**About myself:**

I’m Manjunath Kamisetty, a Solution Architect with over 21 years of experience in IT, specializing in architecture, cloud-native solutions, and Agile delivery. I’ve spent the last 15+ years at AT&T, where I’ve played multiple roles across multiple transformation programs.

I began my journey in the Credit Systems area, where I spent nearly a decade design and developed multiple major/minor releases as well one of the critical modernizing project migrating legacy mainframe platforms to CDP platform and driving architectural improvements as well as designed and implemented Credit Risk management system. I then had a brief stint in OPUS, followed by a role in the Data Integration team, where I worked on data sync from salesforce to Snowflake.

Later, I transitioned into the Enterprise Architecture team for the Customer Connect transformation program. Later, I served as a Technical Architect in the Customer Connect program, where I delivered technical design for initiatives in Broadband, AIA Change Plan, and other strategic areas—focusing on Salesforce, MuleSoft, Snowflake, and Azure-based solutions.

In parallel, I also serve as the Engagement Delivery Lead for Cognizant’s portfolio within AT&T and DIRECTV, where I oversee delivery governance, stakeholder engagement, and team performance across multiple programs.

I’m also certified across multiple domains including AWS, Azure, Salesforce data cloud, and SAFe Agile, which helps me stay current and bring best practices into every engagement.

Beyond technical delivery, I’m passionate about mentoring teams, fostering innovation, and aligning technology with business strategy. I bring a strong mix of technical depth, leadership, and stakeholder engagement, making me well-equipped to drive enterprise-level initiatives.

I actively pursue certifications, attend webinars, and collaborate with product and engineering teams. I also explore emerging technologies like GenAI, AI/ML, and cloud-native frameworks to stay ahead of the curve.

**Architecture principles:**

Alignment with Business Objectives:  
Architecture Principles should directly support and reflect the organization's overall business strategy and goals. consistency in design and implementation, leading to better integration, reduced complexity, and easier maintenance. Encouraging the reuse of components and promoting modular design reduces development time and costs, and enhances flexibility.

Principles should guide the design of systems that can adapt to future needs and growth, ensuring long-term value. Principles should address security concerns and ensure compliance with relevant regulations and standards. Emphasizing the importance of data accuracy, consistency, and reliability is crucial for informed decision-making.

Considering the long-term cost implications of design choices is essential for efficient resource allocation. Designing with the end-user in mind, focusing on usability, accessibility, and a positive user experience.

**High Cohesion**

**Definition**: Related functionality should be grouped together. **BSS Application**:

* Domain-Driven Design: Group related business logic
* Microservices: Each service handles a specific business domain
* Bounded Contexts: Clear boundaries between domains
* Shared Libraries: Common functionality in shared components

**Interview Answer**: “We group related functionality using domain-driven design. Our Customer Service contains all customer-related logic including profile management, preferences, and history. Our Billing Service handles all billing-related functionality including invoicing, payments, and revenue recognition. This makes the system easier to understand and maintain."

**Integration Principles**

**Loose Coupling**

**Definition**: Components should have minimal dependencies on each other.**BSS Application**:

* Event-Driven Architecture: Services communicate through events
* API Gateway: Centralized communication point
* Message Queues: Asynchronous communication
* Service Discovery: Dynamic service location

**Interview Answer**:"We achieve loose coupling through event-driven architecture. Services communicate through Kafka events rather than direct API calls. This means if the Billing Service is down, the Order Service can still process orders and queue billing events for later processing. We use an API Gateway to centralize communication and service discovery for dynamic routing."

**Event-Driven Architecture**

**Definition**: System behavior driven by events.

* Event Streaming: Kafka for event processing
* Event Sourcing: Store events as source of truth Kafka can serve as the event store, where all changes to application state are recorded as a sequence of immutable events
* CQRS: Separate command and query models - Separates the read and write models of an application, with Kafka often used to propagate events between the command and query sides.
* Reactive Systems: Responsive to events

**Interview Answer**: “We use event-driven architecture for real-time processing and loose coupling. When a customer places an order, we publish an OrderCreated event that triggers inventory updates, billing record creation, and service provisioning. This allows services to react to business events in real-time while maintaining independence."

**API-First Design**

**Definition**: APIs are the primary interface for system interaction.

* API Gateway: Centralized API management
* API Versioning: Support multiple API versions
* API Documentation: OpenAPI/Swagger specifications
* API Testing: Comprehensive API testing

**Interview Answer**:"We design APIs first, then build implementations. All our BSS services expose RESTful APIs with OpenAPI specifications. We use an API Gateway for centralized management, versioning, and security. This approach allows frontend teams to develop in parallel and enables easy integration with external partners."

**Fail Fast**

**Definition**: System should detect and report errors quickly.

* Input Validation: Validate data at service boundaries
* Circuit Breakers: Fail quickly when dependencies are down
* Health Checks: Monitor service health continuously
* Error Handling: Clear error messages and logging

**Interview Answer**:"We implement fail-fast principles through comprehensive input validation at service boundaries. If invalid data is received, we reject it immediately rather than processing it and failing later. We use circuit breakers to fail quickly when external services are down, and implement health checks to detect issues before they impact users."

**Cloud-Native Design**

**Definition**: Design for cloud environments and distributed systems.

* Containerization: Docker containers for all services
* Orchestration: Kubernetes for container management
* Microservices: Small, independent services
* API-First: APIs as first-class citizens

**Interview Answer**:"Our BSS system is designed cloud-natively. We use Docker containers for all services and Kubernetes for orchestration. This allows us to deploy anywhere - on-premises, AWS, Azure, or GCP. We follow microservices principles with small, independent services that can be developed, deployed, and scaled independently."

**Separation of Concerns**

Definition: Each component should have a single, well-defined responsibility’s Application:

* Customer Service: Only handles customer data and lifecycle
* Billing Service: Only manages billing and payment processing
* Order Service: Only processes orders and fulfillment
* Product Service: Only manages product catalog and pricing

"In BSS architecture, I apply separation of concerns by ensuring each microservice has a single responsibility. For example, our Customer Service only manages customer profiles and relationships, while our Billing Service handles all payment and invoicing logic. This makes the system more maintainable, testable, and allows teams to work independently."

**Open/Closed Principle**

**Definition**: Software entities should be open for extension but closed for modification.:

* Pricing Engine: Extensible for new pricing rules without modifying existing code
* Payment Gateway: Support new payment methods through plugins
* Notification System: Add new channels without changing core logic

**Interview Answer**: “Our pricing engine follows the open/closed principle. We can add new pricing rules like seasonal discounts or loyalty programs by creating new rule classes that implement our pricing interface, without modifying the existing pricing logic. This allows us to extend functionality while maintaining system stability."

**Interface Segregation Principle**

**Definition**: Clients should not be forced to depend on interfaces they don't use:

* Customer Interface: Separate interfaces for profile management and order history
* Product Interface: Different interfaces for catalog browsing and inventory management
* Billing Interface: Separate interfaces for invoicing and payment processing

**Interview Answer**:"We segregate interfaces based on client needs. For example, our mobile app only needs a lightweight customer interface for profile updates, while our admin portal needs a comprehensive interface with all customer management features. This prevents unnecessary dependencies and improves performance."

**Scalability**

**Definition**: System should handle increased load by adding resources:

* Horizontal Scaling: Multiple instances of each service
* Database Sharding: Distribute data across multiple databases
* Load Balancing: Distribute requests across service instances
* Caching: Reduce database load with Redis/Memcached

**Interview Answer**: “Our BSS system is designed for horizontal scalability. We can handle increased customer load by adding more instances of our Customer Service, Order Service, and Billing Service. We use database sharding to distribute customer data across multiple PostgreSQL instances, and Redis caching to reduce database load during peak times."

**Availability**

**Definition**: System should be operational when needed:

* High Availability: 99.9% uptime for critical services
* Fault Tolerance: Graceful handling of component failures
* Redundancy: Multiple instances of critical components
* Circuit Breakers: Prevent cascade failures

**Interview Answer**: “We ensure high availability through redundancy and fault tolerance. Each service runs in multiple instances across different availability zones. We implement circuit breakers to prevent cascade failures, and use health checks to automatically replace failed instances. Our target is 99.9% uptime for critical BSS functions."

**Performance**

**Definition**: System should respond quickly and efficiently:

* Response Time: < 200ms for API calls
* Throughput: Handle 10,000+ requests per second
* Optimization: Database query optimization, caching, CDN
* Monitoring: Real-time performance metrics

**Interview Answer**: “We optimize performance through multiple strategies. Our APIs respond in under 200ms through database query optimization and Redis caching. We use CDN for static content and implement connection pooling for database access. We monitor performance in real-time and automatically scale resources during peak loads."

**Security**

**Definition**: System should protect data and resources:

* Authentication: OAuth 2.0, JWT tokens
* Authorization: Role-based access control (RBAC)
* Data Protection: Encryption at rest and in transit
* API Security: Rate limiting, input validation

**Interview Answer**: “Security is built into every layer of our BSS architecture. We use OAuth 2.0 for authentication and JWT tokens for session management. RBAC ensures users only access authorized resources. All sensitive data is encrypted, and we implement rate limiting and input validation to prevent attacks."

**Dependency Inversion Principle**

**Definition**: High-level modules should not depend on low-level modules. Both should depend on abstractions:

* Service Layer: Depends on repository interfaces, not concrete implementations
* Business Logic: Depends on service interfaces, not concrete services
* External Integrations: Depends on adapter interfaces, not specific APIs

**Interview Answer**: “Our service layer depends on repository interfaces rather than concrete database implementations. This allows us to switch from PostgreSQL to MongoDB or add caching layers without changing our business logic. We inject dependencies through interfaces, making our system more flexible and testable."

**How to Tell Compelling Stories About Dealing with Conflicts**

**📋 The STAR Method for Conflict Stories**

**S - Situation**

* Set the context and background
* Describe the project, stakeholders, and environment
* Explain the business impact and urgency

**T - Task**

* Define your role and responsibilities
* Explain what needed to be accomplished
* Describe the constraints and challenges

**A - Action**

* Detail the specific steps you took
* Explain your approach and methodology
* Describe the communication and collaboration strategies

**R - Result**

* Quantify the outcomes and benefits
* Explain lessons learned
* Describe the long-term impact

**Scenario: Microservices vs Monolithic Architecture**

**Situation**: "In a large telecom BSS modernization project, we had a conflict between the development team wanting to implement microservices and the operations team insisting on a monolithic approach due to deployment complexity."

**Task**: "As the Solution Architect, I needed to resolve this conflict while ensuring the solution met both technical requirements and operational constraints."

**Action**:

* "I organized a technical workshop with both teams to understand their concerns
* Created a detailed comparison matrix showing pros/cons of each approach
* Proposed a hybrid approach: start with modular monolith, then gradually extract services
* Implemented a proof-of-concept to demonstrate the feasibility
* Established clear migration criteria and timeline"

**Result**:

* "We delivered the initial phase 30% faster than planned
* Reduced deployment complexity by 60%
* Successfully migrated 3 services to microservices within 6 months
* Improved team collaboration and reduced future conflicts"

**2. Stakeholder and Business Conflicts**

**Scenario: Budget vs Feature Requirements**

**Situation**: "The business stakeholders wanted a comprehensive BSS solution with advanced analytics, while the finance team had strict budget constraints that would only allow basic functionality.

**Task**: "I needed to find a way to deliver maximum business value within the budget constraints.

**Action**:

* Conducted detailed requirements analysis and prioritized features by business impact
* Proposed a phased delivery approach with MVP first, then incremental enhancements
* Negotiated with vendors for flexible licensing models
* Identified open-source alternatives for non-critical components
* Created a business case showing ROI for each phase"

**Result**:

* "Delivered MVP 25% under budget
* Achieved 90% of business requirements in Phase 1
* Secured additional funding for Phase 2 based on early success
* Established a sustainable delivery model for future projects"

**Scenario: Legacy System Integration**

**Situation**: "The business wanted to keep the existing legacy billing system while implementing new customer management capabilities, but the technical team wanted to replace everything."

**Task**: "I needed to balance business continuity with technical modernization."

**Action**:

* "Conducted a comprehensive analysis of the legacy system's capabilities and limitations
* Proposed an API-first integration approach to gradually modernize components
* Implemented a service abstraction layer to decouple new and legacy systems
* Created a migration roadmap with clear milestones and rollback plans
* Established a parallel run period to ensure system stability"

**Result**:

* "Reduced integration risks by 70%
* Delivered new customer management features 40% faster
* Maintained 99.9% system availability during transition
* Created a reusable integration pattern for future legacy modernizations"

**Team and Process Conflicts**

**Scenario: Agile vs Waterfall Methodology**

**Situation**: "The development team wanted to adopt Agile methodology, while the business stakeholders preferred Waterfall due to their experience with fixed-scope projects."

**Task**: "I needed to bridge the gap between Agile development practices and business expectations."

**Action**:

* "Organized training sessions for business stakeholders on Agile principles
* Implemented a hybrid approach: Agile for development, Waterfall-style reporting for business
* Created clear communication protocols and regular demo sessions
* Established a change management process that accommodated both methodologies
* Implemented tools for better visibility and tracking"

**Result**:

* "Improved delivery predictability by 50%
* Increased stakeholder satisfaction by 60%
* Reduced scope creep by 40%
* Established a successful hybrid methodology for future projects"

**Scenario: Cross-Functional Team Conflicts**

**Situation**: "During a BSS implementation, the frontend team, backend team, and database team had conflicting priorities and were working in silos, causing delays and quality issues."

**Task**: "I needed to break down silos and align all teams toward common goals."

**Action**:

* "Reorganized teams into cross-functional squads with shared objectives
* Implemented daily stand-ups and regular cross-team retrospectives
* Created shared success metrics and KPIs
* Established a technical architecture review board with representatives from each team
* Implemented pair programming and knowledge sharing sessions"

**Result**:

* "Reduced delivery time by 35%
* Improved code quality by 45%
* Increased team collaboration scores by 70%
* Reduced production incidents by 60%"

**Handled things not in your wheelhouse or comfort zone:**

**Key Principles:**

1. Embrace the Challenge: View unfamiliar situations as growth opportunities
2. Leverage Your Foundation: Use existing skills as a springboard
3. Build a Support Network: Identify and engage with subject matter experts
4. Iterative Learning: Break down complex topics into manageable pieces
5. Document and Share: Create knowledge artifacts for future reference

**The ADAPT Framework**

**A - Assess the Situation**

* Understand the scope and complexity of the challenge
* Identify what you know vs. what you need to learn
* Assess timeline and resource constraints
* Evaluate potential risks and mitigation strategies

**D - Develop a Learning Plan**

* Break down the topic into smaller, manageable components
* Identify key learning resources and experts
* Create a structured timeline for knowledge acquisition
* Set measurable milestones and checkpoints

**A - Acquire Knowledge Rapidly**

* Use multiple learning channels (documentation, experts, hands-on practice)
* Focus on practical application over theoretical knowledge
* Create proof-of-concepts to validate understanding
* Leverage existing patterns and best practices

**P - Practice and Apply**

* Start with low-risk scenarios
* Build confidence through incremental success
* Seek feedback and iterate quickly
* Document lessons learned and best practices

**T - Transfer and Teach**

* Share knowledge with team members
* Create documentation and reusable artifacts
* Mentor others facing similar challenges
* Establish yourself as a subject matter expert

I believe that stepping outside your comfort zone is essential for growth and success.

When I encounter unfamiliar situations, I follow a systematic approach:

First, I assess the situation and identify what I need to learn. For example, when I was asked to implement Kubernetes for BSS microservices, I recognized that while I had strong microservices experience, I needed to learn container orchestration.

Second, I create a structured learning plan. I broke down Kubernetes into manageable components, set up hands-on labs, and connected with experts in the organization.

Third, I practice with low-risk scenarios. I started with a pilot implementation of one BSS service before expanding to the entire system.

This experience taught me that with the right approach and mindset, I can quickly adapt to new technologies and domains while delivering value to the business.

**Key Strengths:**

1. Loyalty – Someone wants to be part of the organization for long time
2. Fitting into a team quickly: I can able to quickly adapt to the team’s way of working.
3. Having Passion for my work: I am passionate about what I do and takes pride of my work and genuinely wants to do best job possible for the company I work.
4. Good Problem Solver: Takes responsibility for difficult challenges instead of leaving them for others to deal with.
5. Being Cost Conscious: I was very conscious about company resources.
6. Supportive Collaborators: I am a great team player and help the co-workers/colleague when asked for any information as well as train new team members whey they joined company as mentor and also Cross-Functional Collaboration, Product management Collaboration as well Technical mentorship and coaching. Beyond technical delivery, I’m passionate about mentoring teams, fostering innovation, and aligning technology with business strategy. I bring a strong mix of technical depth, leadership, and stakeholder engagement, making me well-equipped to drive enterprise-level initiatives.
7. Commercial awareness: Understands how business operates and align with the operating model.
8. Leadership Capabilities: I will take lead in difficult situations and do all I can to resolve the problem instead of passing it to management to deal with.
9. Emerging Technologies & Innovation: I’m also certified across multiple domains including AWS, Azure, Salesforce data cloud, and SAFe Agile, which helps me stay current and bring best practices into every engagement. Beyond technical delivery, I’m passionate about mentoring teams, fostering innovation, and aligning technology with business strategy. I bring a strong mix of technical depth, leadership, and stakeholder engagement, making me well-equipped to drive enterprise-level initiatives. I actively pursue certifications, attend webinars, and collaborate with product and engineering teams. I also explore emerging technologies like GenAI, AI/ML, and cloud-native frameworks to stay ahead of the curve.

**Weakness:**

1. **Public Speaking:** I am not good at public speaking or giving talks to large group of people.
2. **Giving People Honest Feedback:**  I don’t enjoy giving honest feedback as I believe it will lead to distance myself from potential confrontational situations.
3. **Saying Yes:** I am person who will say “Yes” automatically when people ask for help which often results in me taking many responsibilities.
4. **Finding Difficult to let go Project:** I don’t like letting go of projects because of me keep checking over them. I am working on this weakness to let go project and move on to next one.
5. **Being too self-critical:** I am tough on myself and keep analyzing on why I did the things that way. While its good to reflect on the action, beating myself of things are already done is not productive.