Technical Architecture Career Learning Guide

Learning Path Overview

This guide is structured in 4 progressive phases, each building upon the previous one. The estimated timeline is 12-18 months of dedicated study and practice.

Phase 1: Foundation (Months 1-3)

Build core understanding of systems and data

1.1 Database & Storage Fundamentals

Week 1-2: Core Database Concepts

- **SQL vs NoSQL**: Learn when to use relational vs document/key-value stores
- **ACID Properties**: Atomicity, Consistency, Isolation, Durability
- **BASE Model**: Basically Available, Soft state, Eventually consistent
- **OLAP vs OLTP**: Analytical vs transactional processing patterns

Week 3-4: Database Internals

- **Indexing & B+ Trees**: How databases optimize queries
- **Transaction Isolation**: Read uncommitted, committed, repeatable read, serializable
- **WAL (Write Ahead Log)**: Understanding database durability mechanisms
- **Read/Write Patterns**: Optimizing for different access patterns

Week 5-6: Data Management

- **Data Modeling**: Normalization, denormalization strategies
- **Data Partitioning**: Horizontal and vertical partitioning
- **Data Retention**: Archival strategies and compliance
- **Hot/Cold Storage**: Cost-effective data lifecycle management

Practical Exercises:

- Set up PostgreSQL and MongoDB instances
- Design schemas for e-commerce and social media use cases
- Implement different isolation levels and observe behavior
- Practice query optimization with EXPLAIN plans

1.2 Storage Systems

Week 7-8: Modern Storage

- **Object Storage & S3 Basics**: Understanding cloud storage paradigms
- **File Systems**: Traditional vs distributed file systems
- **Backup & Restore**: Strategies for data protection
- **Bloom Filters**: Probabilistic data structures for efficiency

Practical Exercises:

- Deploy MinIO (S3-compatible storage)
- Implement backup strategies with different RTO/RPO requirements
- Build a simple bloom filter implementation

Phase 2: Distributed Systems Core (Months 4-7)

2.1 Scaling Fundamentals

Week 9-10: Basic Scaling Concepts

- **Load Balancing**: Round-robin, weighted, least connections algorithms
- **Caching**: In-memory, distributed caching strategies
- **Redis/Memcached**: Implementing high-performance caching
- **CDN**: Content delivery and edge computing concepts

Week 11-12: Data Distribution

- **Sharding**: Horizontal partitioning strategies
- **Consistent Hashing**: Distributed hash tables and ring topology
- **Replication**: Master-slave, master-master patterns
- **Leader-Follower Replication**: Consensus and consistency

Practical Exercises:

- Set up NGINX load balancer with multiple backend servers
- Implement Redis cluster with sharding
- Build consistent hashing algorithm
- Configure PostgreSQL streaming replication

2.2 Distributed Systems Theory

Week 13-14: Theoretical Foundations

- **CAP Theorem**: Consistency, Availability, Partition tolerance trade-offs
- **Consistency Models**: Strong, eventual, causal consistency
- **Eventual Consistency**: Vector clocks, conflict resolution
- **Distributed Transactions**: Two-phase commit, Saga patterns

Week 15-16: Advanced Patterns

- **Leader Election**: Raft, PBFT consensus algorithms
- **Fault Tolerance**: Byzantine fault tolerance, failure detection
- **Partitioning**: Network partitions and split-brain scenarios

Practical Exercises:

- Implement Raft consensus algorithm (simplified version)
- Build eventually consistent system with conflict resolution
- Simulate network partitions and observe system behavior

2.3 Architecture Patterns

Week 17-18: Service Architecture

- **Monolith vs Microservices**: Trade-offs and migration strategies
- **Microservices**: Service boundaries, data ownership
- **Service Discovery**: Registry patterns, health checks
- **Database Scaling**: Read replicas, write scaling, CQRS

Practical Exercises:

- Decompose monolithic application into microservices
- Set up service discovery with Consul or etcd
- Implement CQRS pattern for read/write separation

Phase 3: Communication & Integration (Months 8-10)

3.1 API Design & Management

Week 19-20: API Fundamentals

- **REST vs gRPC**: HTTP vs binary protocol trade-offs
- **API Versioning**: Backward compatibility strategies
- **Protocol Buffers**: Schema evolution and serialization
- **Serialization**: JSON, Avro, MessagePack performance comparison

Week 21-22: API Infrastructure

- **API Gateway**: Routing, transformation, aggregation
- **Rate Limiting**: Token bucket, sliding window algorithms
- **API Rate Limits**: Per-user, per-endpoint strategies
- **Throttling**: Graceful degradation under load

Week 23: Security & Standards

- **JWT**: Stateless authentication and authorization
- **OAuth**: Delegation and third-party access
- **CORS**: Cross-origin resource sharing policies
- **API Security**: Input validation, SQL injection prevention

Practical Exercises:

- Build REST API with OpenAPI specification
- Implement gRPC service with Protocol Buffers
- Set up Kong or AWS API Gateway
- Implement JWT authentication with refresh tokens

3.2 Asynchronous Communication

Week 24: Messaging Patterns

- **Message Queues**: Point-to-point vs publish-subscribe
- **Asynchronous Processing**: Event-driven architectures
- **Dead Letter Queue**: Error handling and poison messages
- **Fan-out/Fan-in**: Scatter-gather communication patterns

Week 25-26: Advanced Communication

- **WebSockets**: Real-time bidirectional communication
- **Long Polling**: Alternative to WebSockets for real-time updates
- **Service Mesh**: Istio, Linkerd for service communication
- **Queueing**: RabbitMQ, Apache Kafka implementation patterns

Practical Exercises:

- Build chat application with WebSockets
- Implement event sourcing with Apache Kafka
- Set up Istio service mesh in Kubernetes
- Create dead letter queue handling system

Phase 4: Production & Reliability (Months 11-12)

4.1 Reliability Engineering

Week 27-28: Reliability Patterns

- **Circuit Breaker**: Preventing cascade failures
- **Retry Patterns**: Exponential backoff, jitter
- **Idempotency**: Ensuring safe retries
- **Graceful Degradation**: Maintaining core functionality under stress

Week 29-30: Failure Management

- **Failover**: Automatic and manual failover strategies
- **Health Checks**: Liveness and readiness probes
- **Heartbeats**: Detecting and handling node failures
- **Retry Logic**: Smart retry with circuit breaker integration

Practical Exercises:

- Implement circuit breaker pattern
- Build health check endpoints
- Create retry mechanism with exponential backoff
- Test failover scenarios

4.2 Observability & Operations

Week 31-32: Monitoring & Metrics

- **Metrics**: Application and infrastructure monitoring
- **Logging**: Structured logging, log aggregation
- **Distributed Tracing**: Request flow across services
- **Monitoring**: Prometheus, Grafana, ELK stack

Week 33: Performance & Scaling

- **Load Testing**: JMeter, K6 for performance validation
- **Autoscaling**: Horizontal and vertical scaling triggers
- **SLO/SLI/SLA**: Service level objectives and indicators
- **Error Budgets**: Balancing reliability and feature velocity

Practical Exercises:

- Set up monitoring stack (Prometheus + Grafana)
- Implement distributed tracing with Jaeger
- Create comprehensive load testing suite
- Define SLOs and error budgets for services

4.3 Deployment & Incident Management

Week 34: Deployment Strategies

- **Blue-Green Deployment**: Zero-downtime deployments
- **Canary Deployments**: Gradual rollout strategies
- **Rollbacks**: Quick recovery from failed deployments
- **Chaos Engineering**: Proactive failure testing

Week 35-36: Operations

- **Incident Response**: Runbooks and escalation procedures
- **Alerting**: Alert fatigue prevention and smart alerting
- **Chaos Engineering**: Netflix Chaos Monkey concepts

Practical Exercises:

- Implement blue-green deployment pipeline
- Set up canary deployment with automatic rollback
- Create incident response runbooks
- Practice chaos engineering scenarios

Practical Project Recommendations

Project 1: Distributed E-commerce Platform (Months 4-6)

Build a microservices-based e-commerce system including:

- User service with authentication
- Product catalog with search
- Order processing with inventory management
- Payment service with external integration
- Notification service

Technologies: Spring Boot/Node.js, PostgreSQL, Redis, RabbitMQ, Docker

Project 2: Real-time Analytics Platform (Months 8-10)

Create a system for processing streaming data:

- Data ingestion service
- Stream processing with Apache Kafka
- Real-time dashboards
- Historical data analysis
- Alerting system

Technologies: Apache Kafka, Apache Spark, InfluxDB, Grafana, Kubernetes

Project 3: Multi-tenant SaaS Platform (Months 11-12)

Design a scalable multi-tenant application:

- Tenant isolation strategies
- Shared vs dedicated resources
- Billing and metering
- Performance monitoring per tenant
- Automated scaling

Technologies: AWS/GCP, Kubernetes, Istio, Prometheus, Terraform

---

Learning Resources

Books (Priority Reading)

1. **"Designing Data-Intensive Applications" by Martin Kleppmann** - Fundamental concepts
2. **"Building Microservices" by Sam Newman** - Service architecture
3. **"Site Reliability Engineering" by Google** - Production operations
4. **"Release It!" by Michael Nygard** - Production-ready software

Online Resources

- **AWS Architecture Center**: Real-world architecture patterns
- **High Scalability**: Case studies from major tech companies
- **Papers We Love**: Academic papers on distributed systems
- **Kubernetes Documentation**: Container orchestration

Tools to Master

- **Containerization**: Docker, Kubernetes
- **Infrastructure as Code**: Terraform, CloudFormation
- **CI/CD**: Jenkins, GitLab CI, GitHub Actions
- **Monitoring**: Prometheus, Grafana, ELK Stack
- **Service Mesh**: Istio, Linkerd

Career Progression Milestones

Junior/Mid-level (Months 1-6)

- ✅ Understand database fundamentals and scaling basics
- ✅ Can design simple distributed systems
- ✅ Familiar with caching and load balancing
- ✅ Basic understanding of microservices

Senior Level (Months 7-10)

- ✅ Design complex distributed systems
- ✅ Make informed trade-off decisions (CAP theorem)
- ✅ Implement reliable communication patterns
- ✅ Design APIs and integration strategies

Lead/Principal Level (Months 11-12+)

- ✅ System-wide architecture decisions
- ✅ Production reliability and incident management
- ✅ Performance optimization and capacity planning
- ✅ Technical leadership and mentoring

Assessment & Practice

Monthly Self-Assessment Questions

1. Can I explain the trade-offs between consistency and availability?
2. How would I design a system to handle 1M requests per second?
3. What monitoring strategy would I implement for microservices?
4. How do I ensure data consistency across distributed services?

Hands-on Practice Schedule

- **Week 1-2**: Theory and reading
- **Week 3**: Hands-on implementation
- **Week 4**: Project work and reflection

Interview Preparation Topics

- System design scenarios (URL shortener, chat system, etc.)
- Trade-off discussions (SQL vs NoSQL, sync vs async)
- Failure scenarios and recovery strategies
- Performance optimization techniques

This guide provides a structured path to mastering technical architecture. Focus on building practical experience alongside theoretical knowledge, and don't hesitate to dive deeper into areas that align with your specific career interests.