Phase 1: Foundation Setup (Weeks 1-6)

Overview

Phase 1 establishes the core infrastructure and foundational components required for the BioSpark Health AI system. This phase focuses on environment setup, basic security, and initial API framework.

Week-by-Week Implementation Plan

Week 1: Environment Setup

Objectives: Establish development and staging environments

Tasks:
1. **Infrastructure Provisioning**
```bash
# Kubernetes cluster setup
kubectl create namespace biospark-dev
kubectl create namespace biospark-staging

# Install essential operators
helm install ingress-nginx ingress-nginx/ingress-nginx
helm install cert-manager jetstack/cert-manager
```

1. **Database Setup**
   ```sql
   -- PostgreSQL setup
   CREATE DATABASE biospark_dev;
   CREATE DATABASE biospark_staging;
   CREATE USER biospark_user WITH PASSWORD 'secure_password';
   GRANT ALL PRIVILEGES ON DATABASE biospark_dev TO biospark_user;
   ```

2. **Container Registry Configuration**
   ```yaml
   # docker-registry-secret.yaml
   apiVersion: v1
   kind: Secret
   metadata:
     name: docker-registry-secret
   type: kubernetes.io/dockerconfigjson
   data:
     .dockerconfigjson: <base64-encoded-docker-config>
   ```

Deliverables:
- ✅ Kubernetes clusters operational
- ✅ Database instances configured
- ✅ Container registry accessible
- ✅ Basic networking configured

Week 2: Core Infrastructure

Objectives: Deploy foundational services and monitoring

Tasks:

1. **Service Mesh Deployment**

```bash
# Istio installation
istioctl install --set values.defaultRevision=default
kubectl label namespace biospark-dev istio-injection=enabled
```

1. **Monitoring Stack**

```yaml
# prometheus-values.yaml
prometheus:
  prometheusSpec:
    retention: 30d
    storageSpec:
      volumeClaimTemplate:
        spec:
          storageClassName: fast-ssd
          resources:
            requests:
              storage: 100Gi
```

2. **Logging Infrastructure**

```bash
# ELK Stack deployment
helm install elasticsearch elastic/elasticsearch
helm install kibana elastic/kibana
helm install filebeat elastic/filebeat
```

Deliverables:

- ✅ Service mesh operational
- ✅ Monitoring stack deployed
- ✅ Logging infrastructure ready
- ✅ Basic alerting configured

Week 3: Security Foundation

Objectives: Implement core security components

Tasks:

1. **Secrets Management**

```bash
# Vault deployment
helm install vault hashicorp/vault
vault auth enable kubernetes
vault policy write biospark-policy - <<EOF
path "secret/data/biospark/*" {
  capabilities = ["read"]
```

```
  }
  EOF
```

1. **Certificate Management**
   ```yaml
   # cluster-issuer.yaml
   apiVersion: cert-manager.io/v1
   kind: ClusterIssuer
   metadata:
     name: letsencrypt-prod
   spec:
     acme:
       server: https://acme-v02.api.letsencrypt.org/directory
       email: admin@biospark.ai
       privateKeySecretRef:
         name: letsencrypt-prod
   ```

2. **Network Policies**
   ```yaml
   # network-policy.yaml
   apiVersion: networking.k8s.io/v1
   kind: NetworkPolicy
   metadata:
   name: biospark-network-policy
   spec:
   podSelector:
   matchLabels:
   app: biospark
   policyTypes:
   ```
   - Ingress
   - Egress

   ```

**Deliverables**:
- ✅ Secrets management operational
- ✅ TLS certificates automated
- ✅ Network security policies active
- ✅ Basic RBAC configured

## Week 4: API Gateway Setup

**Objectives**: Deploy and configure API gateway

**Tasks**:
1. **Kong Gateway Deployment**

```yaml
  # kong-values.yaml
  image:
    repository: kong
    tag: "3.4"
  env:
    database: postgres
```
```

```
    pg_host: postgresql
    pg_database: kong
  proxy:
    type: LoadBalancer
```

1. **Rate Limiting Configuration**
   ```
   bash
     # Configure rate limiting plugin
     curl -X POST http://kong-admin:8001/plugins \
       --data "name=rate-limiting" \
       --data "config.minute=1000" \
       --data "config.policy=local"
   ```

2. **Authentication Plugin**
   ```
   bash
     # JWT authentication setup
     curl -X POST http://kong-admin:8001/plugins \
       --data "name=jwt" \
       --data "config.secret_is_base64=false"
   ```

Deliverables:
- ✅ API Gateway deployed
- ✅ Rate limiting configured
- ✅ Authentication plugins active
- ✅ Basic routing rules defined

Week 5: Database Schema Implementation

Objectives: Create core database schema and initial data structures

Tasks:
1. **Core Tables Creation**
```sql
– patients table
CREATE TABLE patients (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
first_name VARCHAR(100) NOT NULL,
last_name VARCHAR(100) NOT NULL,
date_of_birth DATE NOT NULL,
email VARCHAR(255) UNIQUE,
phone VARCHAR(20),
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

– health_records table
CREATE TABLE health_records (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
patient_id UUID REFERENCES patients(id),
record_type VARCHAR(50) NOT NULL,
data JSONB NOT NULL,
recorded_at TIMESTAMP NOT NULL,
```

```
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

– agents table
CREATE TABLE agents (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
name VARCHAR(100) NOT NULL,
type VARCHAR(50) NOT NULL,
config JSONB,
status VARCHAR(20) DEFAULT 'inactive',
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

1. **Indexes and Constraints**
   ```sql
   – Performance indexes
   CREATE INDEX idx_patients_email ON patients(email);
   CREATE INDEX idx_health_records_patient_id ON health_records(patient_id);
   CREATE INDEX idx_health_records_recorded_at ON health_records(recorded_at);
   CREATE INDEX idx_agents_type ON agents(type);
   CREATE INDEX idx_agents_status ON agents(status);

– JSONB indexes for health data
CREATE INDEX idx_health_records_data_gin ON health_records USING GIN(data);
```

1. **Initial Data Migration**
   ```sql
   -- Insert default agent configurations
   INSERT INTO agents (name, type, config, status) VALUES
   ('Analyst Agent', 'analyst', '{"version": "1.0", "capabilities": ["data_analysis", "reporting"]}', 'active'),
   ('Architect Agent', 'architect', '{"version": "1.0", "capabilities": ["system_design", "optimization"]}', 'active'),
   ('Developer Agent', 'developer', '{"version": "1.0", "capabilities": ["code_generation", "testing"]}', 'active'),
   ('Orchestrator Agent', 'orchestrator', '{"version": "1.0", "capabilities": ["workflow_management", "coordination"]}', 'active');
   ```

Deliverables:
- ✅ Core database schema implemented
- ✅ Performance indexes created
- ✅ Initial data populated
- ✅ Database migrations framework ready

Week 6: Basic API Framework

Objectives: Implement foundational API endpoints and authentication

Tasks:
1. **Authentication Service**

```typescript
// auth.service.ts
import jwt from 'jsonwebtoken';
import bcrypt from 'bcrypt';

export class AuthService {
async login(email: string, password: string): Promise {
const user = await this.validateUser(email, password);
if (!user) throw new Error('Invalid credentials');

    return jwt.sign(
      { userId: user.id, email: user.email },
      process.env.JWT_SECRET!,
      { expiresIn: '24h' }
    );
  }

  async validateToken(token: string): Promise<any> {
    return jwt.verify(token, process.env.JWT_SECRET!);
  }

}
```

1. **Core API Endpoints**

```typescript
// patients.controller.ts
import { Request, Response } from 'express';
import { PatientService } from '../services/patient.service';

export class PatientsController {
private patientService = new PatientService();

  async getPatients(req: Request, res: Response) {
    const { limit = 50, offset = 0 } = req.query;
    const patients = await this.patientService.findAll({
      limit: Number(limit),
      offset: Number(offset)
    });
    res.json(patients);
  }

  async createPatient(req: Request, res: Response) {
    const patient = await this.patientService.create(req.body);
    res.status(201).json(patient);
  }

}
```

1. **API Documentation**

```yaml
# openapi.yaml
openapi: 3.0.0
info:
```

```
  title: BioSpark Health AI API
  version: 1.0.0
  description: Core API for BioSpark Health AI system
paths:
/api/v1/auth/login:
post:
summary: User authentication
requestBody:
required: true
content:
application/json:
schema:
type: object
properties:
email:
type: string
password:
type: string
responses:
200:
description: Authentication successful
content:
application/json:
schema:
type: object
properties:
token:
type: string
```

Deliverables:
- ✅ Authentication service operational
- ✅ Core API endpoints implemented
- ✅ API documentation generated
- ✅ Basic error handling configured

Phase 1 Quality Gates

Technical Validation

- [ ] All infrastructure components deployed successfully
- [ ] Database schema matches specifications
- [ ] API endpoints respond within 200ms
- [ ] Authentication system functional
- [ ] Basic security measures active

Security Review

- [ ] Secrets properly managed in Vault
- [ ] TLS certificates valid and auto-renewing
- [ ] Network policies restricting traffic

- [ ] RBAC permissions configured correctly
- [ ] No hardcoded credentials in code

Performance Benchmarks

- [ ] API Gateway handling 1000 req/min
- [ ] Database queries under 50ms
- [ ] System startup time under 5 minutes
- [ ] Memory usage within allocated limits
- [ ] CPU utilization under 70%

Documentation Requirements

- [ ] Infrastructure setup documented
- [ ] API documentation complete
- [ ] Security procedures documented
- [ ] Troubleshooting guides available
- [ ] Runbooks for common operations

Troubleshooting Guide

Common Issues

Database Connection Issues

```
# Check database connectivity
kubectl exec -it postgres-pod -- psql -U biospark_user -d biospark_dev -c "SELECT 1;"

# Verify secrets
kubectl get secret postgres-secret -o yaml
```

API Gateway Not Responding

```
# Check Kong status
kubectl get pods -l app=kong
kubectl logs -l app=kong

# Verify service configuration
kubectl get services kong-proxy
```

Certificate Issues

```
# Check certificate status
kubectl get certificates
kubectl describe certificate biospark-tls

# Force certificate renewal
kubectl delete certificate biospark-tls
```

Phase 1 establishes the solid foundation required for successful implementation of subsequent phases.