

Contents

SECTION 12: METRICS & ANALYTICS (v2.5.0)	1
	1
12.1 Analytics Database Schema	1
12.2 Metrics Collector Service	2
	4

SECTION 12: METRICS & ANALYTICS (v2.5.0)

12.1 Analytics Database Schema

-- migrations/022_metrics_analytics.sql

```
CREATE TABLE usage_metrics (  
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),  
    tenant_id UUID NOT NULL REFERENCES tenants(id),  
    user_id UUID REFERENCES users(id),  
    metric_type VARCHAR(50) NOT NULL,  
    metric_name VARCHAR(100) NOT NULL,  
    metric_value DECIMAL(20, 6) NOT NULL,  
    dimensions JSONB DEFAULT '{}',  
    recorded_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE TABLE aggregated_metrics (  
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),  
    tenant_id UUID NOT NULL REFERENCES tenants(id),  
    period_start TIMESTAMPTZ NOT NULL,  
    period_end TIMESTAMPTZ NOT NULL,  
    period_type VARCHAR(20) NOT NULL,  
    metric_type VARCHAR(50) NOT NULL,  
    total_requests BIGINT DEFAULT 0,  
    total_tokens BIGINT DEFAULT 0,  
    total_cost DECIMAL(20, 6) DEFAULT 0,  
    avg_latency_ms DECIMAL(10, 2),  
    p95_latency_ms DECIMAL(10, 2),  
    p99_latency_ms DECIMAL(10, 2),  
    error_count INTEGER DEFAULT 0,  
    unique_users INTEGER DEFAULT 0,  
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE INDEX idx_usage_metrics_tenant_time ON usage_metrics(tenant_id, recorded_at);
```

```

CREATE INDEX idx_usage_metrics_type ON usage_metrics(metric_type);
CREATE INDEX idx_aggregated_period ON aggregated_metrics(tenant_id, period_start, period_type)

ALTER TABLE usage_metrics ENABLE ROW LEVEL SECURITY;
ALTER TABLE aggregated_metrics ENABLE ROW LEVEL SECURITY;

CREATE POLICY usage_metrics_isolation ON usage_metrics USING (tenant_id = current_setting('app
CREATE POLICY aggregated_metrics_isolation ON aggregated_metrics USING (tenant_id = current_se

```

12.2 Metrics Collector Service

// packages/core/src/services/metrics-collector.ts

```

import { Pool } from 'pg';
import { CloudWatchClient, PutMetricDataCommand } from '@aws-sdk/client-cloudwatch';

interface MetricEvent {
  tenantId: string;
  userId?: string;
  metricType: 'api_request' | 'token_usage' | 'model_inference' | 'billing';
  metricName: string;
  value: number;
  dimensions?: Record<string, string>;
}

export class MetricsCollector {
  private pool: Pool;
  private cloudwatch: CloudWatchClient;
  private buffer: MetricEvent[] = [];
  private flushInterval: NodeJS.Timeout;

  constructor(pool: Pool) {
    this.pool = pool;
    this.cloudwatch = new CloudWatchClient({});
    this.flushInterval = setInterval(() => this.flush(), 10000);
  }

  record(event: MetricEvent): void {
    this.buffer.push(event);

    if (this.buffer.length >= 100) {
      this.flush();
    }
  }

  async flush(): Promise<void> {
    if (this.buffer.length === 0) return;

```

```

const events = [...this.buffer];
this.buffer = [];

// Batch insert to PostgreSQL
const values = events.map((e, i) =>
  `(${i*6+1}, ${i*6+2}, ${i*6+3}, ${i*6+4}, ${i*6+5}, ${i*6+6})`
).join(', ');

const params = events.flatMap(e => [
  e.tenantId, e.userId, e.metricType, e.metricName, e.value, JSON.stringify(e.dimensions)
]);

await this.pool.query(`
  INSERT INTO usage_metrics (tenant_id, user_id, metric_type, metric_name, metric_value)
  VALUES ${values}
`, params);

// Send to CloudWatch
await this.sendToCloudWatch(events);
}

private async sendToCloudWatch(events: MetricEvent[]): Promise<void> {
  const metricData = events.map(e => ({
    MetricName: e.metricName,
    Dimensions: [
      { Name: 'TenantId', Value: e.tenantId },
      { Name: 'MetricType', Value: e.metricType }
    ],
    Value: e.value,
    Timestamp: new Date(),
    Unit: this.getUnit(e.metricName)
  }));

  // CloudWatch accepts max 20 metrics per call
  for (let i = 0; i < metricData.length; i += 20) {
    await this.cloudwatch.send(new PutMetricDataCommand({
      Namespace: 'RADIANT',
      MetricData: metricData.slice(i, i + 20)
    }));
  }
}

private getUnit(metricName: string): string {
  if (metricName.includes('latency')) return 'Milliseconds';
  if (metricName.includes('cost')) return 'None';
  if (metricName.includes('tokens')) return 'Count';
  return 'Count';
}

```

```

async getAggregatedMetrics(
  tenantId: string,
  periodType: 'hourly' | 'daily' | 'weekly' | 'monthly',
  startDate: Date,
  endDate: Date
) {
  const result = await this.pool.query(`
    SELECT * FROM aggregated_metrics
    WHERE tenant_id = $1
    AND period_type = $2
    AND period_start >= $3
    AND period_end <= $4
    ORDER BY period_start
  `, [tenantId, periodType, startDate, endDate]);

  return result.rows;
}
}

```