Contents

RADIANT Cost Optimization Guide

Overview

This guide provides strategies for optimizing AWS costs for the RADIANT platform while maintaining performance and reliability.

Current Architecture Costs

Estimated Monthly Costs by Tier

Tier	Infrastructure	Est. Monthly Cost
SEED (Dev)	Minimal	$50-150
STARTUP	Small production	$200-400
GROWTH	Self-hosted models	$1,000-2,500
SCALE	Multi-region	$4,000-8,000
ENTERPRISE	Global, full HA	$15,000-35,000

Cost Breakdown by Service

Service	% of Total	Optimization Potential
Aurora	30-40%	High
Lambda	15-25%	Medium
API Gateway	5-10%	Low
S3	5-10%	Medium
CloudFront	5-10%	Low
ElastiCache	10-15%	Medium
Other	10-15%	Varies

Optimization Strategies

1. Database Optimization

Aurora Serverless v2

```
// Use Serverless v2 for variable workloads
const cluster = new rds.DatabaseCluster(this, 'Database', {
  serverlessV2MinCapacity: 0.5,   // Scale to near-zero
  serverlessV2MaxCapacity: 16,    // Scale up when needed
});
```

Savings: 40-60% vs. provisioned instances for variable workloads

Reserved Instances (Steady Workloads)

```
# Purchase reserved capacity for predictable workloads
aws rds purchase-reserved-db-instances-offering \
  --reserved-db-instances-offering-id xxx \
  --db-instance-count 1
```

Savings: 30-60% for 1-3 year terms

Read Replicas Strategy

```
// Use read replicas only when needed
// Scale readers with traffic
readers: [
  rds.ClusterInstance.serverlessV2('reader', {
    scaleWithWriter: true,  // Auto-scale with primary
  }),
],
```

2. Lambda Optimization

Right-Size Memory

```
// Test different memory sizes to find optimal cost/performance
const memoryOptions = [256, 512, 1024, 2048];

// Use AWS Lambda Power Tuning tool
// https://github.com/alexcasalboni/aws-lambda-power-tuning
```

Function Type	Recommended Memory	Reason
Simple CRUD	256-512 MB	Light compute
API Router	512-1024 MB	Balanced
AI Processing	1024-2048 MB	Heavy compute

Provisioned Concurrency (Strategic)

```
// Only use for latency-critical functions
new lambda.Alias(this, 'LiveAlias', {
  aliasName: 'live',
  version: fn.currentVersion,
  provisionedConcurrentExecutions: 5,  // Keep 5 warm
});
```

Cost: ~$0.015/hour per provisioned instance **Use when:** P99 latency requirements $<$ 200ms

ARM64 (Graviton2)

```
// 20% cheaper, often faster
const fn = new lambda.Function(this, 'Function', {
  architecture: lambda.Architecture.ARM_64,
  runtime: lambda.Runtime.NODEJS_20_X,
});
```

Savings: 20% on compute costs

3. S3 Optimization

Intelligent Tiering

```
const bucket = new s3.Bucket(this, 'Storage', {
  intelligentTieringConfigurations: [{
    name: 'auto-tier',
    archiveAccessTierTime: cdk.Duration.days(90),
    deepArchiveAccessTierTime: cdk.Duration.days(180),
  }],
});
```

Savings: Up to 95% for infrequently accessed data

Lifecycle Rules

```
const bucket = new s3.Bucket(this, 'Storage', {
  lifecycleRules: [
    // Move old versions to cheaper storage
    {
      noncurrentVersionTransitions: [
        {
          storageClass: s3.StorageClass.INFREQUENT_ACCESS,
          transitionAfter: cdk.Duration.days(30),
```

```
        },
        {
          storageClass: s3.StorageClass.GLACIER,
          transitionAfter: cdk.Duration.days(90),
        },
      ],
    },
    // Delete old logs
    {
      prefix: 'logs/',
      expiration: cdk.Duration.days(90),
    },
  ],
});
```

4. API Gateway Optimization

HTTP API vs REST API

```
// HTTP API is 70% cheaper than REST API
// Use when you don't need REST API features

// HTTP API: $1.00/million requests
// REST API: $3.50/million requests
```

Feature	REST API	HTTP API
Cost	$3.50/M	$1.00/M
Lambda integration	Yes	Yes
Request validation	Yes	No
API keys/usage plans	Yes	No
Caching	Yes	No

Caching

```
// Enable caching for GET endpoints
const method = resource.addMethod('GET', integration, {
  cacheKeyParameters: ['method.request.querystring.id'],
});

// Cache stage setting
stage.cacheClusterEnabled = true;
stage.cacheClusterSize = '0.5';  // 0.5 GB minimum
```

Note: Cache costs $0.02/hour (0.5 GB). Calculate break-even point.

5. CloudWatch Optimization

Log Retention

```typescript
// Don't keep logs forever
new logs.LogGroup(this, 'LogGroup', {
  retention: logs.RetentionDays.ONE_MONTH,  // Adjust per environment
});
```

Environment	Retention	Reason
Development	7 days	Quick debugging
Staging	14 days	Testing cycles
Production	90 days	Compliance needs

Metric Filters vs. Logs Insights

```typescript
// Use metric filters for known patterns
// Cheaper than running Logs Insights queries repeatedly

new logs.MetricFilter(this, 'ErrorMetric', {
  logGroup,
  metricNamespace: 'Radiant',
  metricName: 'Errors',
  filterPattern: logs.FilterPattern.literal('ERROR'),
});
```

6. ElastiCache Optimization

Reserved Nodes

```bash
# Purchase reserved nodes for production
aws elasticache purchase-reserved-cache-nodes-offering \
  --reserved-cache-nodes-offering-id xxx
```

Savings: 30-55% for 1-3 year terms

Right-Size Nodes

Use Case	Recommended	Memory
Development	cache.t3.micro	0.5 GB
Small Prod	cache.t3.small	1.4 GB
Medium Prod	cache.r6g.large	13 GB
Large Prod	cache.r6g.xlarge	26 GB

7. Data Transfer Optimization

Use VPC Endpoints

```typescript
// Avoid NAT Gateway costs for AWS services
vpc.addInterfaceEndpoint('S3Endpoint', {
  service: ec2.InterfaceVpcEndpointAwsService.S3,
});
```

```
vpc.addInterfaceEndpoint('SecretsManagerEndpoint', {
  service: ec2.InterfaceVpcEndpointAwsService.SECRETS_MANAGER,
});
```

Savings: $0.045/GB saved vs. NAT Gateway

CloudFront for S3

```
// Serve S3 content through CloudFront
// Cheaper data transfer + better performance
const distribution = new cloudfront.Distribution(this, 'CDN', {
  defaultBehavior: {
    origin: new origins.S3Origin(bucket),
  },
});
```

Cost Monitoring

AWS Cost Explorer

```
# Get cost breakdown by service
aws ce get-cost-and-usage \
  --time-period Start=2024-12-01,End=2024-12-31 \
  --granularity MONTHLY \
  --metrics BlendedCost \
  --group-by Type=DIMENSION,Key=SERVICE
```

CloudWatch Billing Alerts

```
// Alert before surprise bills
new cloudwatch.Alarm(this, 'BillingAlarm', {
  metric: new cloudwatch.Metric({
    namespace: 'AWS/Billing',
    metricName: 'EstimatedCharges',
    dimensionsMap: { Currency: 'USD' },
    statistic: 'Maximum',
    period: cdk.Duration.hours(6),
  }),
  threshold: 1000,  // $1000 threshold
  evaluationPeriods: 1,
});
```

Cost Allocation Tags

```
// Tag all resources for cost tracking
cdk.Tags.of(this).add('Project', 'radiant');
cdk.Tags.of(this).add('Environment', environment);
cdk.Tags.of(this).add('CostCenter', 'platform');
```

Environment-Specific Recommendations

Development

- Use Aurora Serverless v2 (scales to zero)
- Minimal Lambda memory
- No provisioned concurrency
- Short log retention
- Single-AZ deployments

Target: < $100/month

Staging

- Aurora Serverless v2
- Moderate Lambda memory
- No provisioned concurrency
- 14-day log retention
- Single-AZ acceptable

Target: < $300/month

Production

- Aurora Serverless v2 or Reserved (if predictable)
- Right-sized Lambda memory
- Provisioned concurrency for critical paths
- 90-day log retention
- Multi-AZ required

Target: Optimize for reliability, then cost

Monthly Cost Review Checklist

- ☐ Review AWS Cost Explorer for anomalies
- ☐ Check for unused resources (idle RDS, orphan EBS)
- ☐ Review Lambda right-sizing opportunities
- ☐ Check S3 storage class distribution
- ☐ Review data transfer costs
- ☐ Validate reserved capacity utilization
- ☐ Update cost allocation tags
- ☐ Project next month's costs

Tools

- AWS Cost Explorer
- AWS Trusted Advisor
- AWS Compute Optimizer
- Lambda Power Tuning
- Infracost - Cost estimation for IaC