Product API System - Detailed AWS Cost Analysis

Executive Summary

The Product API System is a serverless architecture built on AWS using Lambda, API Gateway, and DynamoDB. This analysis provides comprehensive cost estimates for different usage scenarios, from development to enterprise scale.

Key Findings: - **Development/Testing**: $0.00/month (within free tier) - **Low Production**: $0.05 - $0.15/month - **Medium Production**: $2.50 - $5.00/month
- **High Production**: $25.00 - $50.00/month - **Enterprise Scale**: $250.00 - $500.00/month

Architecture Overview

Components

1. **Amazon API Gateway**: REST API with 2 endpoints
   - `GET /products` - List all products
   - `GET /products/{id}` - Get specific product
2. **AWS Lambda**: 2 functions with 256MB memory
   - `getProducts` - Handles product listing
   - `getProductById` - Handles single product retrieval
3. **Amazon DynamoDB**: Single table with on-demand billing
   - Table: `Products`
   - Partition Key: `product_id`

Detailed Cost Breakdown by Service

1. AWS Lambda

Pricing Structure: - Requests: $0.0000002 per request - Compute (Tier 1): $0.0000166667 per GB-second (first 6B GB-seconds/month) - Compute (Tier 2): $0.0000150000 per GB-second (next 9B GB-seconds/month) - Compute (Tier 3): $0.0000133334 per GB-second (over 15B GB-seconds/month)

Free Tier: First 12 months: 1M requests/month + 400,000 GB-seconds/month

Usage Scenarios:

Scenario	Requests/Month	Duration	GB-Seconds	Request Cost	Compute Cost	Total Cost
Development	1,000	200ms	51.2	$0.0002	$0.0009	**$0.00** (Free Tier)
Low Production	10,000	200ms	512	$0.002	$0.009	**$0.01**
Medium Production	100,000	200ms	5,120	$0.02	$0.085	**$0.11**
High Production	1,000,000	200ms	51,200	$0.20	$0.85	**$1.05**
Enterprise	10,000,000	200ms	512,000	$2.00	$8.53	**$10.53**

2. Amazon API Gateway

Pricing Structure: - Tier 1: $0.0000035 per request (first 333M requests/month) - Tier 2: $0.0000028 per request (next 667M requests/month) - Tier 3: $0.0000238 per request (next 19B requests/month) - Tier 4: $0.0000151 per request (over 20B requests/month)

Free Tier: First 12 months: 1M API calls/month

Usage Scenarios:

Scenario	Requests/Month	Tier	Unit Price	Total Cost
Development	1,000	Free Tier	$0.00	**$0.00**
Low Production	10,000	Free Tier	$0.00	**$0.00**
Medium Production	100,000	Free Tier	$0.00	**$0.00**
High Production	1,000,000	Free Tier	$0.00	**$0.00**
Enterprise	10,000,000	Tier 1	$0.0000035	**$35.00**

3. Amazon DynamoDB

Pricing Structure: - Read Request Units: $0.000000125 per RRU ($0.125 per million) - Write Request Units:

$0.000000625 per WRU ($0.625 per million) - Storage: $0.00 for first 25GB, $0.25/GB-month beyond

Free Tier: Always free: 25 GB storage + 25 WRU + 25 RRU per month

Usage Scenarios:

Scenario	Read Ops	Write Ops	Storage (GB)	RRU Cost	WRU Cost	Storage Cost	Total Cost
Development	5,000	500	1	$0.0006	$0.0003	$0.00	**$0.00** (Free Tier)
Low Production	50,000	5,000	5	$0.006	$0.003	$0.00	**$0.01**
Medium Production	500,000	50,000	10	$0.063	$0.031	$0.00	**$0.09**
High Production	5,000,000	500,000	50	$0.625	$0.313	$6.25	**$7.19**
Enterprise	50,000,000	5,000,000	200	$6.25	$3.13	$43.75	**$53.13**

Total Cost Summary by Scenario

Scenario	Lambda	API Gateway	DynamoDB	Total Monthly Cost
Development	$0.00	$0.00	$0.00	**$0.00**
Low Production	$0.01	$0.00	$0.01	**$0.02**
Medium Production	$0.11	$0.00	$0.09	**$0.20**
High Production	$1.05	$0.00	$7.19	**$8.24**
Enterprise	$10.53	$35.00	$53.13	**$98.66**

Cost Optimization Strategies

Immediate Actions

1. **Start with Free Tier**: Leverage 12-month free tier for API Gateway and Lambda
2. **Monitor Usage**: Use AWS Cost Explorer and CloudWatch for tracking
3. **Optimize Queries**: Implement efficient DynamoDB access patterns
4. **Right-size Functions**: Monitor Lambda memory usage and adjust accordingly

Medium-term Optimizations

1. **API Caching**: Implement API Gateway caching for frequently accessed data
2. **Lambda Optimization**:
   - Optimize function code for faster execution
   - Consider ARM-based Graviton2 processors for 20% cost savings
3. **DynamoDB Optimization**:
   - Use single-table design patterns
   - Implement pagination for large result sets
   - Consider Global Secondary Indexes (GSI) carefully

Long-term Strategies

1. **Reserved Capacity**: For predictable workloads, consider DynamoDB reserved capacity
2. **Provisioned Concurrency**: For consistent performance requirements
3. **Multi-region**: Plan for disaster recovery and global distribution
4. **Monitoring & Alerting**: Implement cost anomaly detection

Risk Factors & Considerations

Cost Risks

- **Traffic Spikes**: Sudden increases in API calls can significantly impact costs
- **Data Growth**: DynamoDB storage costs scale linearly with data size
- **Inefficient Queries**: Poor query patterns can increase RRU/WRU consumption

Mitigation Strategies

- **Rate Limiting**: Implement API throttling to prevent abuse
- **Circuit Breakers**: Add fault tolerance to prevent cascading failures
- **Cost Alerts**: Set up billing alerts for unexpected cost increases
- **Load Testing**: Regular performance testing to understand scaling behavior

Recommendations by Business Stage

Startup/MVP (0-1K users)

- **Budget**: $0-5/month
- **Strategy**: Maximize free tier usage
- **Focus**: Functionality over optimization

Growth Stage (1K-10K users)

- **Budget**: $5-50/month
- **Strategy**: Monitor and optimize based on usage patterns
- **Focus**: Performance optimization and cost monitoring

Scale Stage (10K-100K users)

- **Budget**: $50-500/month
- **Strategy**: Consider reserved capacity and advanced optimizations
- **Focus**: Reliability, performance, and cost efficiency

Enterprise (100K+ users)

- **Budget**: $500+/month
- **Strategy**: Full optimization suite with dedicated support
- **Focus**: Multi-region, disaster recovery, and enterprise features

Conclusion

The Product API System offers excellent cost efficiency for serverless workloads, with the ability to start at $0/month and scale cost-effectively. The key to cost optimization is:

1. **Start Simple**: Use on-demand pricing initially
2. **Monitor Continuously**: Track usage patterns and costs
3. **Optimize Iteratively**: Apply optimizations based on actual usage data
4. **Plan for Scale**: Consider reserved capacity for predictable workloads

Regular review and adjustment of the architecture based on actual usage patterns will ensure optimal cost efficiency as the system grows.