



# Building Self-Healing Cache Infrastructure

## How Platform Engineering Reduced Latency While Scaling to Millions

A comprehensive approach to building intelligent caching platforms that have become the backbone of modern development infrastructure at a large-scale production environment.

**Shailin Saraiya**

Roku Inc.



# The Challenge of Modern Platform Engineering

## The Scale Problem

Digital platforms often face rapid increases in user engagement, putting significant strain on traditional infrastructure approaches:

- Database connections reaching limits during peak traffic
- Unpredictable API response times
- Engineering teams spending increasing time on infrastructure issues instead of building features

## Traditional Caching Limitations

- Static TTL management with hardcoded values
- Manual scaling requiring reactive intervention
- Limited observability for developers
- Inconsistent implementation across teams

# Vision for Self-Healing Infrastructure

## Self-Service Platform

Create a platform that developers could use without deep caching expertise

## Intelligent Systems

Implement systems that adapt automatically to changing traffic patterns

## Self-Healing Mechanisms

Build mechanisms that maintain high availability without human intervention

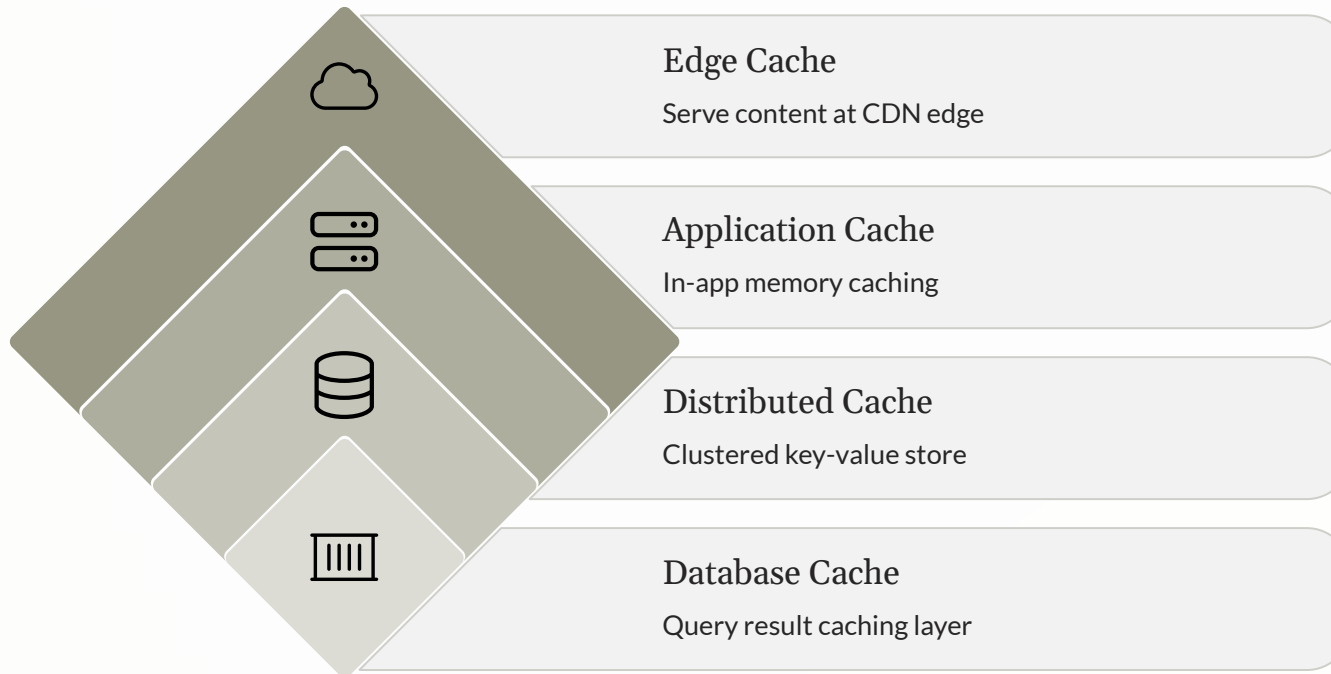
## Comprehensive Observability

Provide visibility that enables data-driven optimization

## Effortless Scaling

Scale effortlessly to support our projected growth trajectory

# Designing the Intelligent Caching Platform



## Key Architectural Principles

**Layered Caching Strategy:** Multi-tiered approach from edge to database level

**Service Mesh Integration:** Enabling automatic service discovery, intelligent routing, circuit breakers, and retry mechanisms

**Event-Driven Architecture:** Real-time event streaming for cache coherency, asynchronous cache warming, and event sourcing for state reconstruction

**Directory-Based Cache Coherency:** Sophisticated protocol maintaining 99.9% accuracy



# Machine Learning for Smarter Caching

## Predictive Cache Warming

Use ML models to anticipate demand based on access patterns, popularity signals, and temporal trends, allowing caches to be pre-populated before requests arrive.

## Adaptive TTL Management

Replace static expiration rules with dynamic TTLs that adjust automatically based on data volatility, access frequency, and system performance.

## Anomaly Detection

Apply ML-based monitoring to detect unusual traffic patterns, cache poisoning attempts, or signs of performance degradation early.



# Key Implementation Patterns

## Automated Deployment Pipeline

- Infrastructure as Code with declarative configurations
- GitOps workflow with automated validation and rollbacks
- Zero-downtime deployments with blue-green strategy

## Developer-Friendly APIs

```
@CacheableService
public class SampleService {
    @Cacheable(strategy = "adaptive",
               mlOptimized = true)
    public Entity getEntity(String id) {
        return database.fetchUserEntity(id);
    }
}
```

## Self-Healing Mechanisms

Node failure detection within milliseconds, automatic replica promotion, traffic rerouting without client awareness, and state reconstruction from event logs.



# Observability and Monitoring

## Comprehensive Dashboards

Key elements of developer-friendly observability dashboards include:

- Hit/miss ratios by service and endpoint
- Latency percentiles (p50, p95, p99)
- Cache utilization and memory pressure
- Error rates and timeout tracking

## Actionable Insights

Monitoring systems can provide recommendations such as:

"Cache hit rate for endpoint /api/entity/{id} dropped 15% after deployment abc123"

"Introducing caching to high-traffic endpoints can significantly improve response times."

# Impact of Self-Healing Caching

## Latency Reduction

Achieved substantial reductions in response times for high-traffic workloads.

## Database Load

Dramatically lowered database query volume through effective caching strategies.

## Cache Hit Rate

Sustained industry-leading cache hit ratios during peak usage periods.

## Global Latency

Improved response times for international users by leveraging geo-distributed cache clusters.





# Machine Learning for Caching (Deep Dive)

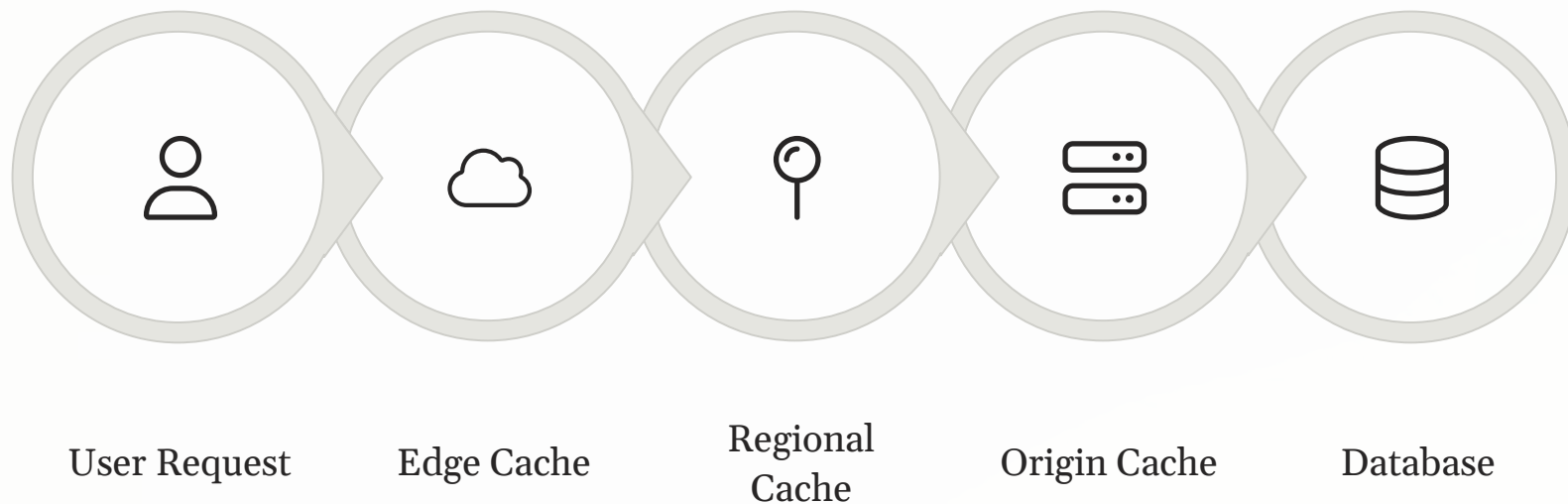
## Predictive Algorithm Architecture

- **Neural Network:** Neural networks applied to historical access patterns and temporal trends
- **Reinforcement Learning:** Reinforcement learning for optimizing cache eviction strategies
- **Clustering:** Clustering techniques to segment workloads and enable targeted cache warming

## Potential Benefits of ML-Driven Caching

- Higher cache hit ratios compared to static caching methods
- Reduced memory footprint through more efficient cache management
- Improved accuracy in anticipating traffic surges and workload spikes
- Better cost efficiency by optimizing infrastructure resource utilization

# Edge Computing Integration



## Common Challenges and Approaches

### Cache Coherency at Scale

Techniques such as eventual consistency, vector clocks, conflict resolution, and read repair are used to maintain accuracy.

### Limited Edge Resources

Adaptive caching strategies, intelligent eviction policies, and compression techniques help optimize constrained environments

### Network Partitions

Resilient architectures can enable autonomous edge operation during partitions, with state reconciliation and graceful degradation upon reconnection.



# Platform Engineering Best Practices

## Self-Service Infrastructure

- One-click provisioning and interactive configuration tools
- Cost estimation and performance simulation environments

## Operational Excellence

- Toil reduction through self-healing mechanisms
- Automatic escalation for complex issues
- Runbook automation and predictive maintenance

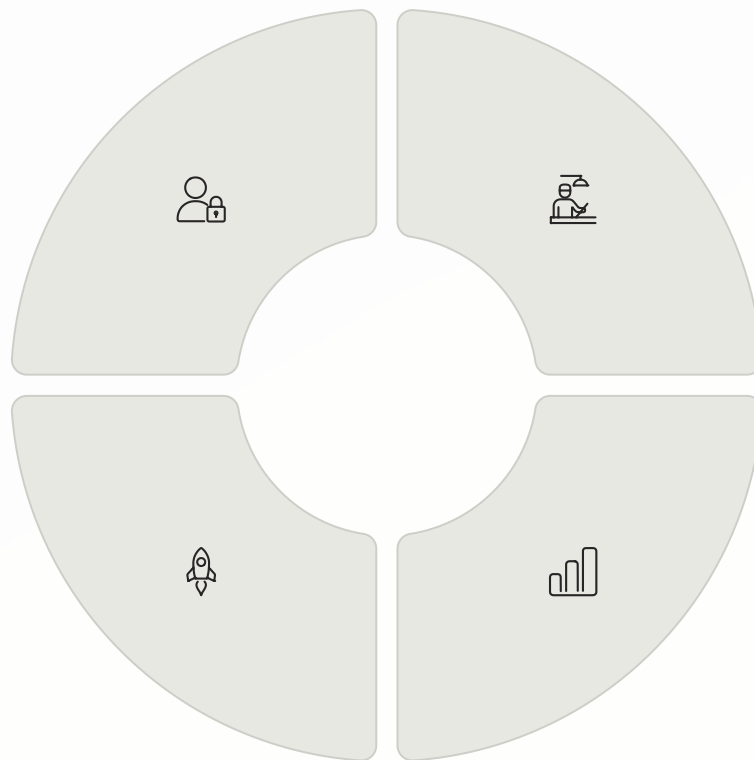
# Cultural Transformation

## Developer Empowerment

- Encourage teams to define and manage their own caching strategies
- Abstract infrastructure complexity so developers can focus on business logic

## Business Impact

- Faster delivery of new features
- Lower infrastructure and operational costs
- Improved user experience and engagement



## Operational Mindset

- Treat reliability as a shared responsibility across teams
- Promote data-driven decision making and continuous improvement

## Measurement Focus

- Aim for high availability targets (e.g., “four nines” uptime)
- Minimize provisioning times
- Reduce support overhead through automation and observability

# Key Lessons



## Start with Developer Experience

Simple APIs trump feature richness, good defaults matter more than configurability, and documentation is critical.



## Incremental Migration Strategy

Support legacy systems during transition, provide clear migration paths, and automate migration where possible.



## Observability First

Metrics drive architectural decisions, visibility enables self-service, and data validates assumptions.

## Challenges Overcome

**Technical:** Cache stampede prevention, hot key management, memory fragmentation, network congestion

**Organizational:** Balancing standardization with flexibility, addressing skills gaps, overcoming resistance to shared platforms



# Future Directions in Caching and Infrastructure

## Near-Term Trends

- GraphQL caching optimization
- Enhanced ML models for content prediction
- Automated capacity management
- Cross-region active-active support

## Mid-Term Innovations

- Serverless cache functions
- Real-time cache analytics and insights
- Novel approaches to cache validation
- Exploration of post-quantum security techniques

## Long-Term Vision

- AI-driven infrastructure optimization
- Autonomous platform operations
- Intent-based cache configuration
- Advanced data distribution models



# Conclusion and Key Takeaways

## Latency Reduction

Achieved substantial reductions in response times for high-traffic workloads.

## Database Load

Dramatically lowered database query volume through effective caching strategies.

## Cache Hit Rate

Sustained industry-leading cache hit ratios during peak usage periods.

## Global Latency

Improved response times for international users by leveraging geo-distributed cache clusters.

Building self-healing cache infrastructure is about creating platforms that empower developers, delight users, and enable business growth while reducing complexity and operational burden.