Contents

SECTION 10: VISUAL AI PIPELINE (v2.3.0)

10.1 Pipeline Overview

The Visual AI Pipeline extends RADIANT with 13 new self-hosted AI models for product photography and video post-production workflows.

New Models Added

Model	Category	Purpose
SAM 2 Large	Segmentation	Precise object/subject isolation
SAM 2 Base Plus	Segmentation	Balanced speed/quality
SAM 2 Small	Segmentation	Fast lightweight segmentation
SAM 2 Tiny	Segmentation	Edge deployment
XMem	Video	Temporal mask propagation
LaMa	Inpainting	Context-aware fill
RIFE	Interpolation	Frame rate upscaling
Real-ESRGAN 4x	Upscaling	Photo-realistic enhancement
Real-ESRGAN Anime	Upscaling	Animation optimization
GFPGAN	Face	Face restoration
CodeFormer	Face	Face enhancement
Background Matting V2	Matting	Alpha matte extraction
MODNet	Matting	Real-time portraits

10.2 Database Schema Extensions

```
-- migrations/020_visual_ai_pipeline.sql

-- Model thermal state tracking
ALTER TABLE self_hosted_models ADD COLUMN IF NOT EXISTS thermal_state VARCHAR(20) DEFAULT 'COLI
ALTER TABLE self_hosted_models ADD COLUMN IF NOT EXISTS warm_until TIMESTAMPTZ;
ALTER TABLE self_hosted_models ADD COLUMN IF NOT EXISTS auto_thermal_enabled BOOLEAN DEFAULT t
```

```sql
-- Visual pipeline job tracking
CREATE TABLE visual_pipeline_jobs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    pipeline_type VARCHAR(50) NOT NULL,
    source_asset_key VARCHAR(500) NOT NULL,
    output_asset_key VARCHAR(500),
    models_used TEXT[] NOT NULL,
    parameters JSONB DEFAULT '{}',
    status VARCHAR(20) NOT NULL DEFAULT 'pending',
    progress INTEGER DEFAULT 0,
    started_at TIMESTAMPTZ,
    completed_at TIMESTAMPTZ,
    error_message TEXT,
    cost DECIMAL(10, 6),
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_visual_jobs_tenant ON visual_pipeline_jobs(tenant_id);
CREATE INDEX idx_visual_jobs_status ON visual_pipeline_jobs(status);

ALTER TABLE visual_pipeline_jobs ENABLE ROW LEVEL SECURITY;
CREATE POLICY visual_jobs_isolation ON visual_pipeline_jobs USING (tenant_id = current_setting
```

10.3 Thermal State Service

```typescript
// packages/core/src/services/thermal-state-service.ts

import { Pool } from 'pg';
import { SageMakerClient, DescribeEndpointCommand, UpdateEndpointCommand } from '@aws-sdk/clie

export type ThermalState = 'OFF' | 'COLD' | 'WARM' | 'HOT' | 'AUTOMATIC';
export type ServiceState = 'RUNNING' | 'DEGRADED' | 'DISABLED' | 'OFFLINE';

interface ThermalConfig {
    warmDurationMinutes: number;
    hotThresholdRequestsPerMinute: number;
    coldThresholdIdleMinutes: number;
}

export class ThermalStateService {
    private pool: Pool;
    private sagemaker: SageMakerClient;

    constructor(pool: Pool) {
        this.pool = pool;
        this.sagemaker = new SageMakerClient({});
```

```
}

async getThermalState(modelId: string): Promise<ThermalState> {
    const result = await this.pool.query(
        `SELECT thermal_state, warm_until, auto_thermal_enabled FROM self_hosted_models WHE
        [modelId]
    );

    if (result.rows.length === 0) throw new Error('Model not found');

    const { thermal_state, warm_until, auto_thermal_enabled } = result.rows[0];

    if (auto_thermal_enabled && warm_until && new Date(warm_until) < new Date()) {
        await this.transitionToCold(modelId);
        return 'COLD';
    }

    return thermal_state;
}

async warmUp(modelId: string, durationMinutes: number = 30): Promise<void> {
    const warmUntil = new Date(Date.now() + durationMinutes * 60 * 1000);

    await this.pool.query(
        `UPDATE self_hosted_models SET thermal_state = 'WARM', warm_until = $2 WHERE id = $
        [modelId, warmUntil]
    );

    // Trigger SageMaker endpoint if needed
    const model = await this.getModel(modelId);
    if (model.endpoint_name) {
        await this.ensureEndpointRunning(model.endpoint_name);
    }
}

async transitionToCold(modelId: string): Promise<void> {
    await this.pool.query(
        `UPDATE self_hosted_models SET thermal_state = 'COLD', warm_until = NULL WHERE id =
        [modelId]
    );
}

private async getModel(modelId: string) {
    const result = await this.pool.query(`SELECT * FROM self_hosted_models WHERE id = $1`,
    return result.rows[0];
}

private async ensureEndpointRunning(endpointName: string): Promise<void> {
```

```
        const command = new DescribeEndpointCommand({ EndpointName: endpointName });
        const response = await this.sagemaker.send(command);

        if (response.EndpointStatus !== 'InService') {
            console.log(`Endpoint ${endpointName} status: ${response.EndpointStatus}`);
        }
    }
}
```

10.4 Visual Pipeline Handler

```
// packages/lambdas/visual-pipeline/handler.ts

import { S3Client, GetObjectCommand, PutObjectCommand } from '@aws-sdk/client-s3';
import { SageMakerRuntimeClient, InvokeEndpointCommand } from '@aws-sdk/client-sagemaker-runti

interface PipelineRequest {
    tenantId: string;
    userId: string;
    pipelineType: 'segment' | 'inpaint' | 'upscale' | 'interpolate' | 'face_restore';
    sourceAssetKey: string;
    parameters: Record<string, any>;
}

export async function handler(event: PipelineRequest) {
    const s3 = new S3Client({});
    const sagemaker = new SageMakerRuntimeClient({});

    const pipelineHandlers: Record<string, (input: Buffer, params: any) => Promise<Buffer>> =
        segment: async (input, params) => {
            const endpoint = params.quality === 'high' ? 'sam2-large' : 'sam2-base';
            return invokeSageMaker(sagemaker, endpoint, input);
        },
        inpaint: async (input, params) => {
            return invokeSageMaker(sagemaker, 'lama-inpaint', input, { mask: params.maskData }
        },
        upscale: async (input, params) => {
            const endpoint = params.style === 'anime' ? 'realesrgan-anime' : 'realesrgan-4x';
            return invokeSageMaker(sagemaker, endpoint, input);
        },
        interpolate: async (input, params) => {
            return invokeSageMaker(sagemaker, 'rife-interpolation', input, { targetFps: params
        },
        face_restore: async (input, params) => {
            const endpoint = params.method === 'codeformer' ? 'codeformer' : 'gfpgan';
            return invokeSageMaker(sagemaker, endpoint, input);
        }
    };
```

```
    // Get source asset
    const sourceObj = await s3.send(new GetObjectCommand({
        Bucket: process.env.ASSETS_BUCKET!,
        Key: event.sourceAssetKey
    }));
    const inputBuffer = Buffer.from(await sourceObj.Body!.transformToByteArray());

    // Process through pipeline
    const handler = pipelineHandlers[event.pipelineType];
    const outputBuffer = await handler(inputBuffer, event.parameters);

    // Save result
    const outputKey = `processed/${event.tenantId}/${Date.now()}_${event.pipelineType}.png`;
    await s3.send(new PutObjectCommand({
        Bucket: process.env.ASSETS_BUCKET!,
        Key: outputKey,
        Body: outputBuffer,
        ContentType: 'image/png'
    }));

    return { outputAssetKey: outputKey };
}

async function invokeSageMaker(
    client: SageMakerRuntimeClient,
    endpoint: string,
    input: Buffer,
    extraParams?: Record<string, any>
): Promise<Buffer> {
    const payload = {
        image: input.toString('base64'),
        ...extraParams
    };

    const response = await client.send(new InvokeEndpointCommand({
        EndpointName: endpoint,
        Body: JSON.stringify(payload),
        ContentType: 'application/json'
    }));

    const result = JSON.parse(new TextDecoder().decode(response.Body));
    return Buffer.from(result.output, 'base64');
}
```