DiagnosisAI - Technical Architecture

Version: 2.0
Last Updated: December 28, 2025
Author: Bharath Nagesh

Table of Contents

Executive Summary

Problem Statement

Doctors in India and the USA face overwhelming patient volumes (40+ patients/day) while juggling symptoms analysis, lab interpretation, treatment protocols, and documentation. Average consultation time: 2-3 hours per patient. High cognitive load increases diagnostic errors and physician burnout.

Solution

DiagnosisAI is a patient-centric AI-powered clinical decision support system that:
- Reduces consultation time from 2.4h to 1.1h (54% reduction)
- Maintains 94%+ diagnostic accuracy
- Works offline with cached medical literature
- Requires zero learning curve (designed for minimal tech proficiency)
- Integrates with existing EMR systems

Key Architectural Decisions

1. Patient-Centric Organization
- All features organized around individual patient profiles
- Five-tab navigation: Overview, Diagnosis, Labs, Treatment, Notes
- Complete patient context always visible

2. Offline-First Architecture
- Local-first data storage with sync when connected
- Cached medical literature and guidelines
- Queue-based sync for EMR updates

3. Color-Coded Visual System
- Red borders = Critical urgency
- Orange borders = Moderate urgency
- Green borders = Routine
- No text labels needed for instant recognition

4. Progressive Web App (PWA) + Native Wrapper
- Web technologies for rapid iteration
- Capacitor for native iOS/Android deployment
- Single codebase for all platforms

System Architecture

High-Level Architecture Diagram

```
┌─────────────────────────────────────────────────────────────────┐
│                         CLIENT LAYER                            │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐        │
│  │  iOS App     │  │ Android App  │  │  Web App     │        │
│  │  (Capacitor) │  │ (Capacitor)  │  │  (PWA)       │        │
│  └──────┬───────┘  └──────┬───────┘  └──────┬───────┘        │
│         │                  │                  │                 │
│         └──────────────────┴──────────────────┘                 │
│                           │                                     │
└───────────────────────────┼─────────────────────────────────────┘
                            │
┌───────────────────────────┼─────────────────────────────────────┐
│                    SERVICE LAYER                                │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  ┌─────────────────────────────────────────────────────────┐  │
│  │             API Gateway (REST/GraphQL)                   │  │
│  │             Authentication & Authorization               │  │
│  └─────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐        │
│  │  Patient     │  │  Diagnosis   │  │  Analytics   │        │
│  │  Service     │  │  Service     │  │  Service     │        │
│  └──────────────┘  └──────────────┘  └──────────────┘        │
│                                                                 │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐        │
│  │  Lab         │  │  Treatment   │  │  Notes       │        │
│  │  Service     │  │  Service     │  │  Service     │        │
│  └──────────────┘  └──────────────┘  └──────────────┘        │
│                                                                 │
└───────────────────────────┼─────────────────────────────────────┘
                            │
┌───────────────────────────┼─────────────────────────────────────┐
│                      AI/ML LAYER                                │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  ┌──────────────────────────────────────────────────────────┐ │
│  │                  Claude API (Anthropic)                   │ │
│  │           Natural Language Processing Engine             │ │
│  └──────────────────────────────────────────────────────────┘ │
│                                                                 │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐        │
│  │  Symptom     │  │  Lab         │  │  Treatment   │        │
│  │  Analysis    │  │  Interpreter │  │  Recommender │        │
│  └──────────────┘  └──────────────┘  └──────────────┘        │
│                                                                 │
└───────────────────────────┬──────────────────────────────────────┘
                            │
┌───────────────────────────┴──────────────────────────────────────┐
│                       DATA LAYER                                 │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐        │
│  │  PostgreSQL  │  │  Redis       │  │  S3/Storage  │        │
│  │  (Primary)   │  │  (Cache)     │  │  (Files)     │        │
│  └──────────────┘  └──────────────┘  └──────────────┘        │
│                                                                  │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐        │
│  │  Vector DB   │  │  Analytics   │  │  Audit Log   │        │
│  │  (Pinecone)  │  │  (BigQuery)  │  │  (MongoDB)   │        │
│  └──────────────┘  └──────────────┘  └──────────────┘        │
│                                                                  │
└───────────────────────────┬──────────────────────────────────────┘
                            │
┌───────────────────────────┴──────────────────────────────────────┐
│                  INTEGRATION LAYER                               │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐        │
│  │  PubMed API  │  │  EMR Systems │  │  Lab Systems │        │
│  │  (NIH)       │  │  (Epic/Cerner)│  │  (HL7/FHIR) │        │
│  └──────────────┘  └──────────────┘  └──────────────┘        │
│                                                                  │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐        │
│  │  WHO         │  │  CDC         │  │  Medical     │        │
│  │  Guidelines  │  │  Database    │  │  Journals    │        │
│  └──────────────┘  └──────────────┘  └──────────────┘        │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

Architecture Layers

1. Client Layer

- **iOS/Android Apps**: Native wrappers using Capacitor

- **Web App**: Progressive Web App (PWA) for desktop access
- **Offline Storage**: IndexedDB for local data persistence
- **Service Workers**: Background sync and caching

2. Service Layer

- **API Gateway**: Centralized request routing and authentication
- **Microservices**: Independent services for each major feature
- **Message Queue**: RabbitMQ/SQS for async processing
- **Load Balancer**: Distribute traffic across service instances

3. AI/ML Layer

- **Claude API**: Primary NLP engine for symptom analysis
- **Custom Models**: Fine-tuned models for specific medical domains
- **Vector Search**: Semantic search over medical literature
- **Recommendation Engine**: Treatment protocol suggestions

4. Data Layer

- **Primary Database**: PostgreSQL for structured data
- **Cache**: Redis for session and frequently accessed data
- **Vector Database**: Pinecone for embeddings and semantic search
- **Analytics Database**: BigQuery for complex analytics queries
- **File Storage**: S3/CloudFlare R2 for medical images and documents

5. Integration Layer

- **Medical Literature**: PubMed API, WHO, CDC databases
- **EMR Systems**: HL7 FHIR standard for interoperability
- **Lab Systems**: HL7 v2.x messaging for lab results
- **Pharmacology**: Drug interaction databases

Data Models

Core Entities

1. Patient

```typescript
interface Patient {
  id: string;                    // UUID
  firstName: string;
  lastName: string;
  dateOfBirth: Date;
  gender: 'male' | 'female' | 'other';
  contactInfo: ContactInfo;
  medicalHistory: MedicalHistory[];
  allergies: Allergy[];
  currentMedications: Medication[];

  // Metadata
  createdAt: Date;
  updatedAt: Date;
  lastVisit: Date;
  doctorId: string;              // Foreign key to Doctor

  // Privacy
  consentGiven: boolean;
  dataRetentionDate: Date;
}

interface ContactInfo {
  phone: string;
  email: string;
  address: Address;
  emergencyContact: EmergencyContact;
}

interface MedicalHistory {
  condition: string;
  diagnosisDate: Date;
  icd10Code: string;
  status: 'active' | 'resolved' | 'chronic';
  notes: string;
}
```

2. Visit/Encounter

```
interface Visit {
  id: string;                    // UUID
  patientId: string;             // Foreign key
  doctorId: string;              // Foreign key
  timestamp: Date;

  // Clinical Data
  chiefComplaint: string;
  presentingSymptoms: Symptom[];
  vitalSigns: VitalSigns;
  physicalExam: PhysicalExam;

  // Assessment
  differentialDiagnoses: Diagnosis[];
  primaryDiagnosis: Diagnosis;
  urgencyLevel: 'critical' | 'moderate' | 'routine';

  // Plan
  labOrders: LabOrder[];
  treatmentPlan: TreatmentPlan;
  prescriptions: Prescription[];
  followUp: FollowUp;

  // Documentation
  clinicalNote: ClinicalNote;
  status: 'active' | 'completed' | 'cancelled';

  // Metadata
  createdAt: Date;
  updatedAt: Date;
  completedAt?: Date;
}

interface VitalSigns {
  bloodPressure: {
    systolic: number;
    diastolic: number;
    timestamp: Date;
  };
  heartRate: number;              // bpm
  respiratoryRate: number;        // breaths/min
  temperature: number;            // Fahrenheit
  oxygenSaturation: number;       // percentage
  weight?: number;                // kg
  height?: number;                // cm
}

interface Symptom {
  name: string;
  onset: Date;
  duration: string;
  severity: 1 | 2 | 3 | 4 | 5;   // 1=mild, 5=severe
  location?: string;
  qualityDescriptors: string[];
  alleviatingFactors: string[];
  exacerbatingFactors: string[];
  associatedSymptoms: string[];
}
```

3. Diagnosis

```
interface Diagnosis {
  id: string;
  visitId: string;               // Foreign key

  // Diagnosis Details
  name: string;
  icd10Code: string;
  snomedCode?: string;
  confidenceScore: number;       // 0-100
  urgencyLevel: 'critical' | 'moderate' | 'routine';

  // Evidence
  supportingFindings: string[];
  contradictingFindings: string[];
  differentialReasoning: string;

  // Literature Support
  evidenceBase: {
    similarCases: number;
    studiesCount: number;
    guidelinesYear: number;
    references: Reference[];
  };

  // AI Metadata
  aiGenerated: boolean;
  aiModel: string;               // e.g., "claude-sonnet-4"
  aiPrompt?: string;
  aiResponse?: string;

  createdAt: Date;
  updatedAt: Date;
}

interface Reference {
  type: 'pubmed' | 'guideline' | 'study';
  title: string;
  authors: string[];
  journal?: string;
  year: number;
  doi?: string;
  pmid?: string;
  url: string;
}
```

4. Lab Results

```
interface LabOrder {
  id: string;
  visitId: string;
  patientId: string;

  // Order Details
  testName: string;
  loincCode: string;             // Standard lab code
  priority: 'stat' | 'urgent' | 'routine';
  orderedAt: Date;
  orderedBy: string;             // Doctor ID

  // Result
  result?: LabResult;
  status: 'ordered' | 'collected' | 'processing' | 'completed' | 'cancelled';
}

interface LabResult {
  id: string;
  labOrderId: string;

  // Result Data
  value: number | string;
  unit: string;
  referenceRange: {
    low: number;
    high: number;
    unit: string;
  };

  // Interpretation
  isAbnormal: boolean;
  abnormalFlag: 'low' | 'high' | 'critical-low' | 'critical-high' | 'normal';
  aiInterpretation: string;
  clinicalSignificance: string;

  // Metadata
  performedAt: Date;
  performedBy: string;           // Lab technician
  verifiedAt: Date;
  verifiedBy: string;            // Pathologist

  createdAt: Date;
}
```

5. Treatment Plan

```
interface TreatmentPlan {
  id: string;
  visitId: string;
  diagnosisId: string;

  // Protocol
  protocolName: string;
  protocolVersion: string;
  timeCritical: boolean;
  targetTime?: string;           // e.g., "door-to-balloon: 90 min"

  // Actions
  immediateActions: Action[];
  diagnosticWorkup: Action[];
  therapeuticInterventions: Action[];

  // Medications
  prescriptions: Prescription[];

  // Follow-up
  followUpInstructions: string;
  followUpDate?: Date;
  referrals: Referral[];

  // Status
  status: 'active' | 'completed' | 'modified' | 'discontinued';

  createdAt: Date;
  updatedAt: Date;
}

interface Action {
  id: string;
  description: string;
  priority: number;              // 1=highest
  completed: boolean;
  completedAt?: Date;
  completedBy?: string;
  notes?: string;
}

interface Prescription {
  id: string;
  medication: string;
  genericName: string;
  dosage: string;
  route: string;                 // PO, IV, IM, etc.
  frequency: string;
  duration: string;
  quantity: number;
  refills: number;
  instructions: string;

  // Safety
  drugInteractions: DrugInteraction[];
  contraindications: string[];

  status: 'active' | 'discontinued' | 'completed';
  prescribedAt: Date;
}

interface DrugInteraction {
  drug1: string;
  drug2: string;
  severity: 'major' | 'moderate' | 'minor';
  description: string;
  recommendation: string;
}
```

6. Clinical Note

```
interface ClinicalNote {
  id: string;
  visitId: string;
  patientId: string;
  doctorId: string;

  // SOAP Format
  subjective: string;
  objective: string;
  assessment: string;
  plan: string;

  // Full Note
  fullNote: string;

  // Metadata
  format: 'SOAP' | 'APSO' | 'DAP' | 'free-text';
  autoGenerated: boolean;
  editedByDoctor: boolean;

  // Status
  status: 'draft' | 'signed' | 'amended' | 'addendum';
  signedAt?: Date;

  // Integration
  sentToEMR: boolean;
  emrId?: string;
  emrSystem?: string;

  createdAt: Date;
  updatedAt: Date;
}
```

7. Doctor/User

```
interface Doctor {
  id: string;

  // Personal Info
  firstName: string;
  lastName: string;
  email: string;
  phone: string;

  // Professional Info
  medicalLicenseNumber: string;
  licenseState: string;
  licenseExpiryDate: Date;
  specialty: string;
  subSpecialties: string[];

  // Credentials
  medicalSchool: string;
  graduationYear: number;
  boardCertifications: Certification[];

  // Settings
  preferences: DoctorPreferences;

  // Status
  accountStatus: 'active' | 'suspended' | 'inactive';
  emailVerified: boolean;

  createdAt: Date;
  updatedAt: Date;
  lastLogin: Date;
}

interface DoctorPreferences {
  // General
  language: string;
  voiceInputEnabled: boolean;
  autoSaveNotes: boolean;
  offlineModeEnabled: boolean;

  // Clinical
  specialtyFocus: string;
  aiConfidenceThreshold: number;  // 0-100
  showICD10Codes: boolean;
  autoOrderLabs: boolean;
  showEvidenceCitations: boolean;

  // Display
  theme: 'light' | 'dark' | 'auto';
  textSize: 'small' | 'medium' | 'large';
  colorCodedUrgency: boolean;
  compactView: boolean;
  animationsEnabled: boolean;

  // Notifications
  criticalLabAlerts: boolean;
  followUpReminders: boolean;
  drugInteractionAlerts: boolean;
  guidelineUpdates: boolean;
  soundEnabled: boolean;

  // Privacy
  biometricLogin: boolean;
  autoLockMinutes: number;
  encryptedBackup: boolean;
  analyticsSharing: boolean;
}

interface Certification {
  name: string;
  issuingBody: string;
  dateIssued: Date;
  expiryDate?: Date;
  certificateNumber: string;
}
```

8. Analytics

```
interface AnalyticsSnapshot {
  id: string;
  doctorId: string;
  period: 'daily' | 'weekly' | 'monthly' | 'yearly';
  startDate: Date;
  endDate: Date;

  // Patient Metrics
  totalPatients: number;
  newPatients: number;
  urgentCases: number;
  averageTimePerPatient: number; // minutes

  // Diagnosis Metrics
  topDiagnoses: DiagnosisCount[];
  diagnosisByCategory: CategoryCount[];

  // Urgency Breakdown
  criticalCount: number;
  moderateCount: number;
  routineCount: number;

  // Outcomes
  dischargedCount: number;
  admittedCount: number;
  referredCount: number;

  // Performance
  accuracyRate: number;          // percentage
  averageConfidence: number;     // percentage

  // Volume
  hourlyVolume?: HourlyCount[];  // for daily
  dailyVolume?: DailyCount[];    // for weekly
  weeklyVolume?: WeeklyCount[];  // for monthly
  monthlyVolume?: MonthlyCount[];// for yearly

  generatedAt: Date;
}

interface DiagnosisCount {
  diagnosisName: string;
  icd10Code: string;
  count: number;
  percentage: number;
}

interface HourlyCount {
  hour: number;                  // 0-23
  count: number;
}
```

Database Schema Relationships

```
Doctor
  ├── 1:N → Patient
  ├── 1:N → Visit
  └── 1:1 → DoctorPreferences

Patient
  ├── 1:N → Visit
  ├── 1:N → LabOrder
  └── 1:N → MedicalHistory

Visit
  ├── N:1 → Patient
  ├── N:1 → Doctor
  ├── 1:N → Diagnosis
  ├── 1:N → LabOrder
  ├── 1:1 → TreatmentPlan
  └── 1:1 → ClinicalNote

Diagnosis
  ├── N:1 → Visit
  ├── 1:N → Reference
  └── 1:1 → TreatmentPlan

LabOrder
  ├── N:1 → Visit
  ├── N:1 → Patient
  └── 0:1 → LabResult

TreatmentPlan
  ├── 1:1 → Visit
  ├── 1:1 → Diagnosis
  ├── 1:N → Action
  ├── 1:N → Prescription
  └── 1:N → Referral

ClinicalNote
  ├── 1:1 → Visit
  ├── N:1 → Patient
  └── N:1 → Doctor
```

Technology Stack

Frontend

Core Technologies

```
Framework: Vanilla JavaScript (ES6+)
UI Library: None (custom components)
Styling: CSS3 with CSS Variables
Build Tool: Vite
Package Manager: npm
```

Mobile/Desktop

```
iOS/Android: Capacitor 5.x
PWA: Service Workers, IndexedDB
Native Features:
  - Biometric authentication (@capacitor/biometric-auth)
  - Camera access (@capacitor/camera)
  - Local notifications (@capacitor/local-notifications)
  - Network status (@capacitor/network)
  - Haptics (@capacitor/haptics)
```

State Management

```
// Simple reactive state pattern
class AppState {
  constructor() {
    this.state = {
      currentPatient: null,
      currentTab: 'overview',
      patients: [],
      isOffline: false,
      syncQueue: []
    };
    this.listeners = {};
  }

  setState(key, value) {
    this.state[key] = value;
    this.notify(key, value);
  }

  subscribe(key, callback) {
    if (!this.listeners[key]) {
      this.listeners[key] = [];
    }
    this.listeners[key].push(callback);
  }

  notify(key, value) {
    if (this.listeners[key]) {
      this.listeners[key].forEach(cb => cb(value));
    }
  }
}
```

Offline Storage

```
// IndexedDB wrapper for offline-first architecture
class OfflineDB {
  constructor() {
    this.db = null;
    this.dbName = 'DiagnosisAI';
    this.version = 1;
  }

  async init() {
    return new Promise((resolve, reject) => {
      const request = indexedDB.open(this.dbName, this.version);

      request.onerror = () => reject(request.error);
      request.onsuccess = () => {
        this.db = request.result;
        resolve(this.db);
      };

      request.onupgradeneeded = (event) => {
        const db = event.target.result;

        // Object stores
        if (!db.objectStoreNames.contains('patients')) {
          db.createObjectStore('patients', { keyPath: 'id' });
        }
        if (!db.objectStoreNames.contains('visits')) {
          const visitStore = db.createObjectStore('visits', { keyPath: 'id' });
          visitStore.createIndex('patientId', 'patientId', { unique: false });
          visitStore.createIndex('timestamp', 'timestamp', { unique: false });
        }
        if (!db.objectStoreNames.contains('syncQueue')) {
          db.createObjectStore('syncQueue', { keyPath: 'id', autoIncrement: true });
        }
        if (!db.objectStoreNames.contains('cache')) {
          db.createObjectStore('cache', { keyPath: 'key' });
        }
      };
    });
  }

  async put(storeName, data) {
    const transaction = this.db.transaction([storeName], 'readwrite');
    const store = transaction.objectStore(storeName);
    return store.put(data);
  }

  async get(storeName, key) {
    const transaction = this.db.transaction([storeName], 'readonly');
    const store = transaction.objectStore(storeName);
    return new Promise((resolve, reject) => {
      const request = store.get(key);
      request.onsuccess = () => resolve(request.result);
      request.onerror = () => reject(request.error);
    });
  }

  async getAll(storeName) {
    const transaction = this.db.transaction([storeName], 'readonly');
    const store = transaction.objectStore(storeName);
    return new Promise((resolve, reject) => {
      const request = store.getAll();
      request.onsuccess = () => resolve(request.result);
      request.onerror = () => reject(request.error);
    });
  }
}
```

Backend

Core Services

```
Language: Node.js 20.x (TypeScript)
Framework: Express.js / Fastify
API Style: REST + GraphQL
Authentication: JWT + OAuth 2.0
Authorization: RBAC (Role-Based Access Control)
```

Microservices Architecture

```
Patient Service:
  - Patient CRUD operations
  - Medical history management
  - Search and filtering

Diagnosis Service:
  - AI-powered diagnosis generation
  - Differential diagnosis ranking
  - Evidence retrieval

Lab Service:
  - Lab order management
  - Result interpretation
  - Integration with lab systems

Treatment Service:
  - Protocol recommendations
  - Medication management
  - Drug interaction checking

Notes Service:
  - Clinical note generation
  - SOAP format templating
  - EMR integration

Analytics Service:
  - Real-time metrics calculation
  - Historical trend analysis
  - Report generation

Sync Service:
  - Offline data synchronization
  - Conflict resolution
  - Queue management
```

API Gateway Configuration

```javascript
// Example API Gateway setup
import express from 'express';
import { createProxyMiddleware } from 'http-proxy-middleware';
import rateLimit from 'express-rate-limit';
import helmet from 'helmet';

const app = express();

// Security
app.use(helmet());
app.use(express.json({ limit: '10mb' }));

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // limit each IP to 100 requests per windowMs
});
app.use('/api/', limiter);

// Service routing
app.use('/api/patients', createProxyMiddleware({
  target: 'http://patient-service:3001',
  changeOrigin: true
}));

app.use('/api/diagnosis', createProxyMiddleware({
  target: 'http://diagnosis-service:3002',
  changeOrigin: true
}));

app.use('/api/labs', createProxyMiddleware({
  target: 'http://lab-service:3003',
  changeOrigin: true
}));

// Health check
app.get('/health', (req, res) => {
  res.json({ status: 'healthy', timestamp: new Date().toISOString() });
});
```

Database

Primary Database: PostgreSQL

```
Version: 15.x
Configuration:
  - Replication: Primary-replica setup
  - Backup: Continuous archiving with PITR
  - Partitioning: By date for visits/analytics
  - Indexing: B-tree, GiST for full-text search

Schema:
  - doctors
  - patients
  - visits
  - diagnoses
  - lab_orders
  - lab_results
  - treatment_plans
  - prescriptions
  - clinical_notes
  - audit_logs
```

Cache: Redis

```
Version: 7.x
Use Cases:
  - Session storage
  - Frequently accessed patient data
  - API response caching
  - Rate limiting counters
  - Real-time analytics aggregation

Configuration:
  - Persistence: AOF (Append-Only File)
  - TTL: Varies by data type
  - Clustering: Redis Cluster for HA
```

Vector Database: Pinecone

```
Use Cases:
  - Medical literature embeddings
  - Semantic search over symptoms
  - Similar case retrieval
  - Drug interaction matching

Index Configuration:
  - Dimensions: 1536 (for OpenAI embeddings)
  - Metric: Cosine similarity
  - Pods: Production-grade pods for low latency
```

Analytics: BigQuery

```
Use Cases:
  - Historical trend analysis
  - Cohort analysis
  - Machine learning feature engineering
  - Business intelligence reporting

Tables:
  - fact_visits
  - fact_diagnoses
  - dim_patients
  - dim_doctors
  - dim_time
```

Infrastructure

Cloud Provider: AWS / Google Cloud

```
Compute:
  - ECS/GKE for containerized services
  - Lambda/Cloud Functions for event-driven tasks
  - EC2/Compute Engine for legacy components

Storage:
  - S3/Cloud Storage for medical images, documents
  - EBS/Persistent Disk for database volumes

Networking:
  - VPC with private subnets for databases
  - ALB/Cloud Load Balancer for traffic distribution
  - CloudFront/Cloud CDN for static assets

Security:
  - IAM for access control
  - KMS for encryption at rest
  - Certificate Manager for SSL/TLS
  - WAF for application firewall
```

Containerization: Docker

```
# Example Dockerfile for Diagnosis Service
FROM node:20-alpine AS builder

WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

COPY . .
RUN npm run build

FROM node:20-alpine

WORKDIR /app
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/package.json ./

USER node
EXPOSE 3000

CMD ["node", "dist/index.js"]
```

Orchestration: Kubernetes

```yaml
# Example Kubernetes deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: diagnosis-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: diagnosis-service
  template:
    metadata:
      labels:
        app: diagnosis-service
    spec:
      containers:
      - name: diagnosis-service
        image: diagnosisai/diagnosis-service:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: url
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 3000
          initialDelaySeconds: 10
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: diagnosis-service
spec:
  selector:
    app: diagnosis-service
  ports:
  - port: 80
    targetPort: 3000
  type: ClusterIP
```

AI/ML Integration

Primary AI Engine: Claude (Anthropic)

Model Selection

```
Model: claude-sonnet-4-20250514
Use Cases:
  - Symptom analysis and interpretation
  - Differential diagnosis generation
  - Lab result interpretation
  - Treatment protocol recommendations
  - Clinical note generation

API Configuration:
  - Max Tokens: 4096
  - Temperature: 0.3 (deterministic medical advice)
  - Top P: 0.9
  - Streaming: Enabled for real-time responses
```

Prompt Engineering

Symptom Analysis Prompt Template:

```
const SYMPTOM_ANALYSIS_PROMPT = `
You are an expert medical AI assistant helping doctors with differential diagnosis.

Patient Information:
- Age: {{age}}
- Gender: {{gender}}
- Chief Complaint: {{chiefComplaint}}

Presenting Symptoms:
{{symptoms}}

Vital Signs:
{{vitalSigns}}

Medical History:
{{medicalHistory}}

Current Medications:
{{medications}}

Task: Provide a ranked differential diagnosis with:
1. Top 5 most likely diagnoses with confidence percentages
2. ICD-10 codes for each diagnosis
3. Supporting and contradicting evidence for each
4. Recommended diagnostic workup
5. Red flags or time-critical considerations

Format your response as JSON:
{
  "differentials": [
    {
      "diagnosis": "string",
      "icd10": "string",
      "confidence": number,
      "urgency": "critical|moderate|routine",
      "supporting": ["string"],
      "contradicting": ["string"],
      "reasoning": "string"
    }
  ],
  "recommendedTests": ["string"],
  "redFlags": ["string"],
  "criticalActions": ["string"]
}
`;
```

Lab Interpretation Prompt Template:

```
const LAB_INTERPRETATION_PROMPT = `
You are a medical AI assistant specializing in laboratory medicine.

Patient Context:
- Age: {{age}}
- Gender: {{gender}}
- Suspected Diagnoses: {{suspectedDiagnoses}}

Lab Results:
{{labResults}}

Reference Ranges:
{{referenceRanges}}

Task: Interpret these lab results:
1. Identify abnormal values with clinical significance
2. Explain what each abnormality might indicate
3. Suggest additional tests if needed
4. Provide clinical correlation with suspected diagnoses

Format as JSON:
{
  "interpretations": [
    {
      "labName": "string",
      "value": "string",
      "isAbnormal": boolean,
      "severity": "critical|moderate|mild",
      "interpretation": "string",
      "clinicalSignificance": "string"
    }
  ],
  "overallAssessment": "string",
  "recommendedFollowUp": ["string"]
}
`;
```

Clinical Note Generation Prompt:

```
const CLINICAL_NOTE_PROMPT = `
You are a medical scribe AI. Generate a professional clinical note in SOAP format.

Visit Data:
- Patient: {{patientName}}, {{age}} year old {{gender}}
- Chief Complaint: {{chiefComplaint}}
- HPI: {{hpi}}
- Vitals: {{vitals}}
- Physical Exam: {{physicalExam}}
- Diagnosis: {{diagnosis}}
- Treatment Plan: {{treatmentPlan}}

Generate a complete SOAP note:

SUBJECTIVE:
[Patient's narrative in third person]

OBJECTIVE:
[Vital signs, physical exam findings, lab results]

ASSESSMENT:
[Diagnosis with ICD-10 codes and clinical reasoning]

PLAN:
[Detailed treatment plan, medications, follow-up]

Use professional medical terminology and proper formatting.
`;
```

Fine-Tuning Strategy

Domain-Specific Models

```
Cardiology Model:
  - Base Model: Claude Sonnet 4
  - Training Data: 50K+ cardiology cases
  - Specialization: MI, heart failure, arrhythmias
  - Accuracy Target: 96%+

Emergency Medicine Model:
  - Base Model: Claude Sonnet 4
  - Training Data: 30K+ ER cases
  - Specialization: Trauma, acute care
  - Accuracy Target: 94%+

Pediatrics Model:
  - Base Model: Claude Sonnet 4
  - Training Data: 40K+ pediatric cases
  - Specialization: Age-specific conditions
  - Accuracy Target: 95%+
```

Medical Knowledge Base

PubMed Integration

```
class PubMedService {
  async searchLiterature(query: string, maxResults: number = 10) {
    const response = await fetch(
      `https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?` +
      `db=pubmed&term=${encodeURIComponent(query)}&retmax=${maxResults}&retmode=json`
    );

    const data = await response.json();
    const pmids = data.esearchresult.idlist;

    // Fetch article details
    const articles = await this.fetchArticleDetails(pmids);
    return articles;
  }

  async fetchArticleDetails(pmids: string[]) {
    const response = await fetch(
      `https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?` +
      `db=pubmed&id=${pmids.join(',')}&retmode=json`
    );

    const data = await response.json();
    return this.parseArticles(data.result);
  }

  async getEvidenceForDiagnosis(diagnosis: string, icd10: string) {
    const query = `${diagnosis} diagnosis treatment`;
    const articles = await this.searchLiterature(query);

    // Extract relevant information
    return {
      totalResults: articles.length,
      recentGuidelines: articles.filter(a => a.type === 'guideline'),
      clinicalTrials: articles.filter(a => a.type === 'clinical_trial'),
      systematicReviews: articles.filter(a => a.type === 'systematic_review')
    };
  }
}
```

Clinical Guidelines Database

```
Sources:
  - American Heart Association (AHA)
  - American College of Cardiology (ACC)
  - Centers for Disease Control (CDC)
  - World Health Organization (WHO)
  - National Institute of Health (NIH)
  - Infectious Diseases Society of America (IDSA)

Update Schedule:
  - Frequency: Weekly automated sync
  - Manual Review: Monthly by medical team
  - Version Control: Git-based changelog

Storage:
  - Format: JSON with metadata
  - Indexing: Full-text search in Elasticsearch
  - Caching: Redis for frequently accessed guidelines
```

Vector Search Implementation

```
import { PineconeClient } from '@pinecone-database/pinecone';
import { OpenAI } from 'openai';

class MedicalKnowledgeSearch {
  private pinecone: PineconeClient;
  private openai: OpenAI;
  private index: any;

  constructor() {
    this.pinecone = new PineconeClient();
    this.openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });
    this.initialize();
  }

  async initialize() {
    await this.pinecone.init({
      apiKey: process.env.PINECONE_API_KEY,
      environment: process.env.PINECONE_ENVIRONMENT
    });
    this.index = this.pinecone.Index('medical-knowledge');
  }

  async generateEmbedding(text: string): Promise<number[]> {
    const response = await this.openai.embeddings.create({
      model: 'text-embedding-3-small',
      input: text
    });
    return response.data[0].embedding;
  }

  async findSimilarCases(symptoms: string[], limit: number = 10) {
    // Create query vector
    const queryText = symptoms.join(' ');
    const queryVector = await this.generateEmbedding(queryText);

    // Search Pinecone
    const results = await this.index.query({
      queryRequest: {
        vector: queryVector,
        topK: limit,
        includeMetadata: true
      }
    });
    // Return similar cases with metadata
    return results.matches.map(match => ({
      caseId: match.id,
      similarity: match.score,
      diagnosis: match.metadata.diagnosis,
      symptoms: match.metadata.symptoms,
      outcome: match.metadata.outcome
    }));
  }

  async semanticSearch(query: string, category?: string) {
    const queryVector = await this.generateEmbedding(query);

    const filter = category ? { category } : undefined;

    const results = await this.index.query({
      queryRequest: {
        vector: queryVector,
        topK: 20,
        filter,
        includeMetadata: true
      }
    });

    return results.matches;
  }
}
```

Drug Interaction Checker

```
interface DrugInteractionDatabase {
  interactions: Map<string, DrugInteraction[]>;
}

class DrugInteractionChecker {
  private database: DrugInteractionDatabase;

  async checkInteractions(medications: string[]): Promise<DrugInteraction[]> {
    const interactions: DrugInteraction[] = [];

    // Check all pairs of medications
    for (let i = 0; i < medications.length; i++) {
      for (let j = i + 1; j < medications.length; j++) {
        const interaction = await this.getInteraction(
          medications[i],
          medications[j]
        );
        if (interaction) {
          interactions.push(interaction);
        }
      }
    }

    // Sort by severity
    return interactions.sort((a, b) => {
      const severityOrder = { major: 0, moderate: 1, minor: 2 };
      return severityOrder[a.severity] - severityOrder[b.severity];
    });
  }

  private async getInteraction(drug1: string, drug2: string) {
    // Check local database first
    const key = `${drug1}-${drug2}`;
    if (this.database.interactions.has(key)) {
      return this.database.interactions.get(key);
    }

    // If not found, query external API
    const response = await fetch(
      `https://rxnav.nlm.nih.gov/REST/interaction/list.json?rxcuis=${drug1}+${drug2}`
    );
    const data = await response.json();

    // Parse and cache result
    const interaction = this.parseInteractionData(data);
    this.database.interactions.set(key, interaction);

    return interaction;
  }
}
```

Security & Compliance

HIPAA Compliance

Technical Safeguards

```
Encryption:
  At Rest:
    - Database: AES-256
    - File Storage: AES-256
    - Backups: Encrypted with separate key
  In Transit:
    - TLS 1.3 for all API communications
    - Certificate pinning on mobile apps

Access Control:
  Authentication:
    - Multi-factor authentication (MFA) required
    - Biometric authentication on mobile
    - Session timeout: 15 minutes
  Authorization:
    - Role-based access control (RBAC)
    - Principle of least privilege
    - Audit logging for all access

Audit Logging:
  Events Logged:
    - User authentication (success/failure)
    - Data access (read/write/delete)
    - Configuration changes
    - Export operations
  Retention: 7 years
  Storage: Immutable append-only logs
```

Administrative Safeguards

```
Policies:
  - Data retention and destruction policy
  - Incident response plan
  - Disaster recovery plan
  - Breach notification procedures

Training:
  - Annual HIPAA training for all staff
  - Security awareness training
  - Role-specific training

Risk Management:
  - Annual risk assessment
  - Vulnerability scanning
  - Penetration testing
  - Security audits
```

Physical Safeguards

```
Data Centers:
  - SOC 2 Type II certified
  - 24/7 security monitoring
  - Biometric access controls
  - Environmental controls

Device Security:
  - Mobile device management (MDM)
  - Remote wipe capability
  - Encryption required
  - Screen lock enforced
```

Authentication & Authorization

JWT Token Structure

```
interface JWTPayload {
  // Standard claims
  sub: string;                    // User ID
  iss: string;                    // Issuer (DiagnosisAI)
  iat: number;                    // Issued at
  exp: number;                    // Expiration

  // Custom claims
  role: 'doctor' | 'admin' | 'staff';
  permissions: string[];
  organizationId: string;
  licenseNumber: string;

  // Session tracking
  sessionId: string;
  deviceId: string;
}

// Token generation
function generateAccessToken(user: Doctor): string {
  const payload: JWTPayload = {
    sub: user.id,
    iss: 'DiagnosisAI',
    iat: Math.floor(Date.now() / 1000),
    exp: Math.floor(Date.now() / 1000) + (15 * 60), // 15 minutes
    role: 'doctor',
    permissions: user.permissions,
    organizationId: user.organizationId,
    licenseNumber: user.medicalLicenseNumber,
    sessionId: generateSessionId(),
    deviceId: user.deviceId
  };

  return jwt.sign(payload, process.env.JWT_SECRET, {
    algorithm: 'RS256'
  });
}

// Refresh token (longer lived)
function generateRefreshToken(userId: string): string {
  return jwt.sign(
    { sub: userId, type: 'refresh' },
    process.env.REFRESH_TOKEN_SECRET,
    { expiresIn: '7d', algorithm: 'RS256' }
    );
}
```

Role-Based Access Control

```
enum Permission {
  // Patient permissions
  PATIENT_READ = 'patient:read',
  PATIENT_WRITE = 'patient:write',
  PATIENT_DELETE = 'patient:delete',

  // Diagnosis permissions
  DIAGNOSIS_CREATE = 'diagnosis:create',
  DIAGNOSIS_READ = 'diagnosis:read',
  DIAGNOSIS_MODIFY = 'diagnosis:modify',

  // Lab permissions
  LAB_ORDER = 'lab:order',
  LAB_READ = 'lab:read',
  LAB_VERIFY = 'lab:verify',

  // Treatment permissions
  TREATMENT_PRESCRIBE = 'treatment:prescribe',
  TREATMENT_MODIFY = 'treatment:modify',

  // Admin permissions
  USER_MANAGE = 'user:manage',
  SETTINGS_MANAGE = 'settings:manage',
  ANALYTICS_VIEW = 'analytics:view'
}

const ROLE_PERMISSIONS = {
  doctor: [
    Permission.PATIENT_READ,
    Permission.PATIENT_WRITE,
    Permission.DIAGNOSIS_CREATE,
    Permission.DIAGNOSIS_READ,
    Permission.DIAGNOSIS_MODIFY,
    Permission.LAB_ORDER,
    Permission.LAB_READ,
    Permission.TREATMENT_PRESCRIBE,
    Permission.TREATMENT_MODIFY,
    Permission.ANALYTICS_VIEW
  ],
  nurse: [
    Permission.PATIENT_READ,
    Permission.LAB_READ,
    Permission.DIAGNOSIS_READ
  ],
  admin: Object.values(Permission)
};

// Middleware to check permissions
function requirePermission(permission: Permission) {
  return (req: Request, res: Response, next: NextFunction) => {
    const user = req.user as JWTPayload;

    if (!user.permissions.includes(permission)) {
      return res.status(403).json({
        error: 'Forbidden',
        message: `Required permission: ${permission}`
      });
    }

    next();
  };
}
```

Data Privacy

PHI Handling

```
// Pseudonymization for analytics
class PHIProtection {
  private keyVault: Map<string, string>;

  constructor() {
    this.keyVault = new Map();
  }

  // One-way hash for analytics
  pseudonymize(patientId: string): string {
    const hash = crypto
      .createHmac('sha256', process.env.ANALYTICS_SALT)
      .update(patientId)
      .digest('hex');
    return hash;
  }

  // Encryption for storage
  encrypt(data: string): string {
    const iv = crypto.randomBytes(16);
    const cipher = crypto.createCipheriv(
      'aes-256-gcm',
      Buffer.from(process.env.ENCRYPTION_KEY, 'hex'),
      iv
    );

    let encrypted = cipher.update(data, 'utf8', 'hex');
    encrypted += cipher.final('hex');
    const authTag = cipher.getAuthTag();

    return `${iv.toString('hex')}:${authTag.toString('hex')}:${encrypted}`;
  }

  // Decryption
  decrypt(encryptedData: string): string {
    const parts = encryptedData.split(':');
    const iv = Buffer.from(parts[0], 'hex');
    const authTag = Buffer.from(parts[1], 'hex');
    const encrypted = parts[2];

    const decipher = crypto.createDecipheriv(
      'aes-256-gcm',
      Buffer.from(process.env.ENCRYPTION_KEY, 'hex'),
      iv
    );
    decipher.setAuthTag(authTag);

    let decrypted = decipher.update(encrypted, 'hex', 'utf8');
    decrypted += decipher.final('utf8');

    return decrypted;
  }

  // Data minimization for exports
  redactPHI(data: any): any {
    const redacted = { ...data };

    // Remove direct identifiers
    delete redacted.firstName;
    delete redacted.lastName;
    delete redacted.ssn;
    delete redacted.email;
    delete redacted.phone;
    delete redacted.address;

    // Keep only essential clinical data
    return {
      id: this.pseudonymize(data.id),
      age: data.age,
      gender: data.gender,
      diagnosis: data.diagnosis,
      treatmentOutcome: data.treatmentOutcome
    };
  }
}
```

Audit Logging

```
interface AuditLog {
  id: string;
  timestamp: Date;
  userId: string;
  action: AuditAction;
  resourceType: string;
  resourceId: string;
  changes?: any;
  ipAddress: string;
  userAgent: string;
  result: 'success' | 'failure';
  errorMessage?: string;
}

enum AuditAction {
  CREATE = 'CREATE',
  READ = 'READ',
  UPDATE = 'UPDATE',
  DELETE = 'DELETE',
  EXPORT = 'EXPORT',
  LOGIN = 'LOGIN',
  LOGOUT = 'LOGOUT',
  FAILED_LOGIN = 'FAILED_LOGIN'
}

class AuditService {
  async log(entry: Omit<AuditLog, 'id' | 'timestamp'>) {
    const auditEntry: AuditLog = {
      id: uuidv4(),
      timestamp: new Date(),
      ...entry
    };

    // Write to immutable audit log
    await this.writeToAuditLog(auditEntry);

    // Alert on sensitive actions
    if (this.isSensitiveAction(entry.action)) {
      await this.sendSecurityAlert(auditEntry);
    }
  }

  private isSensitiveAction(action: AuditAction): boolean {
    return [
      AuditAction.DELETE,
      AuditAction.EXPORT,
      AuditAction.FAILED_LOGIN
    ].includes(action);
  }

  async queryAuditLog(filters: {
    userId?: string;
    resourceType?: string;
    startDate?: Date;
    endDate?: Date;
    action?: AuditAction;
  }) {
    // Query audit logs (typically from separate database)
    return await this.auditLogRepository.find(filters);
  }
}

// Middleware to automatically log all requests
function auditMiddleware(req: Request, res: Response, next: NextFunction) {
  const originalJson = res.json;

  res.json = function(data) {
    // Log after response
    auditService.log({
      userId: req.user?.sub,
      action: mapHttpMethodToAction(req.method),
      resourceType: extractResourceType(req.path),
      resourceId: req.params.id,
      ipAddress: req.ip,
      userAgent: req.get('user-agent'),
      result: res.statusCode < 400 ? 'success' : 'failure'
    });

    return originalJson.call(this, data);
  };

  next();
}
```

API Design

RESTful Endpoints

Patient Management

```
POST   /api/v1/patients                    Create new patient
GET    /api/v1/patients                    List all patients (paginated)
GET    /api/v1/patients/:id                Get patient by ID
PUT    /api/v1/patients/:id                Update patient
DELETE /api/v1/patients/:id                Delete patient (soft delete)
GET    /api/v1/patients/search             Search patients
```

Visit/Encounter Management

```
POST   /api/v1/visits                      Create new visit
GET    /api/v1/visits/:id                  Get visit by ID
PUT    /api/v1/visits/:id                  Update visit
GET    /api/v1/patients/:id/visits         Get all visits for patient
POST   /api/v1/visits/:id/complete         Mark visit as complete
```

Diagnosis

```
POST   /api/v1/visits/:id/diagnosis        Generate differential diagnosis
GET    /api/v1/visits/:id/diagnosis        Get diagnoses for visit
PUT    /api/v1/diagnosis/:id               Update diagnosis
POST   /api/v1/diagnosis/analyze           Analyze symptoms (AI)
```

Lab Orders & Results

```
POST   /api/v1/visits/:id/labs             Order labs
GET    /api/v1/visits/:id/labs             Get lab orders for visit
PUT    /api/v1/labs/:id/results            Update lab results
POST   /api/v1/labs/interpret              AI interpretation of lab results
GET    /api/v1/labs/:id                    Get lab order by ID
```

Treatment

```
POST   /api/v1/visits/:id/treatment        Create treatment plan
GET    /api/v1/visits/:id/treatment        Get treatment plan
PUT    /api/v1/treatment/:id               Update treatment plan
POST   /api/v1/treatment/protocols         Get protocol recommendations
POST   /api/v1/treatment/drug-check        Check drug interactions
```

Clinical Notes

```
POST   /api/v1/visits/:id/notes            Generate clinical note
GET    /api/v1/visits/:id/notes            Get clinical note
PUT    /api/v1/notes/:id                   Update note
POST   /api/v1/notes/:id/sign              Sign note
POST   /api/v1/notes/:id/send-to-emr       Send to EMR system
```

Analytics

```
GET    /api/v1/analytics/daily             Daily analytics
GET    /api/v1/analytics/weekly            Weekly analytics
GET    /api/v1/analytics/monthly           Monthly analytics
GET    /api/v1/analytics/yearly            Yearly analytics
GET    /api/v1/analytics/custom            Custom date range
POST   /api/v1/analytics/export            Export analytics report
```

Settings & Preferences

```
GET    /api/v1/users/me/preferences        Get user preferences
PUT    /api/v1/users/me/preferences        Update preferences
POST   /api/v1/settings/sync               Sync settings
```

API Request/Response Examples

Create Visit with Diagnosis

```
POST /api/v1/visits
Content-Type: application/json
Authorization: Bearer <token>

{
  "patientId": "550e8400-e29b-41d4-a716-446655440000",
  "chiefComplaint": "Chest pain radiating to left arm",
  "symptoms": [
    {
      "name": "Chest pain",
      "onset": "2024-12-28T14:00:00Z",
      "duration": "30 minutes",
      "severity": 5,
      "location": "substernal",
      "qualityDescriptors": ["pressure", "squeezing"],
      "alleviatingFactors": ["none"],
      "exacerbatingFactors": ["exertion"],
      "associatedSymptoms": ["diaphoresis", "shortness of breath"]
    }
  ],
  "vitalSigns": {
    "bloodPressure": { "systolic": 160, "diastolic": 95 },
    "heartRate": 102,
    "respiratoryRate": 22,
    "temperature": 98.2,
    "oxygenSaturation": 96
  }
}
```

Response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "id": "7c9e6679-7425-40de-944b-e07fc1f90ae7",
  "patientId": "550e8400-e29b-41d4-a716-446655440000",
  "timestamp": "2024-12-28T14:30:00Z",
  "chiefComplaint": "Chest pain radiating to left arm",
  "urgencyLevel": "critical",
  "status": "active",
  "differentialDiagnoses": [
    {
      "id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
      "name": "Myocardial Infarction",
      "icd10Code": "I21.9",
      "confidenceScore": 78,
      "urgencyLevel": "critical",
      "supportingFindings": [
        "Substernal chest pain with radiation",
        "Diaphoresis present",
        "Elevated blood pressure and heart rate",
        "Male 45+",
        "Acute onset >20 minutes"
      ],
      "evidenceBase": {
        "similarCases": 2847,
        "studiesCount": 47,
        "guidelinesYear": 2023
      }
    },
    {
      "id": "b2c3d4e5-f6g7-8901-bcde-f12345678901",
      "name": "Unstable Angina",
      "icd10Code": "I20.0",
      "confidenceScore": 65,
      "urgencyLevel": "moderate",
      "supportingFindings": [
        "Similar presentation pattern",
        "Urgent evaluation needed"
      ]
    }
  ],
  "createdAt": "2024-12-28T14:30:00Z",
  "updatedAt": "2024-12-28T14:30:00Z"
}
```

Get Analytics

```
GET /api/v1/analytics/daily?date=2024-12-28
Authorization: Bearer <token>
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "period": "daily",
  "date": "2024-12-28",
  "metrics": {
    "totalPatients": 12,
    "newPatients": 3,
    "urgentCases": 3,
    "averageTimePerPatient": 144,
    "completedVisits": 8,
    "activeVisits": 4
  },
  "hourlyVolume": [
    { "hour": 8, "count": 1 },
    { "hour": 9, "count": 2 },
    { "hour": 10, "count": 3 },
    { "hour": 11, "count": 4 },
    { "hour": 12, "count": 5 },
    { "hour": 13, "count": 4 },
    { "hour": 14, "count": 3 },
    { "hour": 15, "count": 2 }
  ],
  "topDiagnoses": [
    { "name": "Myocardial Infarction", "icd10": "I21.9", "count": 2 },
    { "name": "Pneumonia", "icd10": "J18.9", "count": 3 },
    { "name": "GERD", "icd10": "K21.9", "count": 4 }
  ],
  "urgencyBreakdown": {
    "critical": 3,
    "moderate": 5,
    "routine": 4
  },
  "generatedAt": "2024-12-28T15:00:00Z"
}
```

Error Handling

Standard Error Response

```
interface ErrorResponse {
  error: {
    code: string;
    message: string;
    details?: any;
    timestamp: string;
    requestId: string;
  };
}

// Example error responses
const errorResponses = {
  // 400 Bad Request
  VALIDATION_ERROR: {
    status: 400,
    code: 'VALIDATION_ERROR',
    message: 'Request validation failed'
  },

  // 401 Unauthorized
  AUTHENTICATION_REQUIRED: {
    status: 401,
    code: 'AUTHENTICATION_REQUIRED',
    message: 'Authentication token required'
  },

  // 403 Forbidden
  INSUFFICIENT_PERMISSIONS: {
    status: 403,
    code: 'INSUFFICIENT_PERMISSIONS',
    message: 'User does not have required permissions'
  },

  // 404 Not Found
  RESOURCE_NOT_FOUND: {
    status: 404,
    code: 'RESOURCE_NOT_FOUND',
    message: 'Requested resource not found'
  },

  // 409 Conflict
  RESOURCE_CONFLICT: {
    status: 409,
    code: 'RESOURCE_CONFLICT',
    message: 'Resource conflict detected'
  },

  // 429 Too Many Requests
  RATE_LIMIT_EXCEEDED: {
    status: 429,
    code: 'RATE_LIMIT_EXCEEDED',
    message: 'Rate limit exceeded, try again later'
  },

  // 500 Internal Server Error
  INTERNAL_SERVER_ERROR: {
    status: 500,
    code: 'INTERNAL_SERVER_ERROR',
    message: 'An unexpected error occurred'
  }
};
```

Rate Limiting

```
const rateLimitConfig = {
  // General API endpoints
  default: {
    windowMs: 15 * 60 * 1000,  // 15 minutes
    max: 100                     // 100 requests per window
  },

  // AI-powered endpoints (more expensive)
  aiEndpoints: {
    windowMs: 60 * 1000,         // 1 minute
    max: 10                      // 10 requests per minute
  },

  // Authentication endpoints
  authEndpoints: {
    windowMs: 60 * 1000,         // 1 minute
    max: 5                       // 5 attempts per minute
  },

  // Analytics/export endpoints
  heavyEndpoints: {
    windowMs: 60 * 60 * 1000,    // 1 hour
    max: 20                      // 20 requests per hour
  }
};
```

Offline Capabilities

Offline-First Architecture

Service Worker Implementation

```javascript
// service-worker.js
const CACHE_NAME = 'diagnosisai-v2.0.0';
const STATIC_CACHE = 'diagnosisai-static-v2.0.0';
const DATA_CACHE = 'diagnosisai-data-v2.0.0';

// Static assets to cache
const STATIC_ASSETS = [
  '/',
  '/index.html',
  '/styles.css',
  '/app.js',
  '/manifest.json',
  '/icons/icon-192.png',
  '/icons/icon-512.png'
];

// Install event - cache static assets
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(STATIC_CACHE)
      .then((cache) => cache.addAll(STATIC_ASSETS))
      .then(() => self.skipWaiting())
  );
});

// Activate event - clean up old caches
self.addEventListener('activate', (event) => {
  event.waitUntil(
    caches.keys()
      .then((cacheNames) => {
        return Promise.all(
          cacheNames.map((cacheName) => {
            if (cacheName !== STATIC_CACHE && cacheName !== DATA_CACHE) {
              return caches.delete(cacheName);
            }
          })
        );
      })
      .then(() => self.clients.claim())
  );
});

// Fetch event - network first, then cache
self.addEventListener('fetch', (event) => {
  const { request } = event;
  const url = new URL(request.url);

  // API requests - network first, cache fallback
  if (url.pathname.startsWith('/api/')) {
    event.respondWith(
      fetch(request)
        .then((response) => {
          // Clone and cache successful responses
          if (response.ok) {
            const responseClone = response.clone();
            caches.open(DATA_CACHE).then((cache) => {
              cache.put(request, responseClone);
            });
          }
          return response;
        })
        .catch(() => {
          // Network failed, try cache
          return caches.match(request);
        })
    );
  }
  // Static assets - cache first
  else {
    event.respondWith(
      caches.match(request)
        .then((response) => response || fetch(request))
    );
  }
});

// Background sync
self.addEventListener('sync', (event) => {
  if (event.tag === 'sync-pending-data') {
    event.waitUntil(syncPendingData());
  }
});

async function syncPendingData() {
  const db = await openIndexedDB();
  const syncQueue = await db.getAll('syncQueue');

  for (const item of syncQueue) {
    try {
      await fetch(item.url, {
        method: item.method,
        headers: item.headers,
        body: JSON.stringify(item.data)
      });

      // Remove from queue on success
      await db.delete('syncQueue', item.id);
    } catch (error) {
      console.error('Sync failed for item:', item.id, error);
    }
  }
}
```

Sync Strategy

Queue-Based Synchronization

```
interface SyncQueueItem {
  id: string;
  timestamp: Date;
  operation: 'CREATE' | 'UPDATE' | 'DELETE';
  resource: string;
  resourceId: string;
  data: any;
  attempts: number;
  lastAttempt?: Date;
  status: 'pending' | 'in_progress' | 'failed' | 'completed';
}

class SyncManager {
  private db: OfflineDB;
  private maxRetries = 3;
  private retryDelay = 5000; // 5 seconds

  constructor() {
    this.db = new OfflineDB();
    this.startPeriodicSync();
  }

  async queueForSync(item: Omit<SyncQueueItem, 'id' | 'timestamp' | 'attempts' | 'status'>) {
    const queueItem: SyncQueueItem = {
      id: uuidv4(),
      timestamp: new Date(),
      attempts: 0,
      status: 'pending',
      ...item
    };

    await this.db.put('syncQueue', queueItem);

    // Try to sync immediately if online
    if (navigator.onLine) {
      await this.processSyncQueue();
    }
  }

  async processSyncQueue() {
    const queue = await this.db.getAll('syncQueue');
    const pending = queue.filter(item =>
      item.status === 'pending' && item.attempts < this.maxRetries
    );
    for (const item of pending) {
      try {
        // Mark as in progress
        item.status = 'in_progress';
        await this.db.put('syncQueue', item);

        // Execute sync
        await this.syncItem(item);

        // Remove from queue on success
        await this.db.delete('syncQueue', item.id);
      } catch (error) {
        // Update retry info
        item.attempts++;
        item.lastAttempt = new Date();
        item.status = item.attempts >= this.maxRetries ? 'failed' : 'pending';
        await this.db.put('syncQueue', item);

        console.error(`Sync failed for item ${item.id}:`, error);
      }
    }
  }

  private async syncItem(item: SyncQueueItem) {
    const url = `/api/v1/${item.resource}${item.resourceId ? `/${item.resourceId}` : ''}`;
    const method = item.operation === 'CREATE' ? 'POST' :
                   item.operation === 'UPDATE' ? 'PUT' : 'DELETE';

    const response = await fetch(url, {
      method,
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${await this.getAuthToken()}`
      },
      body: item.operation !== 'DELETE' ? JSON.stringify(item.data) : undefined
    });

    if (!response.ok) {
      throw new Error(`Sync failed: ${response.statusText}`);
    }

    return await response.json();
  }
  private startPeriodicSync() {
    // Check for pending syncs every 30 seconds
    setInterval(async () => {
      if (navigator.onLine) {
        await this.processSyncQueue();
      }
    }, 30000);

    // Also sync when coming back online
    window.addEventListener('online', () => {
      this.processSyncQueue();
    });
  }

  async getSyncStatus() {
    const queue = await this.db.getAll('syncQueue');
    return {
      pending: queue.filter(i => i.status === 'pending').length,
      inProgress: queue.filter(i => i.status === 'in_progress').length,
      failed: queue.filter(i => i.status === 'failed').length
    };
  }
}
```

Conflict Resolution

```
interface ConflictResolution {
  strategy: 'last_write_wins' | 'manual' | 'merge';
  resolvedBy?: string;
  resolvedAt?: Date;
}

class ConflictResolver {
  async resolveConflict(
    localVersion: any,
    serverVersion: any,
    strategy: ConflictResolution['strategy'] = 'last_write_wins'
  ) {
    switch (strategy) {
      case 'last_write_wins':
        return this.lastWriteWins(localVersion, serverVersion);

      case 'merge':
        return this.mergeVersions(localVersion, serverVersion);

      case 'manual':
        return this.promptUserResolution(localVersion, serverVersion);

      default:
        throw new Error(`Unknown conflict resolution strategy: ${strategy}`);
    }
  }

  private lastWriteWins(local: any, server: any) {
    const localTime = new Date(local.updatedAt).getTime();
    const serverTime = new Date(server.updatedAt).getTime();

    return serverTime > localTime ? server : local;
  }

  private mergeVersions(local: any, server: any) {
    // Deep merge with preference for non-null server values
    return {
      ...local,
      ...server,
      // Keep whichever is more recent for timestamp fields
      updatedAt: new Date(Math.max(
        new Date(local.updatedAt).getTime(),
        new Date(server.updatedAt).getTime()
      ))
    };
  }

  private async promptUserResolution(local: any, server: any) {
    // Show UI for user to choose
    return new Promise((resolve) => {
      // Implementation would show a modal with both versions
      // and let user choose or merge manually
    });
  }
}
```

Integration Layer

EMR Integration

HL7 FHIR Support

```
import { Client } from 'fhir-kit-client';

class FHIRIntegration {
  private client: any;

  constructor(baseUrl: string) {
    this.client = new Client({ baseUrl });
  }

  // Send patient data to EMR
  async createPatient(patient: Patient) {
    const fhirPatient = this.toFHIRPatient(patient);

    return await this.client.create({
      resourceType: 'Patient',
      body: fhirPatient
    });
  }

  // Send clinical note
  async createDocumentReference(note: ClinicalNote) {
    const documentRef = {
      resourceType: 'DocumentReference',
      status: 'current',
      type: {
        coding: [{
          system: 'http://loinc.org',
          code: '11488-4',
          display: 'Consultation note'
        }]
      },
      subject: {
        reference: `Patient/${note.patientId}`
      },
      author: [{
        reference: `Practitioner/${note.doctorId}`
      }],
      content: [{
        attachment: {
          contentType: 'text/plain',
          data: Buffer.from(note.fullNote).toString('base64')
        }
      }]
    };
    return await this.client.create({
      resourceType: 'DocumentReference',
      body: documentRef
    });
  }

  // Retrieve lab results
  async getLabResults(patientId: string) {
    const response = await this.client.search({
      resourceType: 'Observation',
      searchParams: {
        patient: patientId,
        category: 'laboratory'
      }
    });

    return response.entry.map(entry => this.fromFHIRObservation(entry.resource));
  }

  private toFHIRPatient(patient: Patient) {
    return {
      resourceType: 'Patient',
      identifier: [{
        system: 'http://diagnosisai.com/patient-id',
        value: patient.id
      }],
      name: [{
        family: patient.lastName,
        given: [patient.firstName]
      }],
      gender: patient.gender,
      birthDate: patient.dateOfBirth.toISOString().split('T')[0],
      telecom: [
        {
          system: 'phone',
          value: patient.contactInfo.phone
        },
        {
          system: 'email',
          value: patient.contactInfo.email
        }
      ]
    };
  }
  private fromFHIRObservation(observation: any): LabResult {
    return {
      id: observation.id,
      testName: observation.code.text,
      value: observation.valueQuantity.value,
      unit: observation.valueQuantity.unit,
      referenceRange: {
        low: observation.referenceRange[0].low.value,
        high: observation.referenceRange[0].high.value,
        unit: observation.referenceRange[0].low.unit
      },
      performedAt: new Date(observation.effectiveDateTime),
      // ... additional mapping
    };
  }
}
```

Epic Integration (Proprietary)

```
class EpicIntegration {
  private baseUrl: string;
  private clientId: string;
  private clientSecret: string;

  constructor(config: any) {
    this.baseUrl = config.baseUrl;
    this.clientId = config.clientId;
    this.clientSecret = config.clientSecret;
  }

  async authenticate() {
    const response = await fetch(`${this.baseUrl}/oauth2/token`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      },
      body: new URLSearchParams({
        grant_type: 'client_credentials',
        client_id: this.clientId,
        client_secret: this.clientSecret
      })
    });

    const data = await response.json();
    return data.access_token;
  }

  async sendClinicalNote(note: ClinicalNote) {
    const token = await this.authenticate();

    const response = await fetch(`${this.baseUrl}/api/FHIR/R4/DocumentReference`, {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/fhir+json'
      },
      body: JSON.stringify({
        resourceType: 'DocumentReference',
        // ... Epic-specific formatting
      })
    });

    return await response.json();
    }
}
```

Lab System Integration

HL7 v2.x Messaging

```
import * as hl7 from 'simple-hl7';

class HL7Integration {
  async parseLabResult(hl7Message: string): Promise<LabResult> {
    const parser = new hl7.Parser();
    const message = parser.parse(hl7Message);

    // Extract OBX segments (observation/result)
    const obxSegments = message.segments.filter(s => s.name === 'OBX');

    return obxSegments.map(segment => ({
      testName: segment.fields[3],
      value: segment.fields[5],
      unit: segment.fields[6],
      referenceRange: this.parseReferenceRange(segment.fields[7]),
      abnormalFlag: segment.fields[8],
      observationDateTime: new Date(segment.fields[14])
    }));
  }

  createLabOrder(order: LabOrder): string {
    const message = new hl7.Message();

    // MSH - Message Header
    message.addSegment([
      'MSH',
      '^~\\&',
      'DiagnosisAI',
      'DiagnosisAI',
      'LabSystem',
      'LabSystem',
      this.formatHL7DateTime(new Date()),
      '',
      'ORM^O01',
      this.generateMessageId(),
      'P',
      '2.5'
    ]);

    // PID - Patient Identification
    message.addSegment([
      'PID',
      '1',
      order.patientId,
      // ... additional patient fields
    ]);

    // ORC - Common Order
    message.addSegment([
      'ORC',
      'NW',
      order.id,
      '',
      '',
      '',
      '',
      this.formatHL7DateTime(order.orderedAt),
      // ... additional order fields
    ]);

    // OBR - Observation Request
    message.addSegment([
      'OBR',
      '1',
      order.id,
      '',
      order.loincCode,
      order.priority,
      // ... additional request fields
    ]);

    return message.toString();
  }

  private formatHL7DateTime(date: Date): string {
    return date.toISOString()
      .replace(/[-:]/g, '')
      .replace('T', '')
      .split('.')[0];
  }

  private generateMessageId(): string {
    return `DIA${Date.now()}${Math.random().toString(36).substr(2, 5).toUpperCase()}`;
  }
}
```

Performance Optimization

Frontend Optimization

Code Splitting

```
// Lazy load components
const PatientProfile = () => import('./components/PatientProfile.js');
const Analytics = () => import('./components/Analytics.js');
const Settings = () => import('./components/Settings.js');

// Route-based code splitting
const routes = {
  'home': () => import('./screens/HomeScreen.js'),
  'patient': () => import('./screens/PatientScreen.js'),
  'analytics': () => import('./screens/AnalyticsScreen.js'),
  'settings': () => import('./screens/SettingsScreen.js')
};

async function navigate(route) {
  const module = await routes[route]();
  module.render();
}
```

Image Optimization

```javascript
// Responsive images
function generateResponsiveImage(src, sizes) {
  const srcset = sizes.map(size => {
    const url = `${src}?w=${size}&q=80&fm=webp`;
    return `${url} ${size}w`;
  }).join(', ');

  return {
    src: `${src}?w=${sizes[0]}&q=80&fm=webp`,
    srcset,
    sizes: '(max-width: 768px) 100vw, 50vw'
  };
}

// Lazy loading images
const imageObserver = new IntersectionObserver((entries) => {
  entries.forEach(entry => {
    if (entry.isIntersecting) {
      const img = entry.target;
      img.src = img.dataset.src;
      img.classList.remove('lazy');
      imageObserver.unobserve(img);
    }
  });
});

document.querySelectorAll('img.lazy').forEach(img => {
  imageObserver.observe(img);
});
```

Virtual Scrolling

```
// For large patient lists
class VirtualScroller {
  constructor(container, itemHeight, renderItem) {
    this.container = container;
    this.itemHeight = itemHeight;
    this.renderItem = renderItem;
    this.scrollTop = 0;
    this.containerHeight = container.clientHeight;

    this.container.addEventListener('scroll', () => this.handleScroll());
  }

  setItems(items) {
    this.items = items;
    this.totalHeight = items.length * this.itemHeight;
    this.visibleCount = Math.ceil(this.containerHeight / this.itemHeight) + 1;
    this.render();
  }

  handleScroll() {
    this.scrollTop = this.container.scrollTop;
    this.render();
  }

  render() {
    const startIndex = Math.floor(this.scrollTop / this.itemHeight);
    const endIndex = Math.min(startIndex + this.visibleCount, this.items.length);

    const offsetY = startIndex * this.itemHeight;

    const html = `
      <div style="height: ${this.totalHeight}px; position: relative;">
        <div style="transform: translateY(${offsetY}px);">
          ${this.items.slice(startIndex, endIndex)
            .map(item => this.renderItem(item))
            .join('')}
        </div>
      </div>
    `;

    this.container.innerHTML = html;
  }
}
```

Backend Optimization

Database Indexing

```sql
-- Patient search index
CREATE INDEX idx_patients_name ON patients (first_name, last_name);
CREATE INDEX idx_patients_dob ON patients (date_of_birth);

-- Visit queries
CREATE INDEX idx_visits_patient ON visits (patient_id, timestamp DESC);
CREATE INDEX idx_visits_doctor ON visits (doctor_id, timestamp DESC);
CREATE INDEX idx_visits_status ON visits (status, timestamp DESC);

-- Diagnosis queries
CREATE INDEX idx_diagnoses_visit ON diagnoses (visit_id);
CREATE INDEX idx_diagnoses_icd10 ON diagnoses (icd10_code);

-- Lab results
CREATE INDEX idx_labs_patient ON lab_orders (patient_id, ordered_at DESC);
CREATE INDEX idx_labs_status ON lab_orders (status, priority);

-- Full-text search on symptoms
CREATE INDEX idx_symptoms_fulltext ON visits USING GIN (to_tsvector('english', chief_complaint

-- Analytics queries
CREATE INDEX idx_visits_date ON visits (DATE(timestamp));
CREATE INDEX idx_diagnoses_date_category ON diagnoses (DATE(created_at), icd10_code);
```

Query Optimization

```
// Use connection pooling
import { Pool } from 'pg';

const pool = new Pool({
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  max: 20,                    // Maximum pool size
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000
});

// Prepared statements
const getPatientQuery = {
  name: 'get-patient',
  text: 'SELECT * FROM patients WHERE id = $1',
  values: (id) => [id]
};

// Use transactions for consistency
async function createVisitWithDiagnosis(visitData, diagnosisData) {
  const client = await pool.connect();

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

    const visitResult = await client.query(
      'INSERT INTO visits (patient_id, doctor_id, ...) VALUES ($1, $2, ...) RETURNING *',
      [visitData.patientId, visitData.doctorId, ...]
    );

    const visit = visitResult.rows[0];

    await client.query(
      'INSERT INTO diagnoses (visit_id, name, icd10_code, ...) VALUES ($1, $2, $3, ...)',
      [visit.id, diagnosisData.name, diagnosisData.icd10Code, ...]
    );

    await client.query('COMMIT');
    return visit;
  } catch (error) {
    await client.query('ROLLBACK');
    throw error;
  } finally {
    client.release();
  }
}
```

Caching Strategy

```
import Redis from 'ioredis';

const redis = new Redis({
  host: process.env.REDIS_HOST,
  port: process.env.REDIS_PORT,
  password: process.env.REDIS_PASSWORD,
  retryStrategy: (times) => Math.min(times * 50, 2000)
});

class CacheService {
  // Cache patient data
  async cachePatient(patient: Patient, ttl: number = 3600) {
    const key = `patient:${patient.id}`;
    await redis.setex(key, ttl, JSON.stringify(patient));
  }

  async getCachedPatient(patientId: string): Promise<Patient | null> {
    const key = `patient:${patientId}`;
    const cached = await redis.get(key);
    return cached ? JSON.parse(cached) : null;
  }

  // Cache-aside pattern
  async getPatient(patientId: string): Promise<Patient> {
    // Try cache first
    let patient = await this.getCachedPatient(patientId);

    if (!patient) {
      // Cache miss - fetch from database
      patient = await database.findPatientById(patientId);

      if (patient) {
        // Cache for future requests
        await this.cachePatient(patient);
      }
    }

    return patient;
  }

  // Invalidate cache on update
  async invalidatePatient(patientId: string) {
    await redis.del(`patient:${patientId}`);
  }
  // Cache API responses
  async cacheResponse(key: string, data: any, ttl: number = 300) {
    await redis.setex(key, ttl, JSON.stringify(data));
  }

  async getCachedResponse(key: string) {
    const cached = await redis.get(key);
    return cached ? JSON.parse(cached) : null;
  }
}
```

Background Jobs

```
import Bull from 'bull';

// Create job queues
const analyticsQueue = new Bull('analytics', {
  redis: {
    host: process.env.REDIS_HOST,
    port: process.env.REDIS_PORT
  }
});

const syncQueue = new Bull('sync', {
  redis: {
    host: process.env.REDIS_HOST,
    port: process.env.REDIS_PORT
  }
});

// Process analytics calculation in background
analyticsQueue.process(async (job) => {
  const { doctorId, period, startDate, endDate } = job.data;

  const analytics = await calculateAnalytics(doctorId, period, startDate, endDate);

  // Cache the result
  await cacheService.cacheResponse(
    `analytics:${doctorId}:${period}:${startDate}`,
    analytics,
    3600
  );

  return analytics;
});

// Schedule analytics generation
async function scheduleAnalyticsGeneration(doctorId: string) {
  // Daily analytics
  await analyticsQueue.add(
    { doctorId, period: 'daily', startDate: new Date(), endDate: new Date() },
    { repeat: { cron: '0 0 * * *' } }  // Midnight every day
  );

  // Weekly analytics
  await analyticsQueue.add(
    { doctorId, period: 'weekly', startDate: getWeekStart(), endDate: new Date() },
    { repeat: { cron: '0 0 * * 0' } }  // Sunday midnight
  );
}
```

Deployment Strategy

CI/CD Pipeline

```yaml
# .github/workflows/deploy.yml
name: Deploy DiagnosisAI

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '20'
      - run: npm ci
      - run: npm test
      - run: npm run lint

  build:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: docker build -t diagnosisai/api:${{ github.sha }} .
      - run: docker push diagnosisai/api:${{ github.sha }}

  deploy-staging:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
      - uses: azure/k8s-set-context@v1
        with:
          kubeconfig: ${{ secrets.KUBE_CONFIG_STAGING }}
      - run: kubectl set image deployment/api api=diagnosisai/api:${{ github.sha }}

  deploy-production:
    needs: deploy-staging
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    environment:
      name: production
      url: https://api.diagnosisai.com
    steps:
      - uses: azure/k8s-set-context@v1
        with:
          kubeconfig: ${{ secrets.KUBE_CONFIG_PROD }}
      - run: kubectl set image deployment/api api=diagnosisai/api:${{ github.sha }}
```

Environment Configuration

```yaml
# Staging
staging:
  database:
    host: staging-db.diagnosisai.com
    name: diagnosisai_staging
    replicas: 1
  redis:
    host: staging-redis.diagnosisai.com
  resources:
    api:
      replicas: 2
      cpu: "500m"
      memory: "512Mi"

# Production
production:
  database:
    host: prod-db.diagnosisai.com
    name: diagnosisai_prod
    replicas: 3
  redis:
    host: prod-redis.diagnosisai.com
    cluster: true
  resources:
    api:
      replicas: 5
      cpu: "1000m"
      memory: "2Gi"
      autoscaling:
        minReplicas: 5
        maxReplicas: 20
        targetCPU: 70
```

Blue-Green Deployment

```yaml
# blue deployment (current)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-blue
spec:
  replicas: 5
  selector:
    matchLabels:
      app: api
      version: blue
  template:
    metadata:
      labels:
        app: api
        version: blue
    spec:
      containers:
      - name: api
        image: diagnosisai/api:1.0.0

---
# green deployment (new)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-green
spec:
  replicas: 5
  selector:
    matchLabels:
      app: api
      version: green
  template:
    metadata:
      labels:
        app: api
        version: green
    spec:
      containers:
      - name: api
        image: diagnosisai/api:2.0.0

---
```

```
# Service switches between blue/green
apiVersion: v1
kind: Service
metadata:
  name: api
spec:
  selector:
    app: api
    version: blue  # Switch to 'green' for new deployment
  ports:
  - port: 80
    targetPort: 3000
```

Scalability Plan

Horizontal Scaling

Auto-scaling Rules

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: api-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: api
  minReplicas: 5
  maxReplicas: 50
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  - type: Pods
    pods:
      metric:
        name: http_requests_per_second
      target:
        type: AverageValue
        averageValue: "1000"
```

Database Scaling

Read Replicas

```
# Primary database (read-write)
Primary:
  host: db-primary.diagnosisai.com
  role: master
  connections: 100

# Read replicas (read-only)
Replicas:
  - host: db-replica-1.diagnosisai.com
    role: replica
    lag_threshold: 100ms
  - host: db-replica-2.diagnosisai.com
    role: replica
    lag_threshold: 100ms
  - host: db-replica-3.diagnosisai.com
    role: replica
    lag_threshold: 100ms
```

Connection Router

```
class DatabaseRouter {
  private primary: Pool;
  private replicas: Pool[];
  private currentReplica: number = 0;

  async query(sql: string, params: any[]) {
    // Write operations go to primary
    if (this.isWriteOperation(sql)) {
      return await this.primary.query(sql, params);
    }

    // Read operations distributed across replicas
    const replica = this.getNextReplica();
    return await replica.query(sql, params);
  }

  private isWriteOperation(sql: string): boolean {
    const writeKeywords = ['INSERT', 'UPDATE', 'DELETE', 'CREATE', 'ALTER', 'DROP'];
    const upperSQL = sql.trim().toUpperCase();
    return writeKeywords.some(keyword => upperSQL.startsWith(keyword));
  }

  private getNextReplica(): Pool {
    // Round-robin load balancing
    const replica = this.replicas[this.currentReplica];
    this.currentReplica = (this.currentReplica + 1) % this.replicas.length;
    return replica;
  }
}
```

CDN & Edge Caching

```
CloudFront Distribution:
  Origins:
    - API: api.diagnosisai.com
    - Static Assets: s3://diagnosisai-assets

  Behaviors:
    - PathPattern: /api/*
      CachingDisabled: true
      AllowedMethods: [GET, POST, PUT, DELETE, OPTIONS, HEAD, PATCH]

    - PathPattern: /assets/*
      CachingEnabled: true
      TTL: 31536000  # 1 year
      Compression: true

  Edge Locations: All

  Cache Policies:
    static-assets:
      MinTTL: 86400
      MaxTTL: 31536000
      DefaultTTL: 86400

    api-responses:
      MinTTL: 0
      MaxTTL: 300
      DefaultTTL: 0
```

Load Testing Results

```
Target: 10,000 concurrent users

Results:
  Average Response Time: 145ms
  95th Percentile: 280ms
  99th Percentile: 450ms
  Error Rate: 0.02%

  Peak Performance:
    Requests/Second: 50,000
    Database Connections: 450
    CPU Usage: 65%
    Memory Usage: 72%

Bottlenecks Identified:
  - AI API calls (external dependency)
  - Complex analytics queries
  - Full-text search on large datasets

Optimizations Applied:
  - Caching AI responses for similar queries
  - Pre-computed analytics snapshots
  - Elasticsearch for full-text search
```

Future Enhancements

Roadmap

Q1 2026

- Multi-language support (Spanish, Hindi, Mandarin)
- Voice-to-text symptom input (whisper API)
- Telemedicine integration (video consultations)

Q2 2026
- Medical image analysis (X-rays, CT scans)
- Wearable device integration (Apple Watch, Fitbit)
- Predictive analytics (readmission risk, disease progression)

Q3 2026
- Natural language querying of medical literature
- Collaborative diagnosis (multi-doctor consultations)
- Patient portal (view their own records, test results)

Q4 2026
- Global deployment (Europe, Asia, South America)
- Insurance claim automation
- Clinical trial matching

End of Technical Architecture Document