Cloud Architecture Design Document for Stable Diffusion Web Application

Table of Contents

System Overview

The system is a web-based application for running Stable Diffusion models with the following key features:

- Frontend built with Vue.js 3
- Backend API with Flask
- Stable Diffusion model inference
- Image storage and management
- User authentication and authorization
- Request queuing and processing

Architecture Components

Frontend (Vue.js 3)

- Single Page Application (SPA)
- Component-based architecture
- State management with Pinia
- RESTful API integration
- WebSocket for real-time updates
- Responsive design

Backend (Flask)

- RESTful API endpoints
- Authentication middleware
- Request validation
- Model inference orchestration
- File handling
- Database operations

Model Serving

- Stable Diffusion model deployment
- GPU resource management
- Inference queue management
- Model versioning
- Cache management

Storage

- Object storage for generated images
- Database for metadata and user data
- Cache layer for frequent requests
- Temporary storage for processing

Message Queue

- Request queue management
- Background task processing
- Event-driven architecture
- Failed job handling

Technology Stack

Frontend

- Vue.js 3
- Pinia (State Management)
- Vue Router
- Axios
- TailwindCSS
- Socket.io-client

Backend

- Flask
- Flask-RESTful
- Flask-SQLAlchemy
- Flask-JWT-Extended
- Gunicorn
- Redis

Infrastructure

- Docker
- Docker Compose
- Nginx
- PostgreSQL
- Redis
- MinIO (Object Storage)

ML Infrastructure

- PyTorch
- CUDA
- Stable Diffusion
- Hugging Face Transformers

Infrastructure Design

Development Environment

```
Local Development Stack:
├── Docker Compose
│   ├── Frontend Container (Vue.js)
│   ├── Backend Container (Flask)
│   ├── PostgreSQL Container
│   ├── Redis Container
│   └── MinIO Container
```

Production Environment Options

Option 1: Cloud Provider (AWS)

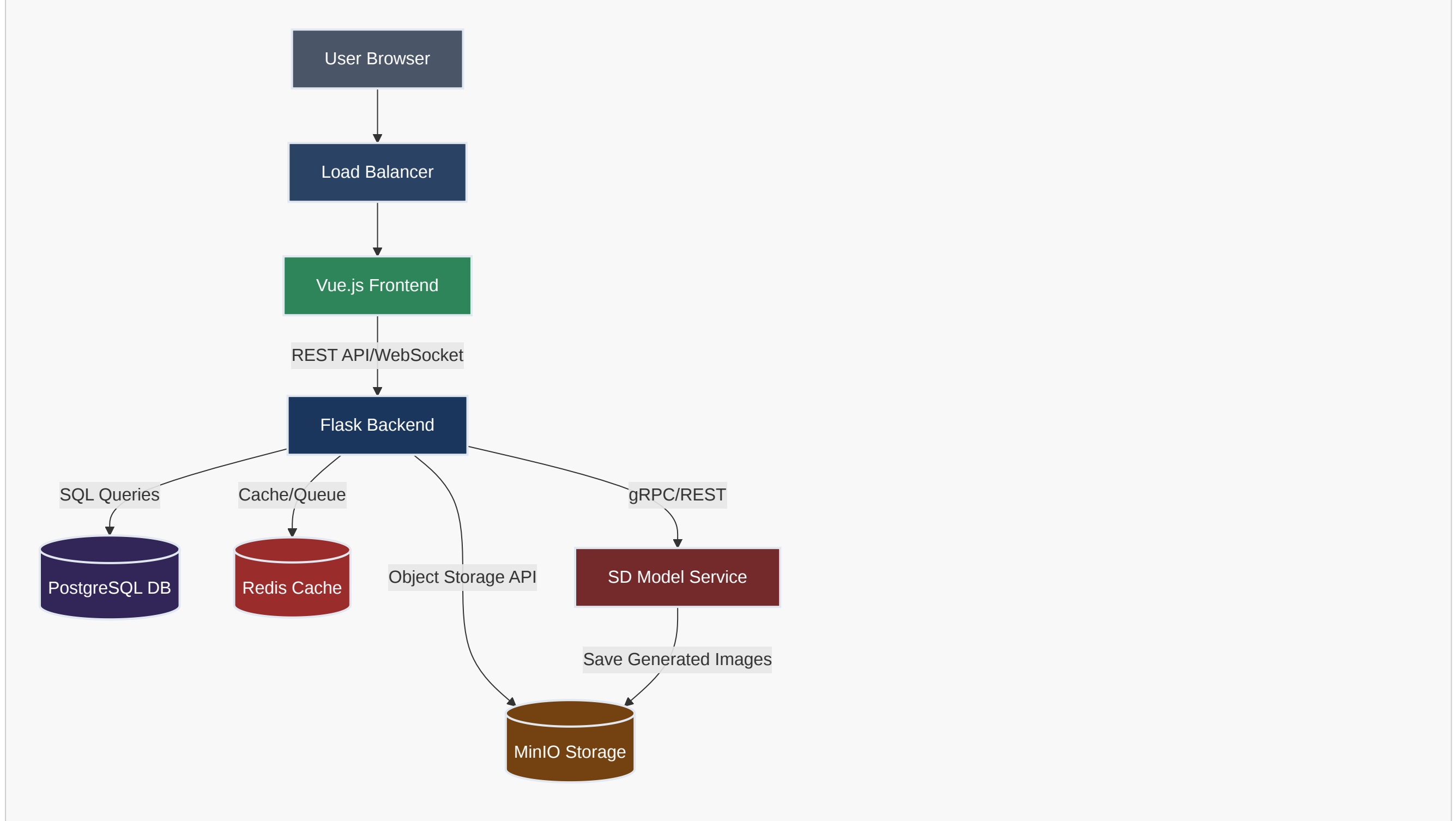
```
AWS Infrastructure:
├── Route 53 (DNS)
├── CloudFront (CDN)
├── Application Load Balancer
├── ECS Fargate
│   ├── Frontend Container
│   └── Backend Container
├── EC2 (GPU Instances)
│   └── Model Inference
├── RDS PostgreSQL
├── ElastiCache Redis
├── S3 Buckets
└── CloudWatch (Monitoring)
```

Option 2: Self-Hosted

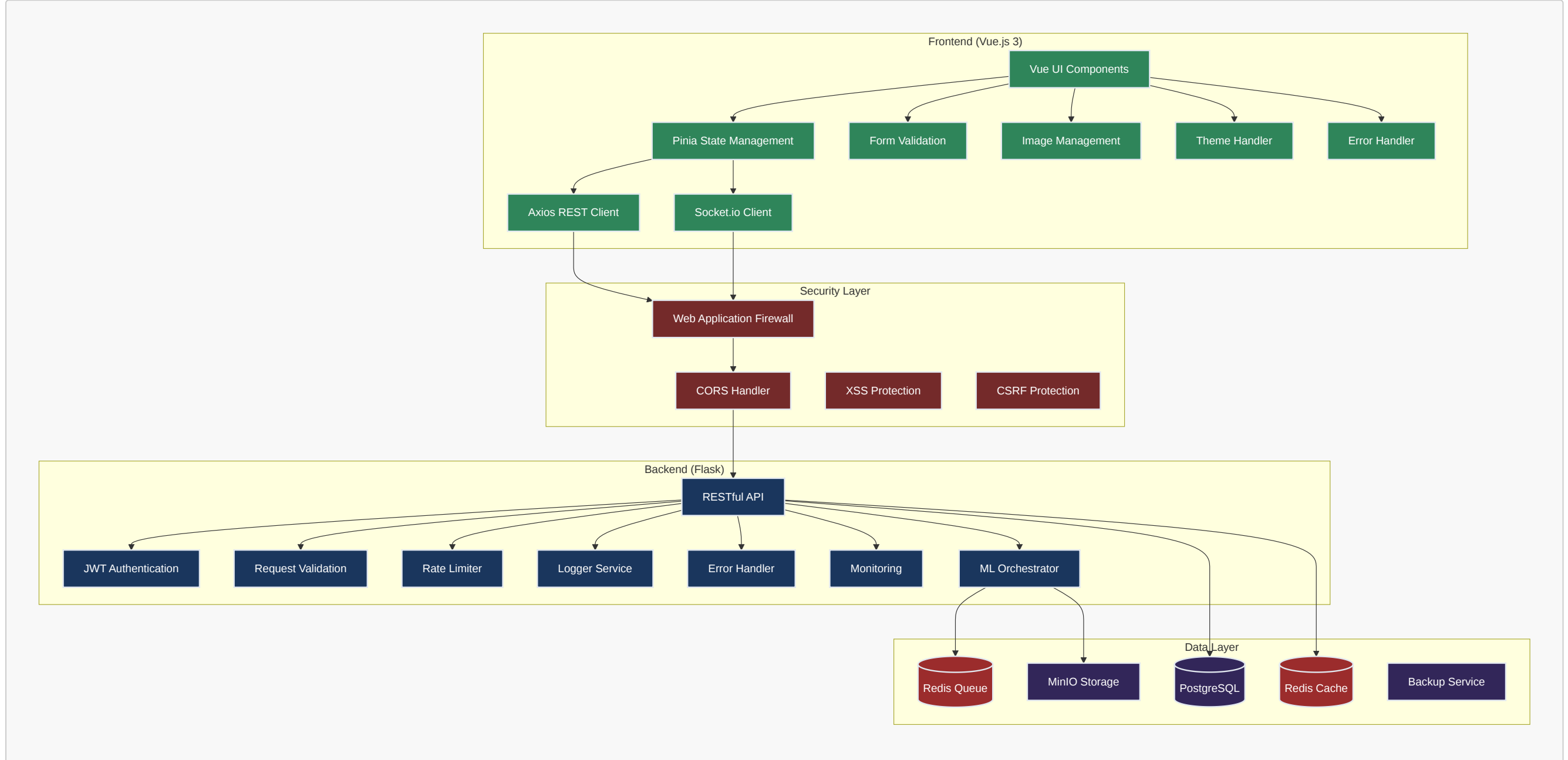
```
Self-Hosted Infrastructure:
├── Nginx (Reverse Proxy)
├── Docker Swarm/Kubernetes
│   ├── Frontend Containers
│   ├── Backend Containers
│   └── Model Inference Containers
├── PostgreSQL
├── Redis
└── MinIO
```

Visual Architecture Diagrams

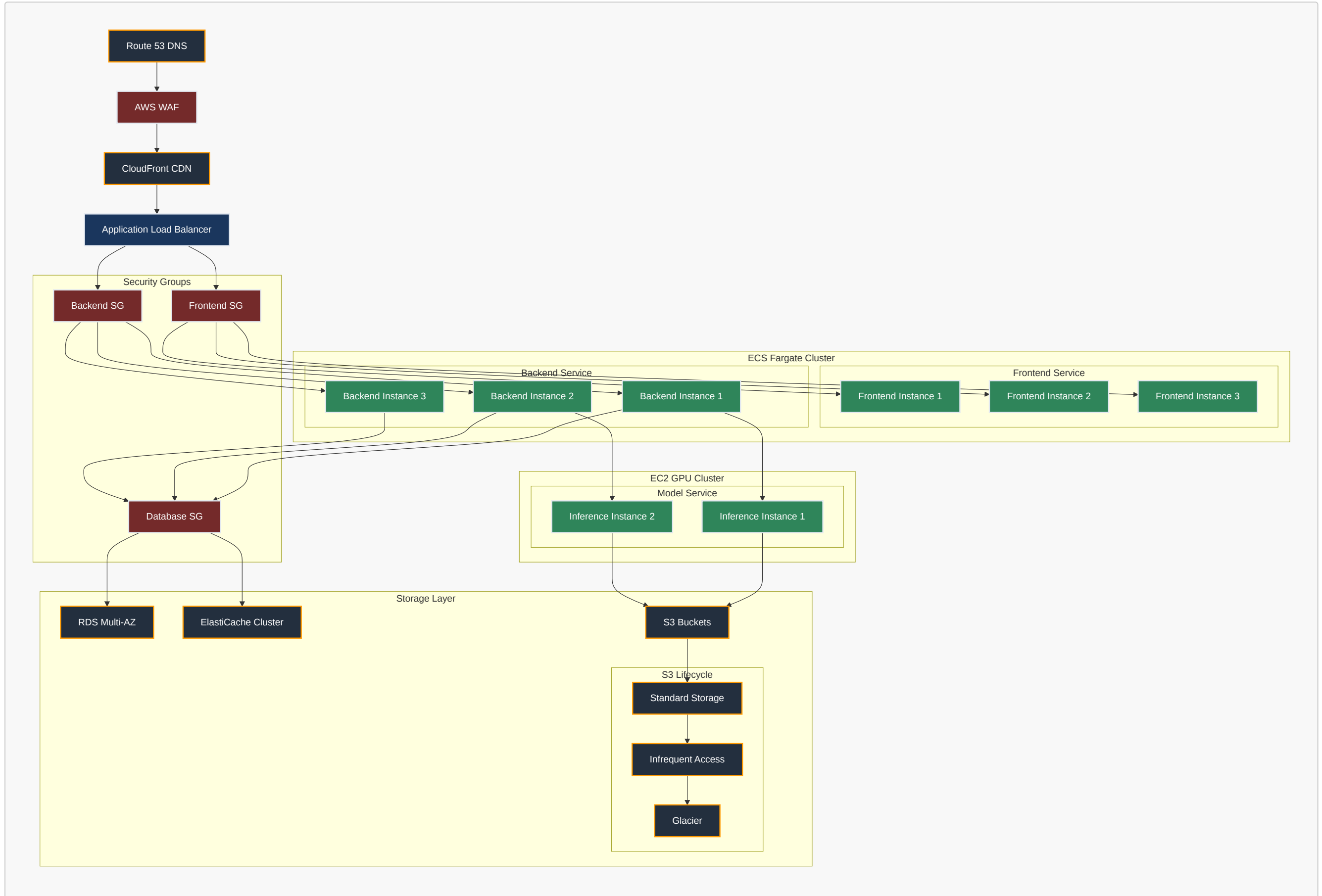
High-Level System Architecture



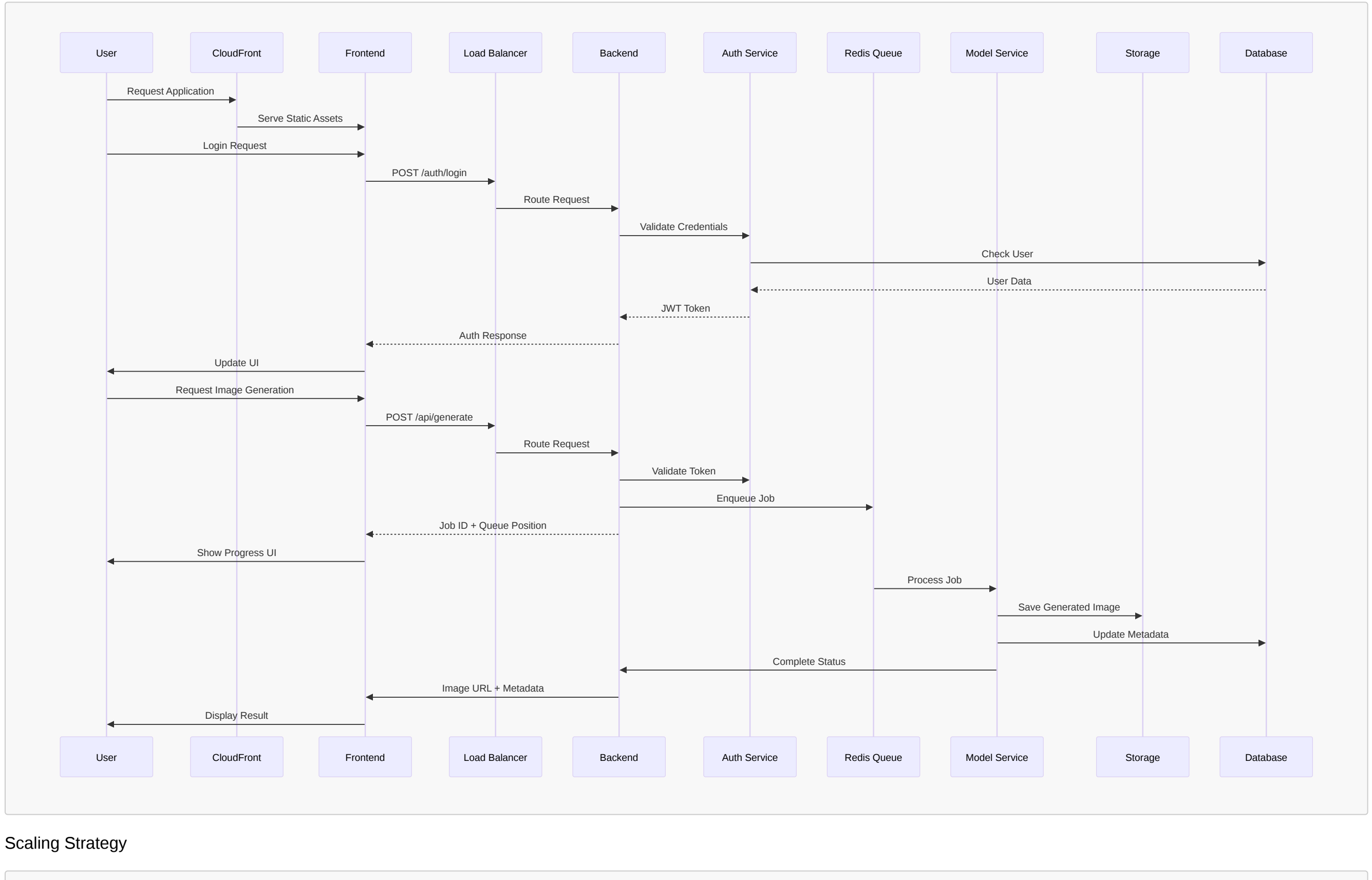
Detailed Component Architecture



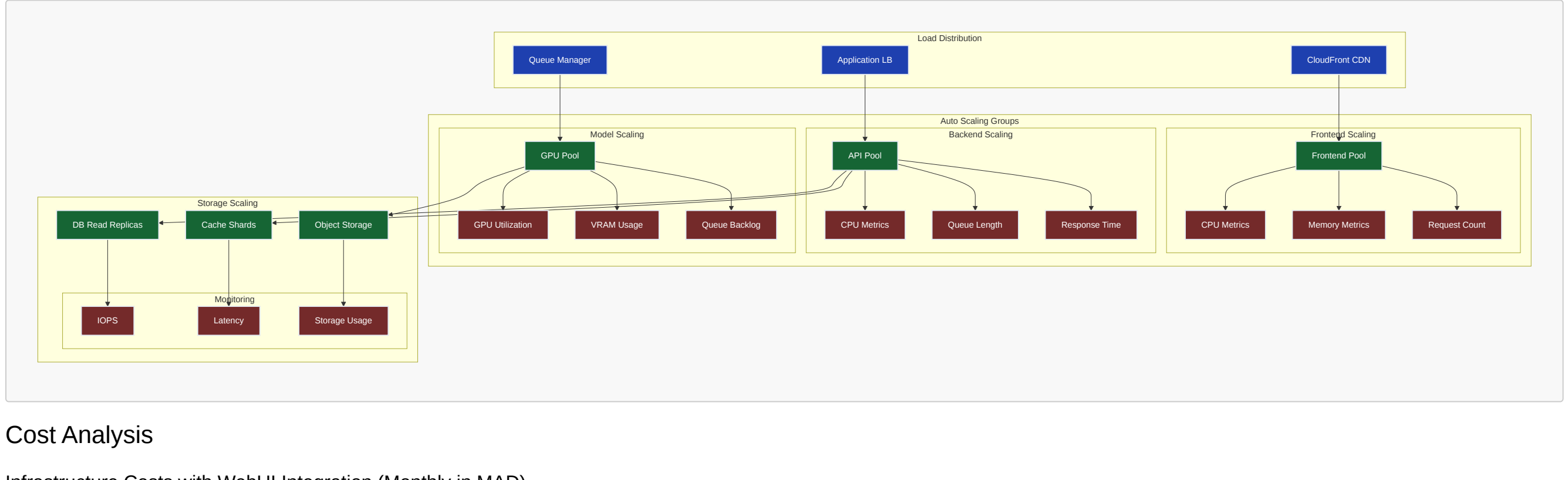
Deployment Architecture (AWS)



Data Flow Diagram



Scaling Strategy



Cost Analysis

Infrastructure Costs with WebUI Integration (Monthly in MAD)

AWS Option

1. Compute Resources
   - EC2 GPU Instance (g4dn.xlarge): 5.26 MAD/hour
     - 24/7 operation: ~3,800 MAD/month
   - ECS Fargate (2 vCPU, 4GB RAM)
     - Frontend: ~700 MAD/month
     - Backend: ~700 MAD/month
2. Storage
   - RDS PostgreSQL (db.t3.small): ~300 MAD/month
   - S3 Storage (500GB): ~120 MAD/month
   - ElastiCache Redis (cache.t3.micro): ~250 MAD/month
3. Networking
   - Data Transfer (1TB/month): ~900 MAD/month
   - CloudFront: ~500 MAD/month
   - Route 53: ~10 MAD/month

Total AWS Monthly Cost: ~7,280 MAD/month

Self-Hosted Option

1. Server Costs
   - GPU Server: ~2,000 MAD
   - Storage Server: ~500 MAD
   - Load Balancer: ~300 MAD
2. Network Costs
   - Bandwidth: ~600 MAD
   - CDN: ~400 MAD
3. Management
   - Monitoring: ~200 MAD
   - Backup Storage: ~100 MAD

Total Self-Hosted Monthly Cost: ~4,100 MAD/month

Cost Optimization Strategies

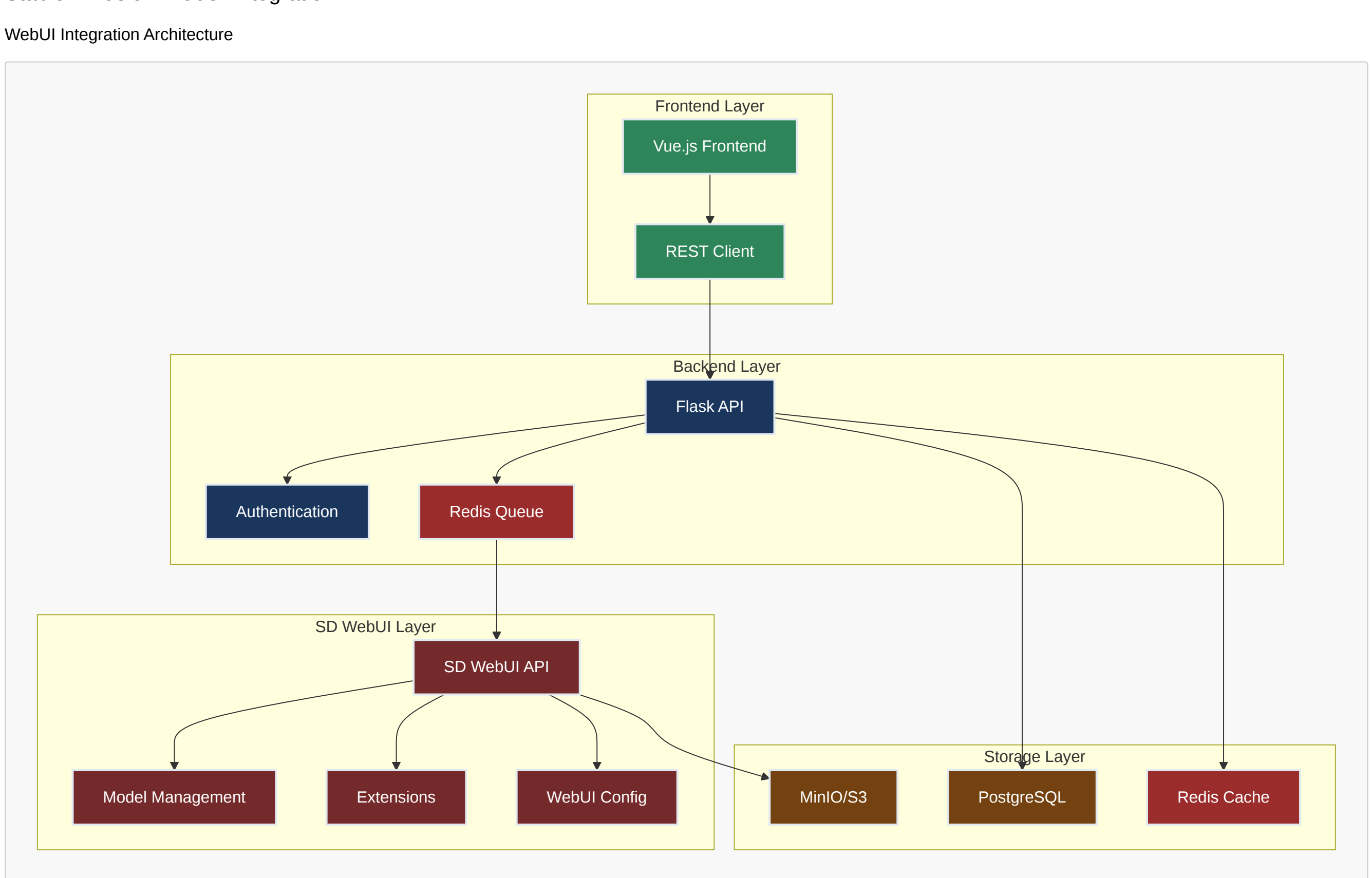
1. Use Spot Instances for GPU workloads
   - Potential savings: Up to 2,660 MAD/month
2. Reserved Instances (1-year commitment)
   - Potential savings: ~1,500 MAD/month
3. S3 Lifecycle Policies
   - Potential savings: ~50 MAD/month
4. CloudFront Caching
   - Potential savings: ~200 MAD/month

Alternative Infrastructure Options

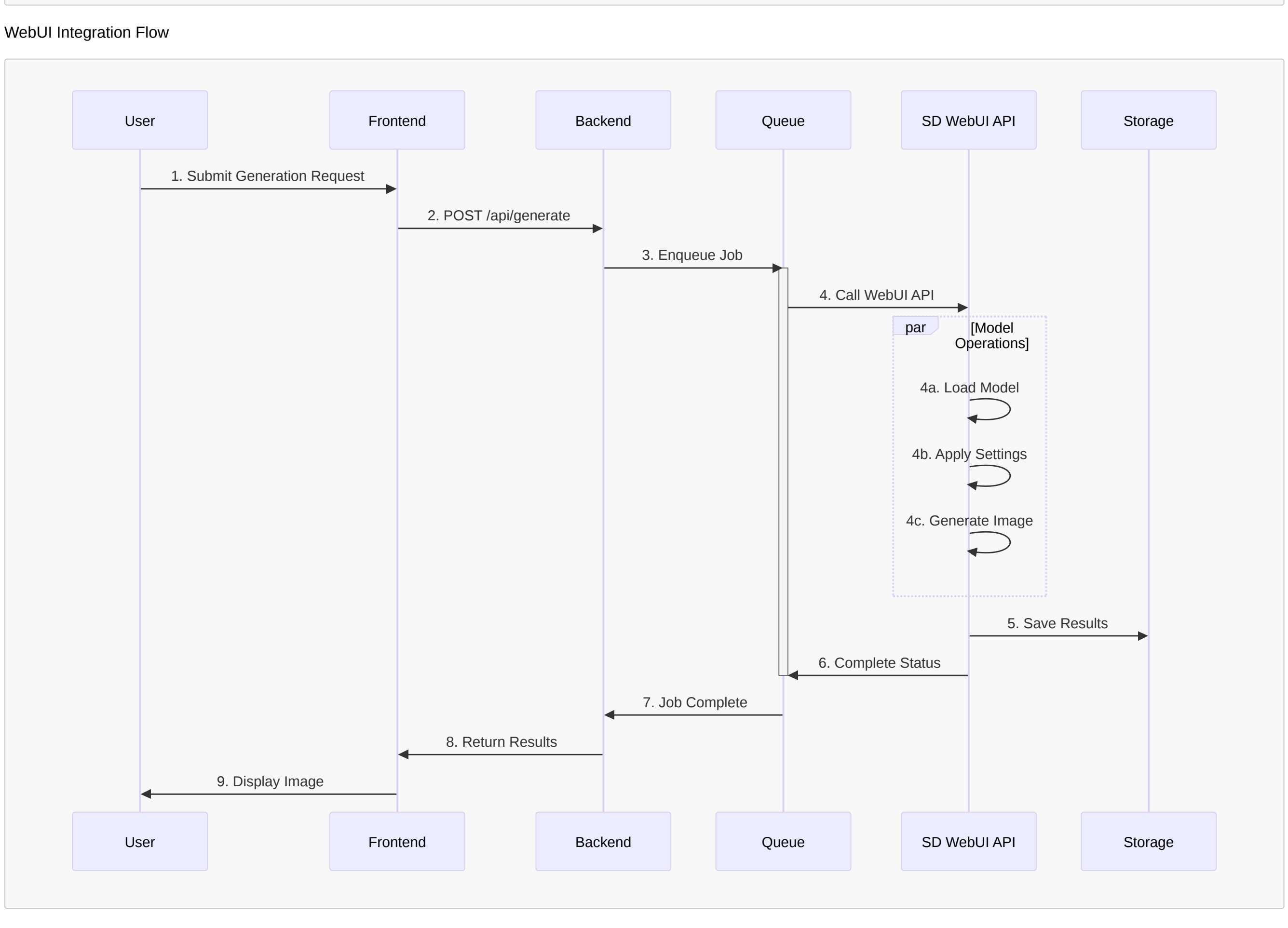
1. **GPU Resources**
   - RunPod.io (Pay as you go GPU)
   - Vast.ai (Marketplace for GPU)
   - Paperspace (GPU instances)
2. **Storage Solutions**
   - MinIO (Self-hosted S3 compatible)
   - Ceph (Distributed storage)
   - SeaweedFS (Simple and fast)
3. **Monitoring Stack**
   - Prometheus + Grafana (Free, self-hosted)
   - ELK Stack (Free, self-hosted)
   - Netdata (Free, self-hosted)
4. **Load Balancing**
   - HAProxy (Free, open-source)
   - Traefik (Free, open-source)
   - Nginx (Free, open-source)
5. **Caching Layer**
   - Redis (Free, open-source)
   - Memcached (Free, open-source)
   - Varnish (Free, open-source)

Stable Diffusion WebUI Integration

WebUI Integration Architecture



WebUI Integration Flow



Cost Revisions with WebUI Integration (in MAD)

Infrastructure Costs (Monthly)

1. Compute Resources
   - EC2 GPU Instance (g4dn.xlarge): 5.26 MAD/hour
     - 24/7 operation: ~3,800 MAD/month
   - ECS Fargate (2 vCPU, 4GB RAM)
     - Frontend: ~700 MAD/month
     - Backend: ~700 MAD/month
2. Storage
   - RDS PostgreSQL (db.t3.small): ~300 MAD/month
   - S3 Storage (500GB): ~120 MAD/month
   - ElastiCache Redis (cache.t3.micro): ~250 MAD/month
3. Networking
   - Data Transfer (1TB/month): ~900 MAD/month
   - CloudFront: ~500 MAD/month
   - Route 53: ~10 MAD/month

Total Estimated Cost: ~7,280 MAD/month

WebUI Integration Benefits

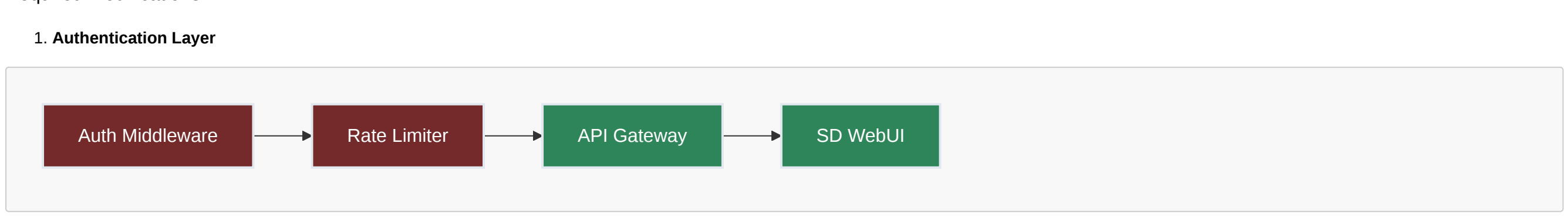
1. **Built-in Features**
   - Model Management
   - Multiple Samplers
   - Image Manipulation
   - Prompt Engineering
   - Extension System
2. **API Endpoints**

```
POST /sdapi/v1/txt2img
POST /sdapi/v1/img2img
GET  /sdapi/v1/sd-models
POST /sdapi/v1/options
GET  /sdapi/v1/progress
```

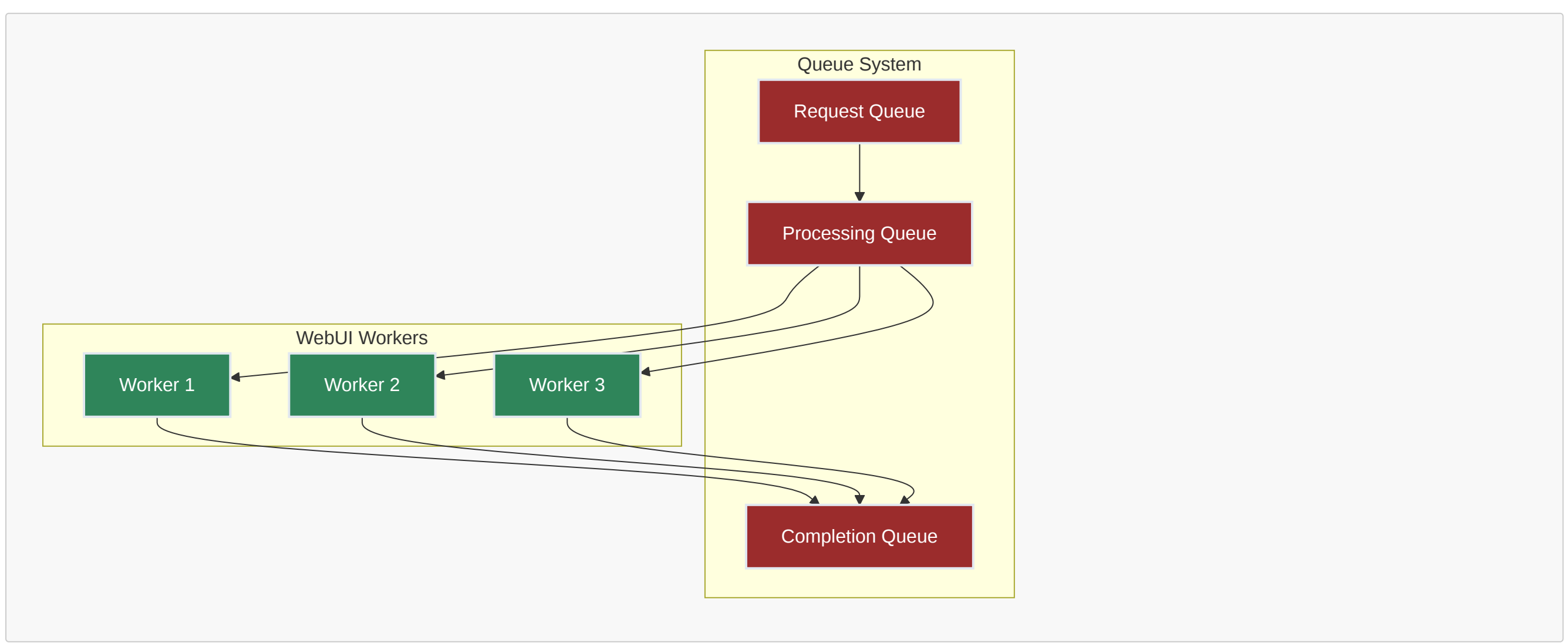
3. **Extension Integration**
   - ControlNet
   - Additional Networks
   - Dynamic Prompting
   - Image Browser
   - Upscaling

Required Modifications

1. **Authentication Layer**



2. **Queue Management**



Implementation Steps

1. **Setup Phase (1-2 weeks)**
   - Install SD WebUI
   - Configure API access
   - Setup authentication
   - Configure storage
2. **Integration Phase (2-3 weeks)**
   - Implement API wrapper
   - Setup queue system
   - Create frontend interface
   - Add monitoring
3. **Optimization Phase (1-2 weeks)**
   - Performance tuning
   - Security hardening
   - Load testing
   - Documentation

Production Configuration

```
stable-diffusion-webui:
  api:
    host: 0.0.0.0
    port: 7860
    cors_origins: ["*"]
    authentication: true

  models:
    location: /models
    default_model: sd_v1_5.safetensors

  compute:
    precision: fp16
    xformers: true
    cuda: true

  queue:
    max_size: 100
    timeout: 600
    retry_count: 3

  storage:
    output_dir: /outputs
    save_format: webp
    quality: 90
```

Conclusion

This architecture provides a scalable and maintainable solution for a Stable Diffusion web application. The estimated total implementation time is 9-13 weeks, with costs varying based on chosen infrastructure provider and scale of operation.