**1. Core Architectural Requirements**

✅ **Scalability and Resource utilization**

* The platform must support **dynamic scaling** to handle peak loads during high-demand events (e.g., movie premieres, concerts).
* The architecture should support **horizontal scaling** (adding more instances) and/or **vertical scaling** (upgrading resources).
* Load should be **evenly distributed** across services to prevent bottlenecks and make sure the resources are utilized evenly.

✅ **Performance & Low Latency**

* Response times must be optimized for a **snappy user experience**, particularly for seat selection and booking transactions.
* **Asynchronous processing** should be considered for non-blocking operations (e.g., notifications, analytics).
* Caching strategies should be considered across different places to reduce redundant processing and improve response times.

✅ **Multi-Tenancy**

* The platform should support **multiple event organizers** (theaters, concert halls, arenas) in a shared environment while ensuring **data isolation**.
* Support for configurable Business logic, pricing, and security policies **per tenant or region**.

✅ **High Availability & Fault Tolerance**

* The system should be **resilient to failures**, ensuring that service outages in one region do not impact others.
* **Automated failover mechanisms** must be in place to ensure continuity during infrastructure failures.
* **Redundant data storage** and **disaster recovery mechanisms** should be in place.

✅ **Security & Compliance**

* Authentication and authorization must follow **industry standards** (e.g., OAuth 2.0, SSO, Role-Based Access Control).
* Data must be **encrypted at rest and in transit** to protect sensitive information.
* The platform must support **regional compliance regulations** (GDPR, CCPA, PCI DSS).
* **Access logs and audit trails** should be maintained for security analysis and compliance reporting.

✅ **Interoperability & Cloud Portability**

* The platform should be **agnostic to cloud providers**, allowing for migration to different cloud environments if necessary.
* **Standardized interfaces** (e.g., REST APIs, GraphQL, gRPC) should be used for communication.
* Infrastructure should support **hybrid cloud** and **on-premise workloads** for compliance-sensitive operations.

✅ **Observability & Monitoring**

* The system should provide **centralized logging, monitoring, and alerting** to ensure proactive issue detection.
* Key metrics should be collected for **performance optimization and anomaly detection**.

**2. Architectural Characteristics**

| **Characteristic** | **Description** |
| --- | --- |
| **Modular** | Services should be loosely coupled and independently deployable. |
| **Event-Driven** | The system should use an event-driven architecture for async processing. |
| **Resilient** | The system should gracefully handle failures with retry mechanisms and circuit breakers. |
| **Stateless & Stateful Services** | Stateless for API calls, stateful where necessary (e.g., seat reservations). |
| **Tenant & Data Isolation** | Must support logical or physical separation of tenant data. |
| **Multi-Region Support** | Ability to deploy services in multiple regions for performance and compliance. |

**3. Architectural Constraints**

✅ **Data Sovereignty & Compliance Constraints**

* The system must **store and process data within regional boundaries** as required by regulations (e.g., EU GDPR, US CCPA).
* **Different data retention policies** should be configurable per region.

✅ **Latency & Network Constraints**

* The system must ensure that users in different regions experience **low-latency responses**.
* **Edge caching/CDN** may be required to minimize latency.

✅ **Hybrid Deployment Constraints**

* Some workloads (e.g., sensitive financial transactions) must run **on-premise** or in a **private cloud**.
* The architecture must support **secure communication** between on-premise and cloud workloads.

✅ **Cost & Resource Optimization**

* The system should be designed to **optimize cloud resource usage** to minimize costs.
* **Auto-scaling policies** should balance performance and cost efficiency.