Migrating Monolithic SaaS to Serverless: Achieving Scalability, Cost Reduction, and Development Agility on AWS

BY: **Srikar Kompella**

Bio

- I am a Senior Software Engineer with over 13 years of expertise in developing developing innovative software solutions and specializes in building scalable scalable applications, with a focus on payment solutions for Prime Video and Video and messaging platforms at Twilio. Skilled in AWS technologies like like Lambda and DynamoDB and a strong background in full-stack development development across multiple languages, including Python, Java, Kotlin, C# and Kotlin, C# and Go. I am an AWS Cloud Practitioner and Oracle Certified Java Certified Java Developer, and specialize in leadership, mentoring, and contributions to high-visibility projects.

Email reachsrikarkompella@gmail.com

Why Choose Golang for Serverless Architectures?

- High Performance
- Efficient Memory Management
- Concurrent Programming

Combining Golang with AWS

- Lambda Function Support
- Cost Efficiency
- Seamless Integration with AWS Services

Golang vs. Other Languages (Java, Node.js, Python)

- Performance
- Java
- Node.js
- Python

Why Migrate to Serverless Architecture?

80%

Cost Reduction

Significant savings through pay-for-execution pricing

70%

Faster Deployment

Streamlined pipelines for rapid market responsiveness

60%

Improved Scalability

Automatic handling of demand spikes without pre-provisioning provisioning

50%

Operational Gains

Reduced maintenance letting teams focus on innovation

Core AWS Serverless Technologies

AWS Lambda

Executes code in response to triggers without without server management. Eliminates infrastructure overhead while providing automatic scaling and pay-per-execution pricing.

Amazon DynamoDB

Managed NoSQL database with consistent millisecond performance at any scale. Supports document and key-value models with automatic scaling that adapts to application demands.

API Gateway

Manages API creation and processing to connect applications with backend services. services. Provides RESTful and WebSocket APIs WebSocket APIs with built-in authorization, authorization, throttling, and monitoring.

Enhancing Operational Efficiency with AWS S3 and and CloudWatch

AWS S3 (Simple Storage Service)

Data Storage and Backup: S3 provides scalable, cost-effective effective storage for application data, backups, and logs during during serverless migration.

- **Serverless Integration**: Seamlessly works with AWS Lambda for efficient data operations without infrastructure management.
- **Scalability & Durability**: Offers unlimited scalability with 99.999999999% durability, ensuring constant data availability. availability.

AWS CloudWatch

- **Real-time Monitoring and Insights**: Provides essential observability into serverless application performance.
  - **Logs & Metrics**: Collects data from Lambda, DynamoDB, EC2 and other services for real-time application insights.
  - **Dashboards**: Offers customizable visualizations of multiple multiple AWS resources for comprehensive system monitoring.

Migration Strategy: Decomposition

1

Identify Service Boundaries

Use domain-driven design to map business capabilities and establish service boundaries. Prioritize components with minimal coupling as initial migration candidates.

2

Extract Microservices

Refactor components into independent microservices with dedicated data stores. Develop versioned APIs through API Gateway for inter-service inter-service communication.

3

Implement Serverless Functions

Convert microservices to Lambda functions aligned with business operations. Design each function with single responsibility, appropriate triggers, and optimized execution.

4

Transition Traffic Gradually

Use strangler pattern with feature flags and canary deployments to redirect traffic incrementally. Monitor metrics closely to enable quick rollbacks if quick rollbacks if needed.

Event-Driven Architecture Design

Event Production

Services publish events on state changes changes

Event Processing

Lambda functions process events asynchronously

State Transformation

System state updates based on event data data

Cascading Events

Updated states trigger new events

Database Migration Considerations Considerations

Relational Database	Migration Path	Benefits
MySQL/PostgreSQL	Aurora Serverless	Auto-scaling, compatibility
Oracle/SQL Server	DynamoDB + Aurora	Cost savings, performance
MongoDB	DocumentDB/DynamoDB	Managed service, scalability
Redis/Memcached	ElastiCache/DAX	Caching layer, reduced latency

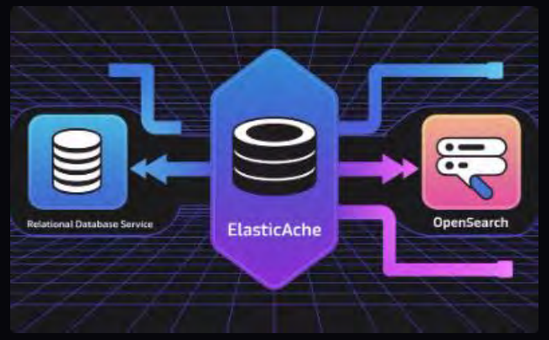
Hybrid Deployment Models

Serverless Functions

Lambda for event-driven, scalable workloads with automatic provisioning and pay-per-use billing

Container Services

ECS/EKS for stateful, complex microservices requiring orchestration and consistent runtime environments



Managed Services

RDS, ElastiCache, OpenSearch for specialized data needs with reduced operational overhead

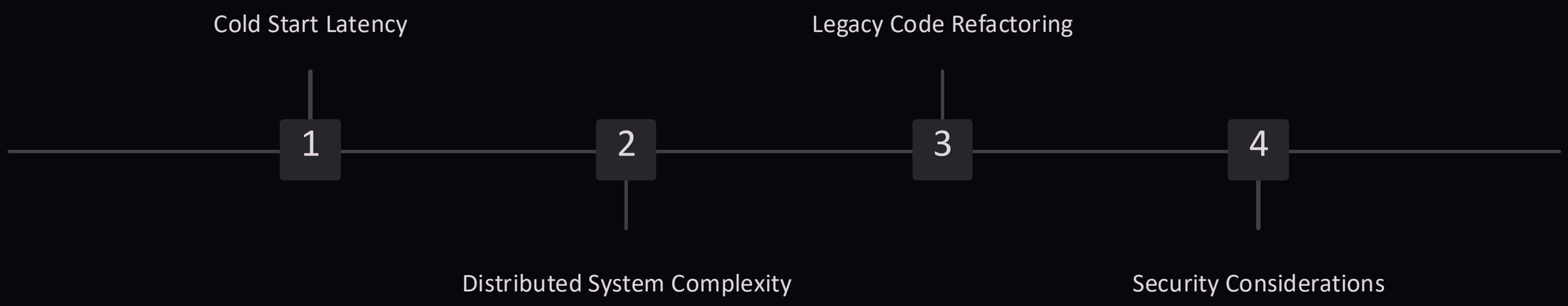
Traditional Infrastructure Infrastructure

EC2 for resource-intensive or legacy workloads requiring full full control over compute resources

DevOps Integration & CI/CD

- Infrastructure as Code
- Continuous Integration
- Deployment Automation
- Monitoring & Observability

Overcoming Migration Challenges



Case Study: E-Commerce Platform Migration

- Initial Assessment
- The e-commerce platform faced scalability failures during peak seasons, with 400% slower transaction processing. Analysis identified bottlenecks in the monolithic architecture, prioritizing inventory and payment systems for migration.
- First Migration Phase
- Payment processing was extracted into Lambda functions with API Gateway using event-driven architecture. Results: 30% lower lower infrastructure costs and 50% faster response times during peak traffic.
- Database Transition
- Product catalog migrated from MySQL to DynamoDB with optimized access patterns. A dual-write approach ensured zero-downtime migration while maintaining data consistency across systems.
- Complete Serverless Architecture
- The serverless implementation delivered 80% cost reduction during normal operations while handling 10x traffic surges without without performance issues. Development cycles improved from monthly releases to multiple daily deployments.

Roadmap for Successful Serverless Migration

1

Assess & Plan

Evaluate architecture, identify migration candidates

2

Initial Pilot

Implement proof-of-concept with isolated service

3

Expand & Refine

Scale to critical workloads, optimize architecture patterns

4

Enterprise Transformation

Institutionalize practices, embrace cloud-native culture

Thank you