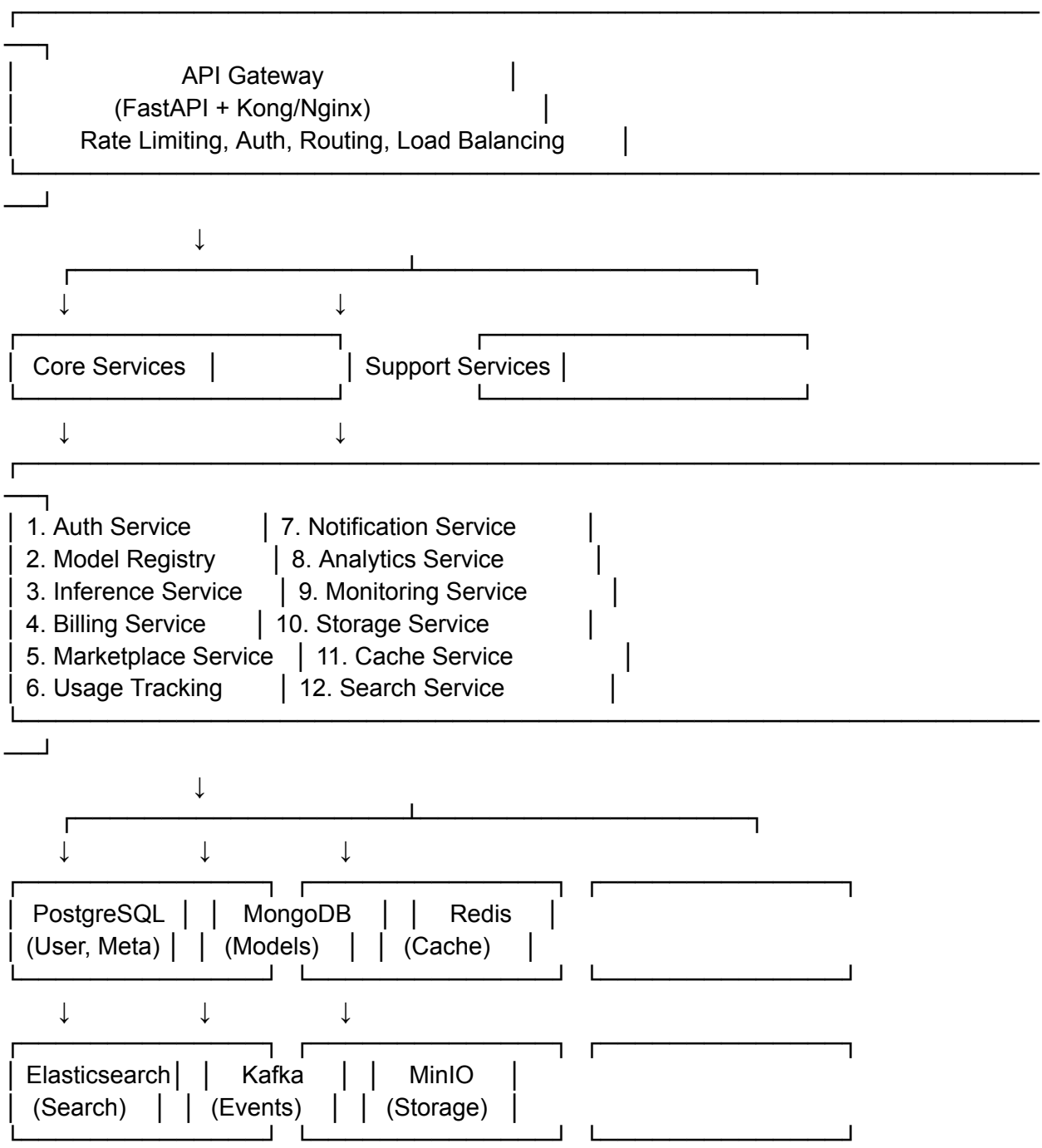
AI Model Marketplace & Inference Platform - Complete Blueprint

🎯 Project Overview

What We're Building:

A comprehensive platform where AI/ML developers can publish, monetize, and manage their models, while consumers can discover, test, and integrate AI capabilities via simple APIs - think of it as "Stripe for AI Models" or "AWS Marketplace for ML".

🏗️ Complete Architecture



📦 Complete Tech Stack

Backend Framework & Languages

FastAPI (Python 3.11+) - All microservices
Pydantic - Data validation
SQLAlchemy 2.0 - ORM (async)
Alembic - Database migrations
Celery - Background task processing
AsyncIO - Async operations

Databases

PostgreSQL 15 - Primary database (users, transactions, billing)
MongoDB - Model metadata, configurations, large JSON
Redis 7 - Caching, session management, rate limiting
TimescaleDB - Time-series data (usage metrics, analytics)

Message Queue & Event Streaming

Apache Kafka - Event streaming, distributed transactions
RabbitMQ - Task queues for Celery
Redis Streams - Real-time notifications

Search & Analytics

Elasticsearch 8 - Full-text search, model discovery
Kibana - Log visualization
ClickHouse (optional) - Heavy analytics queries

Storage

MinIO - S3-compatible object storage (model files, images)
Local SSD - Hot cache for frequently used models

ML/AI Infrastructure

ONNX Runtime - Model inference (framework agnostic)
TorchServe - PyTorch model serving
TensorFlow Serving - TensorFlow models
Triton Inference Server - Multi-framework support
Ray Serve - Distributed inference (scaling)

API Gateway & Proxy

Nginx - Reverse proxy, load balancing
Kong (optional) - Advanced API gateway features
Traefik (alternative) - Cloud-native proxy

Monitoring & Observability

Prometheus - Metrics collection
Grafana - Dashboards and visualization
Jaeger - Distributed tracing
OpenTelemetry - Instrumentation
Sentry - Error tracking

Security

JWT - Authentication tokens
OAuth 2.0 - Third-party authentication
HashiCorp Vault - Secrets management
SSL/TLS - Encryption in transit
Cryptography - Data encryption at rest

Payment Processing

Stripe API - Payment gateway
Webhook handling - Payment confirmations

DevOps & Infrastructure

Docker - Containerization
Docker Compose - Local development
Kubernetes - Container orchestration (production)
Helm - Kubernetes package manager
ArgoCD - GitOps deployment
GitHub Actions - CI/CD pipelines

Testing

Pytest - Unit testing
Pytest-asyncio - Async testing
Locust - Load testing
Testcontainers - Integration testing

Documentation

Swagger/OpenAPI - API documentation
Redoc - Alternative API docs
MkDocs - User documentation

🎨 Complete Feature Set
Phase 1: Core Platform (MVP)
1. User Management & Authentication

User registration (email/OAuth)
Email verification
Multi-factor authentication (MFA)
JWT-based authentication
API key generation and management
Role-based access control (RBAC)

Model Publishers
Model Consumers
Admins

User profiles and preferences
Password reset flows

2. Model Registry Service

Model upload (chunked upload for large files)
Model versioning (semantic versioning)
Model metadata management

Name, description, tags
Input/output schemas
Framework (PyTorch, TensorFlow, ONNX)
Model size, parameters count
License information

Model validation

Format validation
Schema validation
Security scanning

Model storage (MinIO/S3)
Model lifecycle management

Draft → Review → Published → Deprecated

3. Model Marketplace

Browse models by category

NLP (text generation, classification, NER)
Computer Vision (classification, detection, segmentation)
Audio (speech-to-text, text-to-speech)
Multimodal (CLIP, image captioning)

Search functionality (Elasticsearch)

Full-text search
Filter by tags, framework, pricing
Sort by popularity, rating, date

Model detail pages

Overview, documentation
Sample inputs/outputs
Pricing information
Usage statistics
Reviews and ratings

Model playground

Interactive testing interface
Try before you buy
Sample requests

4. Inference Service

REST API endpoints for inference

Request validation
Model loading and caching
GPU/CPU resource management
Batch inference support
Async inference for long-running tasks
Response formatting
Error handling and retries

5. Billing & Subscription Service

Pricing tiers

Free tier (limited requests)
Pay-as-you-go (per API call)
Monthly subscriptions
Enterprise plans

Stripe integration

Payment method management
Subscription management
Invoice generation

Usage-based billing

Track API calls per user/model
Calculate costs dynamically

Revenue sharing (platform fee + model owner payout)
Billing dashboard
Payment history

Phase 2: Advanced Features
6. Usage Tracking & Analytics

Real-time usage monitoring
Request/response logging
Performance metrics

Latency (p50, p95, p99)

Throughput (requests/second)
Error rates

Cost tracking per model/user
Dashboard for publishers

Total requests
Revenue earned
Popular models
Geographic distribution

Dashboard for consumers

Usage statistics
Cost breakdown
Model performance

7. Rate Limiting & Quotas

Token bucket algorithm
Per-user rate limits
Per-model rate limits
Quota management

Daily/monthly limits
Soft limits (warnings)
Hard limits (rejection)

Tier-based limits
Burst allowance

8. Caching Layer

Response caching for identical requests
Cache invalidation strategies
TTL-based expiration
Cache hit/miss tracking
Distributed caching (Redis)

9. Webhook System

Webhook registration
Event types

Model published
Inference completed
Payment received
Quota exceeded

Webhook delivery
Retry mechanism with exponential backoff
Webhook verification (signatures)

10. Notification Service

Email notifications

Welcome emails
Payment confirmations
Quota warnings
Model approval/rejection

In-app notifications
Real-time alerts (WebSocket)
Notification preferences

Phase 3: Production-Grade Features
11. Model Performance Monitoring

Drift detection

Input distribution changes
Output quality degradation

A/B testing framework

Compare model versions
Traffic splitting
Statistical significance testing

Model health checks
Automated alerts for anomalies

12. Advanced Inference Features

Auto-scaling based on load
Multi-region deployment
Model ensembling
Model chaining (pipelines)
Streaming responses (for text generation)
Custom preprocessing/postprocessing
Model warm-up strategies

13. Idempotency & Reliability

Idempotency keys for requests
Request deduplication
Distributed transactions (Saga pattern)
Circuit breaker pattern
Fallback mechanisms
Graceful degradation

14. Security Features

API key rotation
IP whitelisting/blacklisting
Request signing
DDoS protection
SQL injection prevention
XSS protection
CORS configuration
Rate limit bypass protection
Audit logging

All sensitive operations logged
Immutable audit trail
Compliance reporting

15. Advanced Search & Discovery

Recommendation engine

Similar models
Personalized recommendations

Model comparison tool
Leaderboards

Most popular
Highest rated
Best performance

Collections/curated lists

16. Developer Experience

SDK generation (Python, JavaScript, Go)
Code examples
Interactive API documentation
Postman collections
CLI tool for model management
Webhooks testing tool
Sandbox environment

Phase 4: Enterprise Features
17. Multi-Tenancy

Organization accounts
Team management
Shared API keys
Organization billing
Usage aggregation

18. Custom Deployment

Private model hosting
VPC deployment
On-premise support
Dedicated infrastructure

19. Compliance & Governance

Model provenance tracking
Data lineage
Compliance reports (SOC 2, GDPR)
Data retention policies
Right to deletion

20. Advanced Analytics

Business intelligence dashboards
Predictive analytics
Cost optimization recommendations
Anomaly detection
Custom reports

🗄️ Database Schema Design
PostgreSQL Tables

```
sql-- Users
users (
    id UUID PRIMARY KEY,
    email VARCHAR UNIQUE,
    username VARCHAR UNIQUE,
    password_hash VARCHAR,
    role VARCHAR,
    is_verified BOOLEAN,
    created_at TIMESTAMP,
    updated_at TIMESTAMP,
    deleted_at TIMESTAMP NULL  -- Soft delete
)

-- API Keys
api_keys (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users,
    key_hash VARCHAR,
    name VARCHAR,
    scopes JSON,
    last_used_at TIMESTAMP,
    expires_at TIMESTAMP,
    created_at TIMESTAMP,
    is_active BOOLEAN
)
```

```
-- Subscriptions
subscriptions (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users,
    plan_id UUID REFERENCES plans,
    stripe_subscription_id VARCHAR,
    status VARCHAR,
    current_period_start TIMESTAMP,
    current_period_end TIMESTAMP,
    created_at TIMESTAMP
)

-- Transactions
transactions (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users,
    amount DECIMAL,
    currency VARCHAR,
    type VARCHAR,  -- charge, refund, payout
    status VARCHAR,
    stripe_payment_id VARCHAR,
    metadata JSON,
    created_at TIMESTAMP
)

-- Usage Records (TimescaleDB)
usage_records (
    id UUID,
    user_id UUID,
    model_id UUID,
    api_key_id UUID,
    request_id UUID,
    tokens_used INT,
    cost DECIMAL,
    latency_ms INT,
    status_code INT,
    timestamp TIMESTAMP,
    PRIMARY KEY (id, timestamp)
) PARTITION BY RANGE (timestamp);

-- Audit Logs
audit_logs (
    id UUID PRIMARY KEY,
    user_id UUID,
```

```
    action VARCHAR,
    resource_type VARCHAR,
    resource_id UUID,
    metadata JSON,
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMP
)
```

MongoDB Collections

```javascript
// Models
{
    _id: ObjectId,
    model_id: UUID,
    owner_id: UUID,
    name: String,
    description: String,
    version: String,
    framework: String,  // pytorch, tensorflow, onnx
    task_type: String,  // classification, generation, etc.
    tags: [String],

    // Storage
    storage_path: String,
    file_size: Number,
    checksum: String,

    // Schema
    input_schema: Object,
    output_schema: Object,

    // Configuration
    inference_config: {
        max_batch_size: Number,
        timeout_ms: Number,
        gpu_required: Boolean,
        memory_mb: Number
    },

    // Pricing
    pricing: {
        free_tier_limit: Number,
        price_per_request: Number,
        price_per_token: Number,
        subscription_prices: Object
```

```
  },

  // Metrics
  stats: {
    total_requests: Number,
    avg_latency_ms: Number,
    success_rate: Number,
    rating: Number,
    review_count: Number
  },

  // Lifecycle
  status: String,  // draft, review, published, deprecated
  published_at: Date,
  deprecated_at: Date,

  created_at: Date,
  updated_at: Date
}

// Model Reviews
{
  _id: ObjectId,
  model_id: UUID,
  user_id: UUID,
  rating: Number,
  comment: String,
  created_at: Date
}

// Inference Cache
{
  _id: ObjectId,
  model_id: UUID,
  request_hash: String,  // Hash of input
  response: Object,
  ttl: Date,
  created_at: Date
}
```

🔧 Microservices Breakdown

1. API Gateway Service
```
Port: 8000
Technology: FastAPI + Nginx
Responsibilities:
- Request routing
- Authentication validation
- Rate limiting
- Request/response logging
- CORS handling
- SSL termination
```

2. Auth Service
```
Port: 8001
Technology: FastAPI + PostgreSQL + Redis
Responsibilities:
- User registration/login
- JWT token generation
- API key management
- OAuth 2.0 flows
- MFA handling
- Session management
```

3. Model Registry Service
```
Port: 8002
Technology: FastAPI + MongoDB + MinIO
Responsibilities:
- Model CRUD operations
- Version management
- File upload handling
- Model validation
- Metadata management
- Search indexing
```

4. Inference Service
```
Port: 8003
Technology: FastAPI + ONNX/TorchServe + Redis
```

Responsibilities:
- Load models into memory
- Process inference requests
- Batch processing
- Result caching
- Resource management
- Performance optimization
```

### **5. Billing Service**
```
Port: 8004
Technology: FastAPI + PostgreSQL + Stripe
Responsibilities:
- Subscription management
- Payment processing
- Invoice generation
- Usage calculation
- Revenue distribution
- Webhook handling
```

### **6. Usage Tracking Service**
```
Port: 8005
Technology: FastAPI + TimescaleDB + Kafka
Responsibilities:
- Log all API requests
- Calculate usage metrics
- Generate analytics
- Quota management
- Cost tracking
```

### **7. Marketplace Service**
```
Port: 8006
Technology: FastAPI + MongoDB + Elasticsearch
Responsibilities:
- Model discovery
- Search and filtering
- Recommendations
- Reviews and ratings
- Model playground

```

### **8. Notification Service**
```
Port: 8007
Technology: FastAPI + RabbitMQ + Redis
Responsibilities:
- Email sending
- Webhook delivery
- Push notifications
- Alert management
- Notification preferences
```

### **9. Analytics Service**
```
Port: 8008
Technology: FastAPI + TimescaleDB + ClickHouse
Responsibilities:
- Dashboard data
- Report generation
- Trend analysis
- Anomaly detection
- Business intelligence
```

### **10. Storage Service**
```
Port: 8009
Technology: FastAPI + MinIO
Responsibilities:
- File upload/download
- Presigned URL generation
- File versioning
- Storage optimization
```

🚀 Development Roadmap
Week 1-2: Project Setup & Foundation

Set up project structure
Docker Compose for local development
Database setup (PostgreSQL, MongoDB, Redis)
Basic FastAPI gateway
Authentication service (JWT)
```

User CRUD operations

Week 3-4: Model Registry

Model upload endpoint (chunked)
MinIO integration
MongoDB schema for models
Model versioning
Basic validation

Week 5-6: Inference Service

ONNX Runtime integration
Model loading mechanism
Basic inference endpoint
Response caching (Redis)
Error handling

Week 7-8: Billing Integration

Stripe integration
Subscription plans
Usage tracking basics
Payment webhooks

Week 9-10: Marketplace

Model listing API
Search with Elasticsearch
Model detail pages
Playground interface

Week 11-12: Advanced Features

Rate limiting
Idempotency
Circuit breaker
Distributed tracing
Monitoring setup

Week 13-14: Event-Driven Architecture

Kafka setup
Event producers/consumers

Saga pattern for transactions
Webhook system

Week 15-16: Testing & Optimization

Unit tests
Integration tests
Load testing
Performance optimization
Documentation

Week 17-18: DevOps & Deployment

Kubernetes manifests
CI/CD pipelines
Monitoring dashboards
Production deployment

📊 Key Metrics to Track

Business Metrics

Total users (publishers + consumers)
Active models
Total API requests
Revenue (MRR, ARR)
Churn rate
Customer acquisition cost

Technical Metrics

API latency (p50, p95, p99)
Error rate
Uptime (SLA: 99.9%)
Cache hit rate
Database query performance
Message queue lag

User Metrics

Model upload success rate
Average inference time
Cost per request
Most popular models

User retention

🎯 Success Criteria
After completing this project, you'll have:
✅ Portfolio Project - Production-grade system for interviews
✅ Microservices Expertise - Built 10+ interconnected services
✅ Event-Driven Mastery - Kafka, Saga pattern, CQRS
✅ Payment Integration - Real Stripe implementation
✅ Scalability Knowledge - Caching, sharding, load balancing
✅ DevOps Skills - Docker, Kubernetes, CI/CD
✅ ML Integration - AI model serving at scale
✅ Advanced Patterns - Circuit breaker, idempotency, distributed tracing