Contents

SECTION 46: DUAL-ADMIN MIGRATION APPROVAL (v4.16.0)

Version: 4.16.0 | Two-person approval for production database migrations

---

46.1 DUAL-ADMIN APPROVAL PRINCIPLES

- **Production migrations require 2 approvals** (configurable)
- **Requestor cannot self-approve**
- **Complete audit trail** of all decisions
- **Configurable policies** per tenant/environment

46.2 DATABASE SCHEMA

packages/infrastructure/migrations/046_dual_admin_approval.sql

```
-- ============================================================================
-- RADIANT v4.16.0 - Dual-Admin Migration Approval
-- ============================================================================
```

```sql
-- Migration Approval Requests
CREATE TABLE migration_approval_requests (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),

    migration_name VARCHAR(255) NOT NULL,
    migration_version VARCHAR(50) NOT NULL,
    migration_checksum VARCHAR(64) NOT NULL,
    migration_sql TEXT NOT NULL,

    environment VARCHAR(20) NOT NULL CHECK (environment IN ('development', 'staging', 'producti

    requested_by UUID NOT NULL REFERENCES administrators(id),
    requested_at TIMESTAMPTZ DEFAULT NOW(),
    request_reason TEXT,

    status VARCHAR(20) NOT NULL DEFAULT 'pending' CHECK (status IN (
        'pending', 'approved', 'rejected', 'executed', 'failed', 'cancelled'
    )),

    approvals_required INTEGER NOT NULL DEFAULT 2,
    approvals_received INTEGER NOT NULL DEFAULT 0,

    executed_at TIMESTAMPTZ,
    executed_by UUID REFERENCES administrators(id),
    execution_time_ms INTEGER,
    execution_error TEXT,
    rollback_sql TEXT,

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

CREATE INDEX idx_migration_approval_tenant ON migration_approval_requests(tenant_id);
CREATE INDEX idx_migration_approval_status ON migration_approval_requests(status);
CREATE INDEX idx_migration_approval_env ON migration_approval_requests(environment);

-- Individual Approvals
CREATE TABLE migration_approvals (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    request_id UUID NOT NULL REFERENCES migration_approval_requests(id) ON DELETE CASCADE,

    admin_id UUID NOT NULL REFERENCES administrators(id),
    decision VARCHAR(20) NOT NULL CHECK (decision IN ('approved', 'rejected')),
    reason TEXT,
    reviewed_at TIMESTAMPTZ DEFAULT NOW(),

    UNIQUE(request_id, admin_id)
);

CREATE INDEX idx_migration_approvals_request ON migration_approvals(request_id);

-- Approval Policies
CREATE TABLE migration_approval_policies (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    environment VARCHAR(20) NOT NULL,

    approvals_required INTEGER NOT NULL DEFAULT 2,
    self_approval_allowed BOOLEAN DEFAULT FALSE,
    auto_approve_development BOOLEAN DEFAULT TRUE,

    allowed_approvers UUID[] DEFAULT '{}',
    required_approvers UUID[],

    notification_channels JSONB DEFAULT '{}',
    escalation_after_hours INTEGER DEFAULT 24,

    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW(),

    UNIQUE(tenant_id, environment)
);

-- Trigger: Update approval count
CREATE OR REPLACE FUNCTION update_approval_count()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.decision = 'approved' THEN
        UPDATE migration_approval_requests
        SET approvals_received = approvals_received + 1,
            status = CASE WHEN approvals_received + 1 >= approvals_required THEN 'approved' ELS
            updated_at = NOW()
        WHERE id = NEW.request_id;
    ELSIF NEW.decision = 'rejected' THEN
        UPDATE migration_approval_requests SET status = 'rejected', updated_at = NOW() WHERE i
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER migration_approval_update
```

```sql
    AFTER INSERT ON migration_approvals
    FOR EACH ROW EXECUTE FUNCTION update_approval_count();

-- Enable RLS
ALTER TABLE migration_approval_requests ENABLE ROW LEVEL SECURITY;
ALTER TABLE migration_approvals ENABLE ROW LEVEL SECURITY;
ALTER TABLE migration_approval_policies ENABLE ROW LEVEL SECURITY;

CREATE POLICY mar_tenant ON migration_approval_requests
    USING (tenant_id = current_setting('app.current_tenant_id')::UUID);
CREATE POLICY map_tenant ON migration_approval_policies
    USING (tenant_id = current_setting('app.current_tenant_id')::UUID);
```

46.3 MIGRATION APPROVAL SERVICE

packages/functions/src/services/migration-approval.ts

```typescript
/**
 * Dual-Admin Migration Approval Service
 * @version 4.16.0
 */

import { Pool } from 'pg';
import crypto from 'crypto';

export class MigrationApprovalService {
  constructor(private pool: Pool) {}

  async createRequest(
    tenantId: string,
    adminId: string,
    migrationName: string,
    migrationVersion: string,
    migrationSql: string,
    environment: string,
    reason?: string
  ): Promise<{ id: string; status: string; approvalsRequired: number }> {
    // Get policy
    const policyResult = await this.pool.query(`
      SELECT * FROM migration_approval_policies
      WHERE tenant_id = $1 AND environment = $2 AND is_active = TRUE
    `, [tenantId, environment]);

    let approvalsRequired = environment === 'production' ? 2 : (environment === 'staging' ? 1
    if (policyResult.rows.length > 0) {
      const policy = policyResult.rows[0];
      approvalsRequired = policy.approvals_required;
      if (environment === 'development' && policy.auto_approve_development) approvalsRequired =
    }

    const checksum = crypto.createHash('sha256').update(migrationSql).digest('hex');

    const result = await this.pool.query(`
      INSERT INTO migration_approval_requests (
        tenant_id, migration_name, migration_version, migration_checksum,
        migration_sql, environment, requested_by, request_reason,
        approvals_required, status
      ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10)
      RETURNING id, status, approvals_required
    `, [tenantId, migrationName, migrationVersion, checksum, migrationSql, environment, adminI

    return result.rows[0];
  }

  async submitApproval(
    requestId: string,
    adminId: string,
    decision: 'approved' | 'rejected',
    reason?: string
  ): Promise<{ success: boolean; requestStatus: string; canExecute: boolean }> {
    const client = await this.pool.connect();

    try {
      await client.query('BEGIN');

      // Get request
      const requestResult = await client.query(`SELECT * FROM migration_approval_requests WHER
      if (requestResult.rows.length === 0) throw new Error('Request not found');
      const request = requestResult.rows[0];

      if (request.status !== 'pending') throw new Error(`Request is already ${request.status}`

      // Check self-approval
      const policyResult = await client.query(`
        SELECT self_approval_allowed FROM migration_approval_policies
        WHERE tenant_id = $1 AND environment = $2 AND is_active = TRUE
      `, [request.tenant_id, request.environment]);

      const selfApprovalAllowed = policyResult.rows[0]?.self_approval_allowed ?? false;
      if (request.requested_by === adminId && !selfApprovalAllowed) {
        throw new Error('Self-approval is not allowed for this environment');
      }

      // Check duplicate
      const existingApproval = await client.query(`
        SELECT id FROM migration_approvals WHERE request_id = $1 AND admin_id = $2
      `, [requestId, adminId]);
      if (existingApproval.rows.length > 0) throw new Error('You have already submitted an appr

      // Insert approval
      await client.query(`
        INSERT INTO migration_approvals (request_id, admin_id, decision, reason)
        VALUES ($1, $2, $3, $4)
      `, [requestId, adminId, decision, reason]);

      // Get updated status
      const updatedResult = await client.query(`
        SELECT status, approvals_received, approvals_required FROM migration_approval_requests
      `, [requestId]);
      const updated = updatedResult.rows[0];

      await client.query('COMMIT');

      return {
        success: true,
        requestStatus: updated.status,
        canExecute: updated.status === 'approved'
      };
    } catch (error) {
      await client.query('ROLLBACK');
      throw error;
    } finally {
      client.release();
    }
  }

  async executeMigration(requestId: string, adminId: string): Promise<{ success: boolean; erro
    const client = await this.pool.connect();

    try {
      await client.query('BEGIN');

      const requestResult = await client.query(`
        SELECT * FROM migration_approval_requests WHERE id = $1 AND status = 'approved' FOR UPD
      `, [requestId]);
      if (requestResult.rows.length === 0) throw new Error('Request not found or not approved')

      const request = requestResult.rows[0];
      const startTime = Date.now();

      try {
        await client.query(request.migration_sql);
```

```
        await client.query(`
          UPDATE migration_approval_requests
          SET status = 'executed', executed_at = NOW(), executed_by = $2, execution_time_ms = $
          WHERE id = $1
        `, [requestId, adminId, Date.now() - startTime]);

        await client.query('COMMIT');
        return { success: true };
      } catch (execError: any) {
        await client.query('ROLLBACK');
        await this.pool.query(`
          UPDATE migration_approval_requests SET status = 'failed', execution_error = $2, upda
        `, [requestId, execError.message]);
        return { success: false, error: execError.message };
      }
    } catch (error) {
      await client.query('ROLLBACK');
      throw error;
    } finally {
      client.release();
    }
  }

  async getPendingRequests(tenantId: string, adminId: string): Promise<any[]> {
    const result = await this.pool.query(`
      SELECT mar.*, NOT EXISTS (SELECT 1 FROM migration_approvals ma WHERE ma.request_id = mar
      FROM migration_approval_requests mar
      WHERE mar.tenant_id = $1 AND mar.status = 'pending' AND mar.requested_by != $2
      ORDER BY mar.created_at DESC
    `, [tenantId, adminId]);
    return result.rows;
  }
}
```

IMPLEMENTATION VERIFICATION CHECKLIST (v4.13.0 - v4.16.0)

Section 43: Billing & Credits (v4.13.0)

- ☐ Migration 043_billing_system.sql applied
- ☐ All 7 subscription tiers seeded
- ☐ credit_pools table created with RLS

☐ credit_transactions table created
☐ subscriptions table created
☐ CreditsService implemented
☐ Stripe integration working

Section 44: Storage Billing (v4.14.0)

☐ Migration 044_storage_billing.sql applied
☐ storage_usage table created
☐ storage_pricing configured per tier
☐ StorageBillingService implemented
☐ S3 usage calculation working
☐ Quota warnings sending

Section 45: Versioned Subscriptions (v4.15.0)

☐ Migration 045_versioned_subscriptions.sql applied
☐ subscription_plan_versions table created
☐ grandfathered_subscriptions table created
☐ get_effective_plan() function working
☐ GrandfatheringService implemented
☐ Migration offers working

Section 46: Dual-Admin Approval (v4.16.0)

☐ Migration 046_dual_admin_approval.sql applied
☐ migration_approval_requests table created
☐ migration_approvals table created
☐ Approval count trigger working
☐ MigrationApprovalService implemented
☐ Self-approval prevention working

RADIANT v4.17.0 - AI-Optimized for Code Generation Version: 4.17.0 | December 2024 | Prompt 32 Total Sections: 47 (0-46)

v4.17.0 AI Code Generation Enhancements:

- Complete RadiantDeployerApp.swift with @main struct
- Package.swift manifest for Swift Package Manager
- Info.plist and entitlements templates
- RADIANT_VERSION constant (replaces hardcoded strings)
- DOMAIN_PLACEHOLDER constant for configuration
- Sendable conformance for all types crossing actor boundaries
- Swift file creation order for AI implementation
- Platform requirements (macOS 13.0+, Swift 5.9+, Xcode 15+)
- AWS CLI path detection (Homebrew ARM64 + Intel + system)
- DeploymentResult.create() factory method
- Enhanced DeploymentError with helpful messages

Previous Fixes (v4.16.1):

- RLS variable standardization (`app.current_tenant_id`)
- Type deduplication (`SelfHostedModelPricing`, `ExternalProviderPricing`)
- Billing tier localization registry entries

END OF RADIANT-PROMPT-32-v4.17.0