Contents

SECTION 44: STORAGE BILLING SYSTEM (v4.14.0)

Version: 4.14.0 | Tiered storage billing for S3 and database usage

---

44.1 STORAGE BILLING OVERVIEW

Tier	S3 (/*GB*/*mo*)|*DB*(/GB/mo)	Backup ($/GB/mo)	Included	
FREE	$0	$0	N/A	1GB S3, 500MB DB
INDIVIDUAL	$0.10	$0.15	Included	10GB S3, 2GB DB
FAMILY	$0.08	$0.12	Included	25GB S3, 5GB DB
TEAM	$0.06	$0.10	Included	100GB S3, 20GB DB
BUSINESS	$0.04	$0.08	Included	500GB S3, 100GB DB
ENTERPRISE	Custom	Custom	Custom	Unlimited

44.2 DATABASE SCHEMA

packages/infrastructure/migrations/044_storage_billing.sql

```sql
-- ============================================================================
-- RADIANT v4.14.0 - Storage Billing Schema
-- ============================================================================

-- Storage Usage Tracking
CREATE TABLE storage_usage (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID REFERENCES users(id),
    app_id UUID REFERENCES applications(id),
    storage_type VARCHAR(20) NOT NULL CHECK (storage_type IN ('s3', 'database', 'backup', 'embe

    bytes_used BIGINT NOT NULL DEFAULT 0,
    bytes_quota BIGINT,

    period_start TIMESTAMPTZ NOT NULL,
    period_end TIMESTAMPTZ NOT NULL,

    price_per_gb_cents INTEGER NOT NULL,
    total_cost_cents INTEGER NOT NULL DEFAULT 0,

    is_over_quota BOOLEAN DEFAULT FALSE,
    quota_warning_sent BOOLEAN DEFAULT FALSE,
    quota_exceeded_sent BOOLEAN DEFAULT FALSE,

    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_storage_usage_tenant ON storage_usage(tenant_id);
CREATE INDEX idx_storage_usage_type ON storage_usage(storage_type);
CREATE INDEX idx_storage_usage_period ON storage_usage(period_start, period_end);

-- Storage Pricing Configuration
CREATE TABLE storage_pricing (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tier_id VARCHAR(50) NOT NULL REFERENCES subscription_tiers(id),
    storage_type VARCHAR(20) NOT NULL,

    price_per_gb_cents INTEGER NOT NULL,
    included_gb DECIMAL(10,2) NOT NULL DEFAULT 0,
    max_gb DECIMAL(10,2),
    overage_price_per_gb_cents INTEGER,

    is_active BOOLEAN DEFAULT TRUE,
    effective_from TIMESTAMPTZ DEFAULT NOW(),
    effective_until TIMESTAMPTZ,

    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW(),

    UNIQUE(tier_id, storage_type, effective_from)
);

-- Storage Events
CREATE TABLE storage_events (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID REFERENCES users(id),

    event_type VARCHAR(30) NOT NULL CHECK (event_type IN (
        'upload', 'delete', 'archive', 'restore', 'expire', 'quota_warning', 'quota_exceeded'
    )),

    storage_type VARCHAR(20) NOT NULL,
    bytes_delta BIGINT NOT NULL,

    resource_id VARCHAR(255),
    resource_type VARCHAR(50),
    resource_path TEXT,

    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_storage_events_tenant ON storage_events(tenant_id);
CREATE INDEX idx_storage_events_type ON storage_events(event_type);

-- Enable RLS
ALTER TABLE storage_usage ENABLE ROW LEVEL SECURITY;
ALTER TABLE storage_events ENABLE ROW LEVEL SECURITY;

CREATE POLICY storage_usage_tenant ON storage_usage
    USING (tenant_id = current_setting('app.current_tenant_id')::UUID);
CREATE POLICY storage_events_tenant ON storage_events
    USING (tenant_id = current_setting('app.current_tenant_id')::UUID);
```

44.3 STORAGE SERVICE

packages/functions/src/services/storage-billing.ts

```
/**
 * Storage Billing Service
 * @version 4.14.0
 */

import { Pool } from 'pg';
import { S3Client, ListObjectsV2Command } from '@aws-sdk/client-s3';

export class StorageBillingService {
  constructor(private pool: Pool, private s3Client: S3Client) {}

  async getStorageUsage(tenantId: string): Promise<{
    storageType: string;
    bytesUsed: number;
    bytesQuota: number | null;
    pricePerGbCents: number;
    includedGb: number;
    totalCostCents: number;
  }[]> {
    const result = await this.pool.query(`
      SELECT
        su.storage_type,
        su.bytes_used,
        su.bytes_quota,
        sp.price_per_gb_cents,
        sp.included_gb,
        CASE
          WHEN su.bytes_used <= sp.included_gb * 1073741824 THEN 0
          ELSE CEIL((su.bytes_used - sp.included_gb * 1073741824) / 1073741824.0) * sp.price_pe
        END as total_cost_cents
      FROM storage_usage su
      JOIN subscriptions s ON s.tenant_id = su.tenant_id AND s.status = 'active'
      JOIN storage_pricing sp ON sp.tier_id = s.tier_id AND sp.storage_type = su.storage_type
      WHERE su.tenant_id = $1 AND sp.is_active = TRUE AND su.period_end > NOW()
    `, [tenantId]);

    return result.rows;
  }

  async recordStorageEvent(
    tenantId: string,
    userId: string | null,
    eventType: string,
    storageType: string,
    bytesDelta: number,
    resourceId?: string
  ): Promise<void> {
    await this.pool.query(`
      INSERT INTO storage_events (tenant_id, user_id, event_type, storage_type, bytes_delta, re
      VALUES ($1, $2, $3, $4, $5, $6)
    `, [tenantId, userId, eventType, storageType, bytesDelta, resourceId]);

    await this.updateStorageUsage(tenantId, storageType, bytesDelta);
  }

  private async updateStorageUsage(tenantId: string, storageType: string, bytesDelta: number):
    await this.pool.query(`
      INSERT INTO storage_usage (tenant_id, storage_type, bytes_used, period_start, period_end
      SELECT $1, $2, GREATEST(0, $3), date_trunc('month', NOW()), date_trunc('month', NOW()) +
        COALESCE((SELECT price_per_gb_cents FROM storage_pricing sp
          JOIN subscriptions s ON s.tier_id = sp.tier_id AND s.tenant_id = $1
          WHERE sp.storage_type = $2 AND sp.is_active = TRUE LIMIT 1), 10)
        ON CONFLICT (tenant_id, storage_type, period_start, period_end)
        DO UPDATE SET bytes_used = GREATEST(0, storage_usage.bytes_used + $3), updated_at = NOW(
      `, [tenantId, storageType, bytesDelta]);
    }

    async calculateS3Usage(tenantId: string): Promise<number> {
      let totalBytes = 0;
      let continuationToken: string | undefined;

      do {
        const response = await this.s3Client.send(new ListObjectsV2Command({
          Bucket: process.env.S3_BUCKET_NAME,
          Prefix: `tenants/${tenantId}/`,
          ContinuationToken: continuationToken,
        }));

        if (response.Contents) {
          totalBytes += response.Contents.reduce((sum, obj) => sum + (obj.Size || 0), 0);
        }
        continuationToken = response.NextContinuationToken;
      } while (continuationToken);

      return totalBytes;
    }
  }
```