Contents

RADIANT Performance Optimization Guide

Version: 5.42.0

This document outlines performance optimizations implemented and recommended for the RADIANT platform.

---

1. Lambda Cold Start Optimization

Current Configuration

Lambda	Memory	Timeout	Provisioned Concurrency
Admin API	1024 MB	30s	0 (on-demand)
Think Tank API	2048 MB	60s	0 (on-demand)
Agent Worker	2048 MB	300s	0 (configurable)
Transparency Worker	512 MB	30s	0

Recommendations

1.1 Bundle Optimization

```
// Use esbuild for smaller bundles
// Current: ~5MB bundled
// Target: <2MB bundled

// In CDK stack:
bundling: {
  minify: true,
  sourceMap: false,
  treeshaking: true,
  externalModules: ['@aws-sdk/*'], // Use Lambda's built-in SDK
}
```

1.2 Lazy Loading

```
// Defer heavy imports until needed
let bedrockClient: BedrockRuntimeClient | null = null;

function getBedrockClient() {
  if (!bedrockClient) {
    bedrockClient = new BedrockRuntimeClient({});
  }
  return bedrockClient;
}
```

1.3 Connection Reuse

```
// Reuse HTTP connections
const httpAgent = new https.Agent({
  keepAlive: true,
  maxSockets: 50,
});
```

---

2. Database Query Optimization

2.1 Query Patterns

Use Indexes Effectively:

```
-- Ensure indexes exist for common queries
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_ai_reports_tenant_updated
ON ai_reports(tenant_id, updated_at DESC);

CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_ai_report_insights_tenant_created
ON ai_report_insights(tenant_id, created_at DESC);
```

Batch Operations:

```
// Instead of N individual inserts
for (const item of items) {
  await executeStatement('INSERT INTO...', [item]);
}

// Use batch insert
const values = items.map((_, i) => `($${i*3+1}, $${i*3+2}, $${i*3+3})`).join(',');
await executeStatement(`INSERT INTO table (a, b, c) VALUES ${values}`, flatParams);
```

2.2 Connection Pooling

Aurora Data API handles connection pooling automatically. Ensure: - `maxConnections` is appropriate for tier - Idle connections are released

---

3. Caching Strategy

3.1 In-Memory Caching (Lambda)

```
// Cache expensive computations during Lambda lifetime
const cache = new Map<string, { data: unknown; expiry: number }>();

function getCached<T>(key: string, ttlMs: number, compute: () => T): T {
  const now = Date.now();
  const cached = cache.get(key);

  if (cached && cached.expiry > now) {
    return cached.data as T;
  }

  const data = compute();
  cache.set(key, { data, expiry: now + ttlMs });
  return data;
}
```

3.2 Redis/ElastiCache (Production)

For high-traffic endpoints: - Model configurations: 5 min TTL - Tenant settings: 1 min TTL - User sessions: 15 min TTL

3.3 API Gateway Caching

```
// CDK configuration
const api = new apigateway.RestApi(this, 'Api', {
  deployOptions: {
    cachingEnabled: true,
    cacheClusterEnabled: true,
    cacheClusterSize: '0.5',
    cacheTtl: cdk.Duration.minutes(1),
  },
});

// Per-method caching
method.addMethodResponse({
  statusCode: '200',
  responseParameters: {
    'method.response.header.Cache-Control': true,
  },
});
```

---

4. Response Optimization

4.1 Compression

```
// Enable gzip for large responses
if (body.length > 1024) {
  const compressed = zlib.gzipSync(body);
  return {
    statusCode: 200,
    headers: {
      'Content-Encoding': 'gzip',
      'Content-Type': 'application/json',
    },
    body: compressed.toString('base64'),
    isBase64Encoded: true,
  };
}
```

4.2 Pagination

```
// Always paginate large result sets
const DEFAULT_PAGE_SIZE = 50;
const MAX_PAGE_SIZE = 200;

function paginate<T>(items: T[], page: number, pageSize: number): PaginatedResult<T> {
  const size = Math.min(pageSize || DEFAULT_PAGE_SIZE, MAX_PAGE_SIZE);
  const offset = (page - 1) * size;
  return {
    items: items.slice(offset, offset + size),
    total: items.length,
    page,
    pageSize: size,
    totalPages: Math.ceil(items.length / size),
  };
}
```

---

5. AI Model Call Optimization

5.1 Streaming Responses

```
// Use streaming for long generations
const stream = await bedrockClient.send(new InvokeModelWithResponseStreamCommand({
  modelId,
  body: JSON.stringify(request),
}));

for await (const event of stream.body) {
  if (event.chunk) {
    yield JSON.parse(new TextDecoder().decode(event.chunk.bytes));
  }
}
```

5.2 Request Batching

```
// Batch similar requests
const batchWindow = 50; // ms
const pendingRequests: Map<string, Promise<Response>> = new Map();

async function batchedInvoke(prompt: string): Promise<Response> {
  const key = hashPrompt(prompt);

  if (pendingRequests.has(key)) {
    return pendingRequests.get(key)!;
  }

  const promise = invoke(prompt);
  pendingRequests.set(key, promise);

  setTimeout(() => pendingRequests.delete(key), batchWindow);

  return promise;
}
```

5.3 Prompt Caching (Claude)

```
// Use prompt caching for repeated system prompts
const request = {
  anthropic_version: 'bedrock-2023-05-31',
  system: [
    {
      type: 'text',
      text: systemPrompt,
      cache_control: { type: 'ephemeral' }, // Enable caching
    },
  ],
  messages,
};
```

---

6. Frontend Performance

6.1 Code Splitting

```
// Dynamic imports for large components
const AIReportsPage = dynamic(() => import('./ai-reports'), {
  loading: () => <Skeleton />,
  ssr: false,
});
```

6.2 Data Fetching

```
// Use SWR for caching and revalidation
const { data, error, isLoading } = useSWR(
  '/api/admin/ai-reports',
  fetcher,
  {
    revalidateOnFocus: false,
    dedupingInterval: 30000,
  }
);
```

6.3 Image Optimization

```
// Use Next.js Image component
import Image from 'next/image';

<Image
  src={logoUrl}
  width={200}
  height={50}
  priority={false}
  loading="lazy"
/>
```

---

7. Monitoring & Alerts

7.1 Key Metrics

Metric	Warning	Critical
P95 Latency	> 1s	> 3s
Error Rate	> 1%	> 5%
Cold Start Rate	> 10%	> 25%
Cache Hit Rate	< 80%	< 60%

7.2 CloudWatch Alarms

```
new cloudwatch.Alarm(this, 'HighLatencyAlarm', {
  metric: api.metricLatency({ statistic: 'p95' }),
  threshold: 1000,
  evaluationPeriods: 3,
  comparisonOperator: cloudwatch.ComparisonOperator.GREATER_THAN_THRESHOLD,
});
```

---

8. Cost Optimization

8.1 Right-Sizing

- Start with minimum viable memory
- Use Lambda Power Tuning to find optimal
- Monitor actual memory usage

8.2 Reserved Concurrency

```
// For predictable workloads
const fn = new lambda.Function(this, 'Function', {
  // ...
  reservedConcurrentExecutions: 100, // Limit max concurrency
});
```

8.3 Spot Instances (ECS/Fargate)

For self-hosted models:

```
fargateService.taskDefinition.addContainer('model', {
  // ...
  capacityProviderStrategies: [
    { capacityProvider: 'FARGATE_SPOT', weight: 2 },
    { capacityProvider: 'FARGATE', weight: 1 },
  ],
});
```

---

9. Quick Wins Checklist

- ☐ Enable API Gateway caching for read endpoints
- ☐ Add database indexes for frequent queries
- ☐ Implement response compression for large payloads
- ☐ Use streaming for AI model responses
- ☐ Enable HTTP keep-alive for external calls
- ☐ Lazy-load heavy SDK clients
- ☐ Paginate all list endpoints
- ☐ Add CloudWatch alarms for latency
- ☐ Review Lambda memory settings monthly
- ☐ Enable prompt caching for Claude models