**ARULKUMAR.** **A**

**Mobile:** **(91)** **8754300277** **Email**: [**a.arulkumar@gmail.com**](mailto:a.arulkumar@gmail.com)

**Career** **Objective:**

In the quest of senior level assignments over innovative software projects that will reveal my R&D skills, through which I could put forth my valuable **Technical** & **Managerial** contribution to IT industry

**IT** **Experience:** 24+ years into Software Development

10+ years as **Technical Architect** & Being as **Technical Project Manager** & guiding large volume of development team across the globe both on-site & offshore.

**Last Served** **Company:** Tech Mahindra Ltd from 24th Nov 2023 to 13th Jan 2025

**Last Played** **Roles:** Managing the Development Team sized 175 and above

**Technica Stack:**

Hands-on -;

Java.J2EE, Spring Framework, Spring Boot, Hibernate, AWS – SQS, MSK(Kafka), S3, Credential Manager, Lambda, Event Source Mapping, Cloud Watch Log / Alarm, Step Function, State Machines, RDBMA- Oracle – CDC (Change Data Capture – Golden Gateway), PostgreSQL – With Materialistic views for transactional tables.

**Design Patterns**: Singleton, Factory, Abstract Factory, Observer (observable – Spring Actuator atc.,)

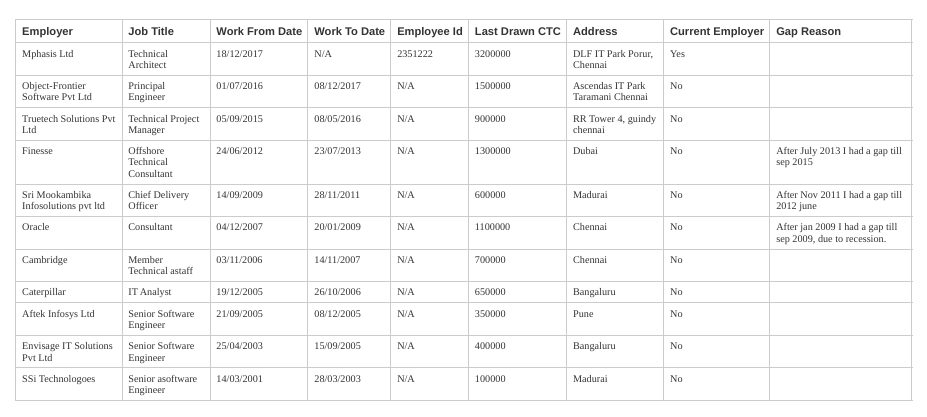
1. **Microservices Patterns: -**
2. **Saaga** –
3. **Choreography** – Using Rest Template – Used to avail asynchronous & reduced http https calls from one microservice to another..
4. **Orchestrator** – Which I have not used & no hands on
5. **Bounded Context Pattern**: Microservice per Schema like Table per microservice
6. **Transactional Outbox Pattern**: I have implemented in places where we were in need to use dual & concurrent writing such as a producer writing in Topic of Partition & Parallelly writing it to DB or Rabbit MQ.
7. **CQRS** – we had the scenario but I did not implement it. I know it’s a clear segregation of read & other write, update operations pattern
8. **AWS – SQS** – It provides built in FIFO mechanism but it hits in perspective of Performance & Cost of implementation, Therefore I have suggested to use **MSK (Managed Services on Kafka)** which has no restriction on messages memory limit. It provides 1 MB default.
9. AWD MSK Producer Configuration – Producer with **Linger.ms as 5 to 10 milli seconds** to avail it ad batch service & increased the default batch size from it’s default 16kb to 12bkb for performance & cost effectiveness.
10. Consumer – Should be Ensured with it confirmation on Acknowledgement & Offset Commit, therefore it will not be redundant reading the same message again. Consumer should be configured with batch consumption by changing it’s default value from 500 to 1000 or 5000 messages at a particular time
11. To handle exception during push & consumption, I have configured three retry mechanisms such as Plain java try catch, Resilience Framework’s callback method & in AWS cloud side with Step Function / State Machines call wait & retry mechanism along with Enabling & disabling Event source mapping configured with Kafka Listener…
12. Idempotency Implemented
13. Lambda – Implemented Nodejs for simple message push & Implemented Java for SOAP to parse complicated SOAP messages into JSON / XML.
14. Event Source Mapping configured across Topic & Lambda and other EC2 hosted microservices
15. Used Cloud watch Log & Alarm Configured with Error Metrics combined with Step Functions and State Machines.
16. **Data**: Based on CAP Theorem for Consistency, Availability & Tolerance

Transactional: RDBMS: Oracle with **MVCC** (Multi Version Concurrency Control Along **with CDC - Change Data Capture** which uses **Golden Gate** works without intrusiveness in case of data insert, update or delete - DML operations)., PostgreSQL

Non Transactional: NoSQL such as MongoDB (Document Based)

1. **Code Quality:**
2. Sonar – Scanner, Lint (For On the Fly notification / suggestions) , Sonar Qube Server Integration with Quality Gate & Profile
3. Veracode – Security Vulnerability with both Code based such as SQL Injection, OS command Injection & other HTML encoding etc., along with SCA (Software Component Analysis) Vulnerabilities like Log4j with JNDI etc.,
4. Manual vulnerability checks such as Code Comments, Readabilities, Simplicity, and other robust Patterns etc.,
5. **DevOps**: Jenkins with **Maven**, **Gradle** (For fast build, Build Cache & For Multi Modular Project), SonarQube, Veracode, **Lattix**, **JFrog** **Artifactory** & **Argo CD** (For Continuous Deployment)
6. **JDK 1**.8 with Streams & Executor for concurrency then JDK 21 – For **Virtual Thread**
7. **APM –** I have created my own tool of APM with the use of Java **reflection** API & **Instrumentation** API to identify the runtime classes & objects with its complete details along with their runtime params, memory consumptions etc., This I have used for Developer machine. Client used **AppDynamics** in other Dev, QA, Stage, Pre-Prod & Prod environments. Thru which we can identify the clear cal; trace from UI 🡪 Microservice. Controller, Service, DAO (Hibernate Layer with it’s dynamic Query params etc.,)
8. **Technical Team Management:** Java 2.0 to 21, AWS – till 2025 Jan, Angular17, Kafka 2.8, Spring 4.x, Swagger Open API, Lombok – to reduce boiler plate code etc.,
9. **Domain :** Core Banking Sift Connect with java wife API, Forex with cross currency pair – Clients CBD (Commercial Bank of Dubai, State Bank of Mauritius), Razor Pay – Payment App using Flurt, worked on Corporate Banking **for First Gulf Bank – Abu Dhabi** for all **Conventional**, **Islamic** & **HNI banking** along with **Temenos T24 by 2012 to 2013 July**

**Previous Employment Details:**



**Education Details:**

