Comparing Control vs Responsibility (Summary Slide)

- IaaS: maximum control, maximum responsibility
- PaaS: balanced control and convenience
- SaaS: minimal control, minimal effort
- FaaS: extreme abstraction, event-focused logic
- Practical lens: “What do you want to customize?” vs “What do you want to avoid operating?”

Shared Responsibility and Security

- Responsibility shifts with the service model
- Provider always secures the underlying infrastructure
- Customers remain responsible for data and access
- Model choice affects compliance, auditing, and risk
- “Security work” moves upward: fewer servers to patch, more identity/data governance to get right

Wrap-Up: A Simple Decision Framework

- Step 1: Identify what must be customized (OS? runtime? app features?)
- Step 2: Identify what must be controlled (latency, residency, compliance, integration)
- Step 3: Quantify ops capacity (who patches/monitors/responds?)
- Step 4: Choose the highest abstraction that still meets constraints
- Step 5: Document shared responsibility boundaries (data, access, config, incident response)