Contents

SECTION 26: SCHEDULED PROMPTS (v3.6.0) 1

1

26.1 Scheduled Prompts Overview 1
26.2 Scheduled Prompts Database Schema 1
26.3 Scheduler Service 2

6

SECTION 26: SCHEDULED PROMPTS (v3.6.0)

26.1 Scheduled Prompts Overview

Schedule AI tasks to run at specific times or intervals.

26.2 Scheduled Prompts Database Schema

```
-- migrations/035_scheduled_prompts.sql

CREATE TABLE scheduled_prompts (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    prompt_name VARCHAR(200) NOT NULL,
    prompt_text TEXT NOT NULL,
    model VARCHAR(100) NOT NULL,
    schedule_type VARCHAR(20) NOT NULL,
    cron_expression VARCHAR(100),
    run_at TIMESTAMPTZ,
    timezone VARCHAR(50) DEFAULT 'UTC',
    is_active BOOLEAN DEFAULT true,
    max_runs INTEGER,
    run_count INTEGER DEFAULT 0,
    last_run TIMESTAMPTZ,
    next_run TIMESTAMPTZ,
    notification_email VARCHAR(255),
    output_destination VARCHAR(50) DEFAULT 'email',
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE scheduled_prompt_runs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    prompt_id UUID NOT NULL REFERENCES scheduled_prompts(id) ON DELETE CASCADE,
    status VARCHAR(20) NOT NULL DEFAULT 'pending',
    output TEXT,
```

```
    tokens_used INTEGER,
    cost DECIMAL(10, 6),
    latency_ms INTEGER,
    error_message TEXT,
    started_at TIMESTAMPTZ,
    completed_at TIMESTAMPTZ,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_scheduled_prompts_user ON scheduled_prompts(tenant_id, user_id);
CREATE INDEX idx_scheduled_prompts_next_run ON scheduled_prompts(next_run) WHERE is_active = tr
CREATE INDEX idx_prompt_runs ON scheduled_prompt_runs(prompt_id, created_at DESC);

ALTER TABLE scheduled_prompts ENABLE ROW LEVEL SECURITY;
ALTER TABLE scheduled_prompt_runs ENABLE ROW LEVEL SECURITY;

CREATE POLICY scheduled_prompts_isolation ON scheduled_prompts USING (tenant_id = current_setti
CREATE POLICY prompt_runs_isolation ON scheduled_prompt_runs USING (
    prompt_id IN (SELECT id FROM scheduled_prompts WHERE tenant_id = current_setting('app.curre
);
```

26.3 Scheduler Service

```
// packages/core/src/services/scheduler-service.ts

import { Pool } from 'pg';
import { EventBridgeClient, PutRuleCommand, PutTargetsCommand, DeleteRuleCommand } from '@aws-s
import * as cronParser from 'cron-parser';

type ScheduleType = 'once' | 'cron' | 'interval';

interface ScheduleCreate {
    name: string;
    prompt: string;
    model: string;
    type: ScheduleType;
    cronExpression?: string;
    runAt?: Date;
    intervalMinutes?: number;
    timezone?: string;
    maxRuns?: number;
    notificationEmail?: string;
    outputDestination?: 'email' | 'webhook' | 'storage';
}

export class SchedulerService {
    private pool: Pool;
    private eventBridge: EventBridgeClient;
```

```
constructor(pool: Pool) {
    this.pool = pool;
    this.eventBridge = new EventBridgeClient({});
}

async createSchedule(tenantId: string, userId: string, schedule: ScheduleCreate): Promise<s
    const nextRun = this.calculateNextRun(schedule);

    const result = await this.pool.query(`
        INSERT INTO scheduled_prompts (
            tenant_id, user_id, prompt_name, prompt_text, model, schedule_type,
            cron_expression, run_at, timezone, max_runs, next_run,
            notification_email, output_destination
        )
        VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12, $13)
        RETURNING id
    `, [
        tenantId, userId, schedule.name, schedule.prompt, schedule.model, schedule.type,
        schedule.cronExpression, schedule.runAt, schedule.timezone || 'UTC', schedule.maxRu
        nextRun, schedule.notificationEmail, schedule.outputDestination || 'email'
    ]);

    const promptId = result.rows[0].id;

    // Create EventBridge rule for cron schedules
    if (schedule.type === 'cron' && schedule.cronExpression) {
        await this.createEventBridgeRule(promptId, schedule.cronExpression);
    }

    return promptId;
}

async executeScheduledPrompt(promptId: string): Promise<string> {
    const prompt = await this.getScheduledPrompt(promptId);
    if (!prompt || !prompt.is_active) {
        throw new Error('Scheduled prompt not found or inactive');
    }

    // Create run record
    const runResult = await this.pool.query(`
        INSERT INTO scheduled_prompt_runs (prompt_id, status, started_at)
        VALUES ($1, 'running', NOW())
        RETURNING id
    `, [promptId]);

    const runId = runResult.rows[0].id;
```

```
        try {
            // Execute the prompt (would call AI service)
            const startTime = Date.now();
            const output = await this.executePrompt(prompt.prompt_text, prompt.model);
            const latencyMs = Date.now() - startTime;

            // Update run record
            await this.pool.query(`
                UPDATE scheduled_prompt_runs
                SET status = 'completed', output = $2, latency_ms = $3, completed_at = NOW()
                WHERE id = $1
            `, [runId, output, latencyMs]);

            // Update schedule
            const nextRun = this.calculateNextRun({
                type: prompt.schedule_type,
                cronExpression: prompt.cron_expression
            });

            await this.pool.query(`
                UPDATE scheduled_prompts
                SET last_run = NOW(), run_count = run_count + 1, next_run = $2
                WHERE id = $1
            `, [promptId, nextRun]);

            // Check if max runs reached
            if (prompt.max_runs && prompt.run_count + 1 >= prompt.max_runs) {
                await this.pool.query(`UPDATE scheduled_prompts SET is_active = false WHERE id
            }

            return runId;
        } catch (error: any) {
            await this.pool.query(`
                UPDATE scheduled_prompt_runs
                SET status = 'failed', error_message = $2, completed_at = NOW()
                WHERE id = $1
            `, [runId, error.message]);

            throw error;
        }
    }

    async getScheduledPrompt(promptId: string): Promise<any> {
        const result = await this.pool.query(`SELECT * FROM scheduled_prompts WHERE id = $1`,
        return result.rows[0];
    }

    async getUserSchedules(tenantId: string, userId: string): Promise<any[]> {
```

```
    const result = await this.pool.query(`
        SELECT sp.*,
               (SELECT COUNT(*) FROM scheduled_prompt_runs WHERE prompt_id = sp.id) as tota
               (SELECT status FROM scheduled_prompt_runs WHERE prompt_id = sp.id ORDER BY c
        FROM scheduled_prompts sp
        WHERE sp.tenant_id = $1 AND sp.user_id = $2
        ORDER BY sp.created_at DESC
    `, [tenantId, userId]);

    return result.rows;
}

async pauseSchedule(promptId: string): Promise<void> {
    await this.pool.query(`UPDATE scheduled_prompts SET is_active = false WHERE id = $1`,
}

async resumeSchedule(promptId: string): Promise<void> {
    const nextRun = this.calculateNextRun(await this.getScheduledPrompt(promptId));
    await this.pool.query(`
        UPDATE scheduled_prompts SET is_active = true, next_run = $2 WHERE id = $1
    `, [promptId, nextRun]);
}

private calculateNextRun(schedule: any): Date | null {
    if (schedule.type === 'once' && schedule.runAt) {
        return new Date(schedule.runAt);
    }

    if (schedule.type === 'cron' && schedule.cronExpression) {
        const interval = cronParser.parseExpression(schedule.cronExpression);
        return interval.next().toDate();
    }

    return null;
}

private async createEventBridgeRule(promptId: string, cronExpression: string): Promise<voi
    const ruleName = `radiant-schedule-${promptId}`;

    await this.eventBridge.send(new PutRuleCommand({
        Name: ruleName,
        ScheduleExpression: `cron(${cronExpression})`,
        State: 'ENABLED'
    }));

    await this.eventBridge.send(new PutTargetsCommand({
        Rule: ruleName,
        Targets: [{
```

```
                Id: 'scheduled-prompt-target',
                Arn: process.env.SCHEDULER_LAMBDA_ARN!,
                Input: JSON.stringify({ promptId })
            }]
        }));
    }

    private async executePrompt(prompt: string, model: string): Promise<string> {
        // This would call the AI service
        return `Executed prompt with model ${model}`;
    }
}
```