**1️⃣ Monolith vs Microservices**

**🔹 What is a Monolith?**

A **monolith** is like a *single, tightly packed application*. All parts of the application — UI, business logic, database access — are built and deployed together as **one unit**.

**🔹 What is a Microservices Architecture?**

A **microservices** approach breaks the system into **independent components (services)**, each handling a specific business capability — e.g., login, payments, catalog, notifications.

Each service:

* Runs independently
* Communicates over APIs
* Can be deployed/scaled separately

**✅ TPM Relevance**

| **Aspect** | **Monolith** | **Microservices** | **TPM's Role** |
| --- | --- | --- | --- |
| **Project Complexity** | Low | High | Ensure clear documentation, service boundaries, and ownership |
| **Deployment** | One unit | Multiple units | Coordinate CI/CD pipelines, service-level releases |
| **Team Structure** | Centralized | Decentralized (Service Owners) | Align teams to services, manage dependencies |
| **Scalability** | Whole system scales | Scale individual services | Prioritize scaling efforts based on bottlenecks |

👉 **TPM Tip:** Microservices = More flexibility + more moving parts. You’ll need **strong program coordination**, service ownership mapping, and tracking of **inter-service SLAs**.

**2️⃣ Scalability, Reliability, Availability**

Let’s break these down like pillars of system health — **every TPM must know how to talk about and prioritize these** with engineers, SREs, and product teams.

**🟦 Scalability**

**Definition:** System’s ability to handle **increasing load** without performance loss.

* **Vertical scaling:** Add more power to a single machine (CPU/RAM).
* **Horizontal scaling:** Add more machines/services.

**🎯 TPM Focus:**

* **Forecast traffic/load with product and business teams**
* **Identify scaling thresholds** (when to scale, what to scale)
* **Ensure budget alignment** for infra costs

**🟩 Reliability**

**Definition:** System’s ability to **work consistently and correctly** over time.

* Is the system returning correct responses?
* Does it degrade gracefully during failures?

**🎯 TPM Focus:**

* Track **error rates, failed requests, retries**
* Prioritize **retries, fallback mechanisms**, and **incident response workflows**
* **Define and monitor SLIs** (e.g., success rate > 99.9%)

**🟨 Availability**

**Definition:** Percentage of time the system is **up and usable**.

* Usually measured in **nines** (e.g., 99.9% uptime = ~43 mins of downtime/month)

**🎯 TPM Focus:**

* Collaborate on **redundancy, failover** plans (multi-region, active-active setups)
* Ensure **SLA alignment** with engineering and business
* Drive **incident reviews** to improve uptime

**3️⃣ CAP Theorem and Its Impact**

**🔸 What is CAP Theorem?**

The **CAP Theorem** says a distributed system can only guarantee **2 out of 3** things at the same time:

| **Letter** | **Stands For** | **Meaning** |
| --- | --- | --- |
| **C** | **Consistency** | Every read returns the latest write |
| **A** | **Availability** | Every request gets a response |
| **P** | **Partition Tolerance** | System continues to work even if parts of it are disconnected |

You must choose 2 — **you can’t have all 3** in a distributed system.

**🧠 Real-World Examples (TPM-Focused)**

| **System Type** | **Chooses** | **Example** | **TPM Concerns** |
| --- | --- | --- | --- |
| Banking App | **CP** | Prioritize correctness over speed | Coordinate with SREs to reduce unavailability during partition |
| Social Media Feed | **AP** | Show older posts if database is slow | Accept *eventual consistency*, ensure smooth UX fallback |
| e-Commerce Cart | **CA (not always possible)** | Consistent and available until network partition happens | Handle retries, use local caching during partitions |

**🧭 Final Notes for TPMs:**

| **Principle** | **What to Remember as TPM** |
| --- | --- |
| **Monolith vs Microservices** | Focus on service boundaries, ownership clarity, and deployment coordination |
| **Scalability** | Work with engineering to plan for growth and bottlenecks |
| **Reliability** | Drive initiatives around monitoring, alerting, and graceful degradation |
| **Availability** | Prioritize high availability SLAs, ensure proper incident handling |
| **CAP Tradeoffs** | Help stakeholders understand technical trade-offs in terms of user/business impact |

**1️⃣ Load Balancers**

**🔹 What is a Load Balancer?**

Think of a **load balancer** like a **traffic police officer** — it sits in front of your backend services and **distributes incoming requests** evenly so that **no server gets overwhelmed**.

**🔹 Types:**

* **Layer 4 (Transport-level):** Routes based on IP, TCP/UDP.
* **Layer 7 (Application-level):** Routes based on content (URL path, cookies, headers).

**✅ TPM Relevance:**

| **TPM Concern** | **What to Do** |
| --- | --- |
| **Scalability** | Load balancers help scale horizontally (more backend servers behind one endpoint). |
| **Fault Tolerance** | Can reroute traffic if a server fails. TPMs must ensure health check policies are defined. |
| **Release Coordination** | Use LB for **blue-green** or **canary deployments**. TPM helps coordinate phased rollouts. |

**2️⃣ Databases**

**🔹 What is a Database?**

A **database** stores your application’s data — customer info, orders, files, logs, etc.

**🔹 Common Types:**

| **Type** | **Purpose** | **Example** |
| --- | --- | --- |
| **Relational (SQL)** | Structured data, transactions | PostgreSQL, MySQL |
| **NoSQL (Document/Key-Value)** | High-speed, flexible schema | MongoDB, DynamoDB |
| **Time-Series, Graph, Columnar** | Specialized cases | InfluxDB, Neo4j, Redshift |

**✅ TPM Relevance:**

| **TPM Concern** | **What to Do** |
| --- | --- |
| **Data Modeling Milestones** | Coordinate schema design with DBAs and backend engineers |
| **Backup & Restore Plans** | Ensure disaster recovery plans are in place |
| **Latency SLAs** | Support monitoring database query latency and throughput |

**3️⃣ Queues (Message Brokers)**

**🔹 What is a Queue?**

A **queue** decouples systems. When one service sends a message but the other is not ready, the **message waits in the queue**.

Like a restaurant order queue: kitchen handles one order at a time, but orders can keep coming.

**🔹 Examples:**

Amazon SQS, RabbitMQ, Kafka

**✅ TPM Relevance:**

| **TPM Concern** | **What to Do** |
| --- | --- |
| **Asynchronous Workflows** | Use queues for background tasks (e.g., processing images, sending emails) |
| **Retry Logic** | Ensure failed messages are retried or dead-lettered |
| **Throughput Bottlenecks** | Monitor queue length and consumer lag to avoid backlogs |

**4️⃣ Caches**

**🔹 What is a Cache?**

A **cache** is a temporary storage for **frequently used data** — to avoid hitting the main database again and again.

Like keeping your favorite snacks on the desk — no need to go to the kitchen every time.

**🔹 Types:**

* **In-Memory Cache:** Redis, Memcached
* **Browser Cache / CDN:** Static content caching (JS, images)

**✅ TPM Relevance:**

| **TPM Concern** | **What to Do** |
| --- | --- |
| **Performance** | Help identify caching opportunities to reduce latency |
| **Cost Management** | Reduce database and compute costs via cache hits |
| **Stale Data Risks** | Coordinate cache invalidation strategies with developers |

**5️⃣ Stateless vs Stateful Services**

**🔹 Stateless Services**

A **stateless service** does not remember anything about previous requests.

E.g., Every time you ask for the weather, it processes it the same way — it doesn’t remember your last request.

**🔹 Stateful Services**

A **stateful service** keeps track of past interactions.

E.g., Video streaming keeps track of where you paused.

**✅ TPM Relevance:**

| **Aspect** | **Stateless** | **Stateful** | **TPM's Role** |
| --- | --- | --- | --- |
| **Scaling** | Easy to scale | Harder (requires session handling) | Encourage stateless design for better scaling |
| **Resilience** | Restartable easily | Risk of session loss | Track failover requirements and state persistence |
| **Cost** | Often cheaper | More infra + complexity | Evaluate trade-offs for project goals |

**6️⃣ API Gateways and Service Mesh**

**🔹 What is an API Gateway?**

A **central entry point** for all external API requests to your microservices.

Think of it as a **reception desk**: it receives requests, verifies them, and routes them to the right service.

**Features:**

* Rate limiting
* Authentication/Authorization
* Logging and monitoring
* Request transformation

**🔹 What is a Service Mesh?**

A **dedicated infrastructure layer** that handles **internal service-to-service communication** in microservice systems.

Like an **internal postal system** between microservices with tracking, security, and retry logic.

Examples: Istio, Linkerd, AWS App Mesh

**✅ TPM Relevance:**

| **Area** | **API Gateway** | **Service Mesh** |
| --- | --- | --- |
| **Purpose** | External traffic control | Internal microservices communication |
| **Security** | Handle auth/authz, rate limits | Enforce mutual TLS |
| **Monitoring** | Logging, API analytics | Tracing, retries, circuit breakers |
| **TPM Actions** | Define usage quotas with business | Ensure observability and standardization |

**🧭 Final TPM Summary Table**

| **Component** | **TPM Should...** |
| --- | --- |
| **Load Balancer** | Coordinate health checks, phased rollouts, and high availability |
| **Database** | Ensure data modeling reviews, backups, and performance monitoring |
| **Queues** | Drive async flow design, backlog monitoring, and failure handling |
| **Cache** | Promote latency reduction and cost optimization via smart caching |
| **Stateless/Stateful** | Push for stateless services for better resilience and scaling |
| **API Gateway** | Secure and control external access to services |
| **Service Mesh** | Ensure service observability, retries, and standardization at scale |

**1️⃣ Latency, Throughput, and Uptime**

These three metrics define **how well** your system performs.

**🔹 Latency – *“How fast is the response?”***

**Definition:** Time taken to respond to a request (usually in milliseconds).

Example:  
If a user clicks **“Download File”**, and the file takes 300ms to start downloading, that’s your **latency**.

**🎯 TPM Relevance:**

| **Scenario** | **TPM Actions** |
| --- | --- |
| Slow page load or API response | Raise latency alerts, prioritize root cause analysis |
| Business asks for better UX | Drive caching, edge delivery, or API optimization programs |
| Target: < 100ms latency for key APIs | Track via dashboards and ensure engineers are optimizing code paths |

**🔹 Throughput – *“How much can we handle?”***

**Definition:** Number of requests the system can handle in a given time (e.g., requests/second).

Example:  
If 10,000 users are uploading files at the same time, **can your backend process them fast enough?**

**🎯 TPM Relevance:**

| **Scenario** | **TPM Actions** |
| --- | --- |
| Peak traffic (campaigns, product launches) | Coordinate load testing, infra scaling, and throttling policies |
| File uploads slow during peak | Identify throughput bottlenecks (disk I/O, DB writes, queues) |
| Partner apps pushing too many API calls | Enforce API rate limits, advocate for quotas via the API gateway |

**🔹 Uptime – *“How often is the system available?”***

**Definition:** The percentage of time the system is running and accessible (e.g., 99.9% uptime).

| **Uptime** | **Downtime/month** |
| --- | --- |
| 99.9% | ~43 minutes |
| 99.99% | ~4.3 minutes |
| 99.999% | ~26 seconds |

**🎯 TPM Relevance:**

| **Scenario** | **TPM Actions** |
| --- | --- |
| SLA breach risk | Ensure active-active failover, redundancy, monitoring in place |
| Incident review shows repeated outages | Push for root cause analysis (RCA) and postmortem follow-up |
| Enterprise client demands 99.99% uptime | Collaborate with SRE/DevOps to design for availability zones/multi-region |

**2️⃣ SLIs, SLOs, and SLAs – What TPMs Should Monitor**

**🔹 SLI (Service Level Indicator)**

A **measurable metric** — e.g., “99.95% of login requests responded under 300ms.”

**Examples:**

* Request success rate
* Availability (% of successful requests)
* Error rate

🎯 TPM Use:

* **Track real-time service health**
* **Baseline performance before setting SLOs**

**🔹 SLO (Service Level Objective)**

**Internal performance goals** the team agrees on — targets tied to SLIs.

**Examples:**

* “API success rate should be ≥ 99.9%”
* “Median latency < 250ms”

🎯 TPM Use:

* Help teams define **realistic targets**
* **Negotiate trade-offs** (e.g., latency vs feature speed)
* Track **error budget usage** (how much failure is acceptable before investigation)

**🔹 SLA (Service Level Agreement)**

**Formal contract** with customers — includes SLOs but adds penalties if breached.

**Examples:**

* “We guarantee 99.9% uptime or refund 10% of monthly bill”
* “Support ticket response within 2 hours”

🎯 TPM Use:

* Understand the **business implications** of SLA breaches
* Escalate when teams are nearing SLA thresholds
* Ensure **monitoring and alerting** are aligned to SLA definitions

**✅ TPM Comparison Table**

| **Term** | **Who It’s For** | **Enforceable?** | **TPM Focus** |
| --- | --- | --- | --- |
| **SLI** | Engineers/TPMs | No | Use for measurement and trend analysis |
| **SLO** | Internal teams | No | Help define and track team goals |
| **SLA** | Customers | Yes, legal binding | Drive compliance, risk mitigation, customer reporting |

**3️⃣ Workshop: Whiteboard a Basic System – Online File Storage/Sharing Platform**

**🧩 Goal:**

Design a **basic version of Dropbox/Google Drive**, focusing on:

* File upload/download
* User authentication
* Sharing with others

**✅ TPM-Focused System Design Breakdown**

| **Component** | **Description** | **TPM Guidance** |
| --- | --- | --- |
| **Frontend (Web + Mobile UI)** | User interface for file actions | Ensure UI team milestones align with API backend |
| **API Gateway** | Routes requests to correct microservices | Define service boundaries and routing rules |
| **Authentication Service** | Login, tokens, sessions | Verify rate limits, session expiry, and security |
| **File Upload Service** | Handles user uploads | Manage file size limits, chunking strategy, retries |
| **Storage Service (e.g., S3)** | Stores files securely and durably | Decide storage class (standard, infrequent), versioning policy |
| **Sharing Service** | Link generation, permissions | Discuss RBAC vs ACL model with engineers |
| **Database** | Metadata (filename, owner, permissions) | Confirm schema design, query latency, indexing needs |
| **CDN / Cache** | Accelerates file download speed | Evaluate performance gains, cost of global edge delivery |
| **Queue (Async Ops)** | Thumbnail generation, virus scan | Align async SLAs (e.g., thumbnail within 3s), track queue health |
| **Monitoring & Logs** | Observability for failures, SLIs | Work with SRE team to define key metrics and dashboards |

**📝 Whiteboarding Prompts (Ask Participants):**

1. **What happens when a user uploads a large file?**
   * (Chunking? Retry? Queueing?)
2. **How do we share a file securely with external users?**
   * (Token-based link? Expiry? Access revocation?)
3. **What if the file service goes down?**
   * (Failover strategy? Retry logic? SLO violation alert?)
4. **Where will we track latency and throughput?**
   * (Which SLIs matter most here? API latency? File availability?)

**🎯 Final Takeaways for TPMs**

| **Metric or Tool** | **TPM’s Core Responsibility** |
| --- | --- |
| **Latency** | Drive performance optimization for user experience |
| **Throughput** | Monitor system load vs capacity, coordinate scale-out planning |
| **Uptime** | Ensure resilience design and post-incident learning |
| **SLI/SLO/SLA** | Speak both **engineering** and **business language**; link metrics to customer outcomes |
| **System Design (Workshop)** | Facilitate design discussions, clarify ownership, track milestone risks |

**✅ 1. System Design Basics**

**Monolith vs Microservices | Scalability | Reliability | Availability | CAP Theorem**

| **AWS** | **Azure** | **GCP** |
| --- | --- | --- |
| AWS Elastic Beanstalk (Monolith) | Azure App Service | App Engine (Standard) |
| AWS ECS / EKS / Lambda (Microservices) | Azure Kubernetes Service (AKS) / Azure Functions | GKE / Cloud Functions |
| Auto Scaling Groups | Azure VM Scale Sets | Managed Instance Groups |
| AWS Global Accelerator / Route 53 | Azure Traffic Manager | Cloud Load Balancing |
| AWS S3 / DynamoDB (High Availability + Partition Tolerance) | Azure Cosmos DB | Cloud Spanner / Bigtable |

**✅ 2. Key Components of System Design**

**Load Balancers | Databases | Queues | Caches | Stateless vs Stateful | API Gateway | Service Mesh**

| **Component** | **AWS** | **Azure** | **GCP** |
| --- | --- | --- | --- |
| **Load Balancer** | Elastic Load Balancer (ALB/NLB) | Azure Load Balancer / Application Gateway | Cloud Load Balancing |
| **Database (Relational)** | RDS (MySQL, PostgreSQL) | Azure SQL / PostgreSQL | Cloud SQL |
| **Database (NoSQL)** | DynamoDB | Azure Cosmos DB | Firestore / Bigtable |
| **Queues** | SQS / SNS | Azure Service Bus / Queue Storage | Pub/Sub |
| **Cache** | ElastiCache (Redis, Memcached) | Azure Cache for Redis | Memorystore (Redis) |
| **Stateless/Stateful** | Lambda (Stateless), EBS-backed EC2 (Stateful) | App Service (Stateless), Stateful VMs | Cloud Run (Stateless), Persistent Disks |
| **API Gateway** | Amazon API Gateway | Azure API Management | Apigee / API Gateway |
| **Service Mesh** | AWS App Mesh | Azure Service Fabric Mesh / Open Service Mesh | Anthos Service Mesh / Istio on GKE |

**✅ 3. Metrics and SLAs for TPMs**

**Latency | Throughput | Uptime | SLIs, SLOs, SLAs**

| **AWS** | **Azure** | **GCP** |
| --- | --- | --- |
| Amazon CloudWatch | Azure Monitor | Cloud Monitoring (formerly Stackdriver) |
| AWS X-Ray (Tracing) | Azure Application Insights | Cloud Trace |
| AWS CloudTrail (Audit) | Azure Log Analytics | Cloud Logging |
| AWS Trusted Advisor / Health Dashboard | Azure Service Health | GCP Operations Suite (Uptime Checks) |
| AWS Service Catalog (for SLA definitions) | Azure Service Level Objectives | Cloud Service Level Objectives (via Anthos/SLM tooling) |

**✅ 4. Workshop: Online File Storage/Sharing Platform**

| **Component** | **AWS** | **Azure** | **GCP** |
| --- | --- | --- | --- |
| **File Storage** | S3 | Azure Blob Storage | Cloud Storage |
| **User Auth** | Cognito | Azure AD B2C | Firebase Auth / IAM |
| **Metadata DB** | DynamoDB / RDS | Cosmos DB / SQL | Firestore / Cloud SQL |
| **File Upload API** | API Gateway + Lambda / ECS | API Management + Functions / AKS | API Gateway + Cloud Run / Functions |
| **CDN** | CloudFront | Azure Front Door / CDN | Cloud CDN |
| **Virus Scan / Async Processing** | Lambda + S3 Event + SQS | Functions + Event Grid + Service Bus | Cloud Functions + Pub/Sub |
| **File Sharing Logic** | Signed URLs (S3) | Shared Access Signatures (SAS) | Signed URLs (GCS) |
| **Monitoring** | CloudWatch + X-Ray | Monitor + App Insights | Cloud Monitoring + Trace |