Platform provisioning, sizing & Release requirements

Platform provisioning, sizing & Release requirements

1. Tech Stack

Programming Language

- Java (Spring Boot) is often considered a top choice for implementing microservices architectures due to several compelling reasons compared to other tech stacks.
- Robust Framework with Built-In Features like Embedded Servers,Dependency Injection,Starter Packs,Configuration Management.
- Spring Cloud extends Spring Boot with tools tailored for microservices like Service Discovery, API Gateway, Load Balancing, Distributed Tracing, Circuit Breakers.
- Language Strengths like Mature Ecosystem, Platform Independence, Concurrency Support.
- Scalability and Performance.

Comparison with Other Tech Stacks

Feature	Java (Spring Boot)	Node.js	Python (Flask/Django)
Performance	High	High for I/O; limited for CPU	Moderate
Concurrency	Excellent with Threads	Event-driven (single-threaded)	Limited
Ecosystem	Mature and extensive	Large, especially for I/O	Growing
Community Support	Excellent	Excellent	Good
Distributed Systems	Comprehensive (Spring Cloud)	Moderate (e.g., Express + tools)	Moderate (manual setup)

Message Queue

- Kafka for event-driven updates

Platform provisioning, sizing & Release requirements

2. Database

- MYSQL relational database for Movie,Hall,Theatre management.
- PostgreSQL relational database for Booking seats which handles concurrency better.
- Redis for Global caching.
- Elastic search for Movie search.

3. Data Modeling

Spring Data JPA

- Spring Data JPA, built on Hibernate, is a popular choice for data modeling in Spring Boot because it offers several advantages compared to other options.
- Seamless Integration with Spring Ecosystem
    - Spring Data JPA is part of the Spring ecosystem, meaning it integrates tightly with other Spring Boot components like Spring MVC, Spring Security, and Spring Data. This makes it easy to configure and use in a Spring Boot project with minimal setup.
- Rich and Declarative API like Repository Abstraction, Derived Query Methods, Custom Queries.
- Spring Data JPA uses Hibernate under the hood by default, which
    - Is a powerful and mature ORM framework.
    - Provides rich features like lazy/eager fetching, entity relationships, and dirty-checking.
- Reduced Boilerplate Code.
- Flexibility with Other Tools
    - It supports multiple databases and dynamic switching between them with minimal configuration.

Why Not Other Options in Spring Boot?

- JDBC Template: While lightweight and powerful, it requires manual query writing and mapping, leading to more boilerplate code.
- Spring Data JDBC: A simpler alternative to Spring Data JPA, but lacks advanced ORM features like entity relationships and lazy loading.
- Hibernate (Standalone): Using Hibernate without Spring Data JPA gives fine-grained control but involves more configuration and lacks the repository abstraction.

Platform provisioning, sizing & Release requirements

4. Hosting Solution

Amazon Web Services

- Use AWS Cloud for Hosting.
- Use AWS ECS Cluster (Fargate - Serverless containers) for containerization of all micro services for Horizontal scaling per region.
- Use AWS RDS for MYSQL and POSTGRESQL which accepts max 64 TB of table sharding.
- Use AWS Aurora scales automatically, up to 128 TB of storage per database instance for shared data more than 64 TB which has high throughput and handles 1000's of transactions per second.

AWS vs GCP

Service Offerings for Microservices

AWS:

- **Compute:** EC2, Elastic Beanstalk, AWS Fargate (serverless containers), Lambda (serverless functions).
- **Container Orchestration:** Amazon ECS, Amazon EKS (managed Kubernetes).
- **Service Mesh:** AWS App Mesh.
- **Integration Services:** API Gateway, Step Functions, EventBridge, SQS (queueing), SNS (messaging).
- **Monitoring:** CloudWatch, X-Ray (distributed tracing).

GCP:

- **Compute:** Compute Engine, App Engine, Cloud Functions (serverless functions).
- **Container Orchestration:** Google Kubernetes Engine (GKE), Cloud Run (serverless containers).
- **Service Mesh:** Anthos Service Mesh (Istio-based).
- **Integration Services:** Cloud Endpoints, Cloud Tasks, Pub/Sub (messaging).
- **Monitoring:** Cloud Monitoring, Cloud Trace.

Comparison:

- **GCP** often excels with Kubernetes due to its native connection to Kubernetes (Google created Kubernetes).

Platform provisioning, sizing & Release requirements

- **AWS** has a broader range of services and is more established in the cloud space, offering more options for hybrid deployments.

Pricing

AWS:

- Offers **pay-as-you-go** pricing but can be complex due to numerous services and pricing tiers.
- Reserved and Spot Instances can reduce costs for EC2, but detailed cost optimization requires monitoring.

GCP:

- Known for its **simpler pricing** and automatic sustained-use discounts.
- Preemptible VMs (similar to AWS Spot Instances) and transparent billing often appeal to startups.

Comparison:

- **GCP** has a slight edge for cost simplicity and transparency.
- **AWS** may offer better flexibility for enterprises that require highly specific configurations.

Performance & Scalability

AWS:

- Global leader in cloud regions, availability zones, and services.
- Offers more infrastructure regions than GCP, which could be critical for globally distributed services.

GCP:

- Also highly scalable, with its edge in analytics and machine learning integration (e.g., BigQuery, TensorFlow).
- GKE is highly performant and widely recognized for Kubernetes workloads.

Comparison:

- For **global reach**, AWS may be better.

- For **Kubernetes-first microservices** or AI/ML integration, GCP excels.

Cost vs. Complexity

AWS:

- Rich ecosystem but can be overwhelming, requiring dedicated expertise.

GCP:

- Easier to get started with but may lack some of the depth AWS offers for enterprise-grade microservices.

5. Release Management

Release Planning

Across Geos

- Create AWS ECS Cluster per region via Jenkins (Automated Script).
- Create Databases per region.

Stakeholder Alignment

- Collaborate with product managers, development teams, QA teams, and business stakeholders to define release goals and timelines.

Development

Version Control

- Use version control systems like Git to manage codebases and feature branches.

Environment Setup

- Development, staging, and production environments are essential.
- Use containerization tools like Docker for consistent deployments.

Platform provisioning, sizing & Release requirements

Continuous Integration (CI)

- Implement CI pipelines to automate code integration and run unit tests.

Deployment

Release Candidate Preparation

- Bundle the approved changes into a release candidate.
- Tag the version in the repository for traceability.

Release Strategy

Rolling Update : Gradually replaces the old tasks with new ones and allows rollback if issues arise.

Blue-Green Deployments : Minimize downtime and allow rollback if issues arise.

Canary Releases : Gradually roll out updates to a subset of users to monitor for issues.

Feature Toggles : Deploy features behind flags for controlled activation.

Post-Release

Monitoring and Feedback

- Actively monitor error rates, crash reports, and user feedback post-deployment.
- Resolve critical issues promptly through hotfixes.

Release Retrospective

- Conduct a review meeting to analyze what went well, what didn't, and areas for improvement.
- Document lessons learned for future releases.

Tools and Technologies

- **Version Control**: Git, GitLab, Bitbucket.
- **CI/CD**: Jenkins, GitHub Actions.
- **Monitoring**: New Relic, Grafana.
- **Testing**: Selenium, JUnit.
- **Project Management**: JIRA, Trello.