Enhanced SICA Technical Documentation

Version: 1.0
Date: December 2024
Document Type: Technical Documentation
Classification: Technical

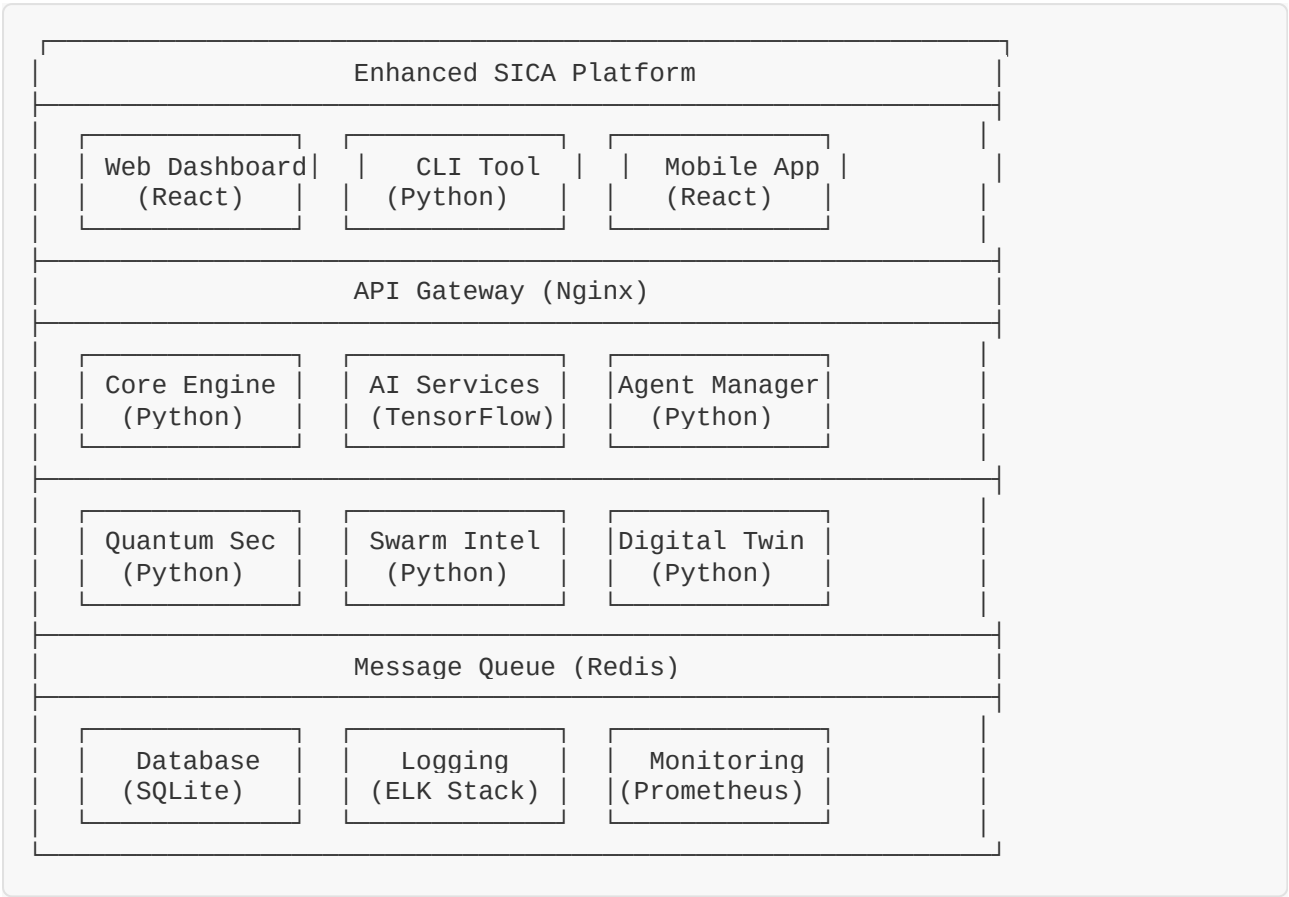
Table of Contents

System Architecture

Overview

Enhanced SICA employs a microservices architecture designed for scalability, resilience, and modularity. The system is built on modern technologies and follows cloud-native principles.

High-Level Architecture



Core Components

1. Enhanced SICA Engine

Technology Stack: - **Language**: Python 3.11 - **Framework**: Flask/FastAPI - **Concurrency**: AsyncIO - **Process Management**: Multiprocessing

Key Modules:

```
enhanced_sica_engine/
├── core/
│   ├── sica_engine.py          # Main engine
│   ├── cyber_stem_cells.py     # Self-healing system
│   ├── adaptive_immune.py      # Immune response
│   └── threat_processor.py     # Threat processing
├── protocols/
│   ├── modbus_handler.py       # Modbus protocol
│   ├── opcua_handler.py        # OPC UA protocol
│   ├── dnp3_handler.py         # DNP3 protocol
│   └── protocol_analyzer.py    # Protocol analysis
├── ai/
│   ├── predictive_intel.py     # Predictive AI
│   ├── explainable_ai.py       # XAI components
│   └── ml_models.py            # ML models
└── quantum/
    ├── quantum_crypto.py       # Quantum cryptography
    ├── qkd_protocol.py         # QKD implementation
    └── post_quantum.py         # Post-quantum algorithms
```

Architecture Patterns: - **Event-Driven**: Asynchronous event processing - **Plugin Architecture**: Modular component loading - **Observer Pattern**: Real-time monitoring - **Strategy Pattern**: Configurable algorithms

2. Web Dashboard

Technology Stack: - **Frontend**: React 18, TypeScript - **UI Framework**: Tailwind CSS, shadcn/ui - **State Management**: Zustand - **Charts**: Recharts, D3.js - **Build Tool**: Vite

Component Architecture:

```
src/
├── components/
│   ├── dashboard/
│   │   ├── Overview.tsx        # System overview
│   │   ├── Threats.tsx         # Threat monitoring
│   │   ├── Protocols.tsx       # Protocol analysis
│   │   ├── Agents.tsx          # Agent management
│   │   └── Analytics.tsx       # Advanced analytics
│   ├── ui/                     # Reusable UI components
│   └── charts/                 # Chart components
├── hooks/                      # Custom React hooks
├── services/                   # API services
├── stores/                     # State management
└── utils/                      # Utility functions
```

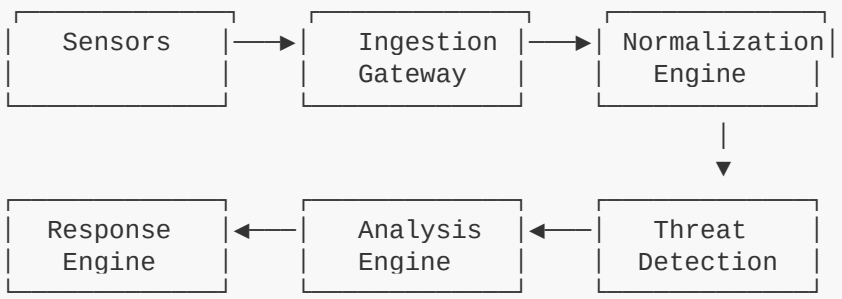
3. Agent System

ECO Agent Architecture:

```python
class ECOAgent:
    def __init__(self, agent_id: str, config: AgentConfig):
        self.agent_id = agent_id
        self.config = config
        self.capabilities = self._load_capabilities()
        self.communication = SecureCommunication()

    async def execute_mission(self, mission: Mission) -> MissionResult:
        """Execute reconnaissance mission"""
        try:
            # Stealth mode activation
            await self.activate_stealth_mode()

            # Target reconnaissance
            targets = await self.discover_targets()

            # Vulnerability scanning
            vulnerabilities = await self.scan_vulnerabilities(targets)

            # Data collection
            intelligence = await self.collect_intelligence()

            # Report generation
            return self.generate_report(vulnerabilities, intelligence)

        except Exception as e:
            await self.handle_mission_failure(e)
            raise
```

Data Flow Architecture

1. Event Processing Pipeline



2. AI Processing Pipeline

```python
class ThreatAnalysisPipeline:
    def __init__(self):
        self.preprocessor = DataPreprocessor()
        self.feature_extractor = FeatureExtractor()
        self.ml_models = ModelEnsemble([
            LSTMThreatPredictor(),
            RandomForestClassifier(),
            XGBoostDetector()
        ])
        self.explainer = ExplainableAI()

    async def process_threat(self, raw_data: bytes) -> ThreatAnalysis:
        # Data preprocessing
        processed_data = self.preprocessor.process(raw_data)

        # Feature extraction
        features = self.feature_extractor.extract(processed_data)

        # ML prediction
        predictions = await self.ml_models.predict(features)

        # Confidence scoring
        confidence = self.calculate_confidence(predictions)

        # Explanation generation
        explanation = self.explainer.explain(features, predictions)

        return ThreatAnalysis(
            threat_type=predictions.threat_type,
            severity=predictions.severity,
            confidence=confidence,
            explanation=explanation,
            recommended_actions=self.get_recommendations(predictions)
        )
```

Scalability Architecture

Horizontal Scaling

Microservices Deployment:

```yaml
# Kubernetes deployment example
apiVersion: apps/v1
kind: Deployment
metadata:
  name: enhanced-sica-core
spec:
  replicas: 3
  selector:
    matchLabels:
      app: enhanced-sica-core
  template:
    metadata:
      labels:
        app: enhanced-sica-core
    spec:
      containers:
      - name: sica-core
        image: enhanced-sica/core:latest
        ports:
        - containerPort: 5000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: sica-secrets
              key: database-url
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "500m"
```

Load Balancing

Nginx Configuration:

```
upstream sica_backend {
    least_conn;
    server sica-core-1:5000 weight=3;
    server sica-core-2:5000 weight=3;
    server sica-core-3:5000 weight=2;
}

server {
    listen 80;
    server_name enhanced-sica.local;

    location /api/ {
        proxy_pass http://sica_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        # Health check
        proxy_next_upstream error timeout invalid_header http_500;
        proxy_connect_timeout 2s;
        proxy_read_timeout 10s;
    }
}
```

API Reference

Authentication

Enhanced SICA uses JWT-based authentication with optional multi-factor authentication.

Authentication Endpoints

POST /api/v1/auth/login

Request:

```
{
  "username": "admin",
  "password": "secure_password",
  "mfa_token": "123456"
}
```

Response:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "expires_in": 3600,
  "user": {
    "id": "user_123",
    "username": "admin",
    "role": "administrator",
    "permissions": ["read", "write", "admin"]
  }
}
```

POST /api/v1/auth/refresh

Request:

```
{
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Core API Endpoints

System Status

GET /api/v1/system/status

Response:

```
{
  "status": "healthy",
  "version": "1.0.0",
  "uptime": 86400,
  "components": {
    "core_engine": {
      "status": "running",
      "cpu_usage": 15.2,
      "memory_usage": 45.8
    },
    "ai_services": {
      "status": "running",
      "models_loaded": 5,
      "prediction_queue": 12
    },
    "agent_manager": {
      "status": "running",
      "active_agents": 8,
      "total_agents": 12
    }
  },
  "metrics": {
    "events_processed": 125847,
    "threats_detected": 342,
    "response_time_ms": 47,
    "immunity_level": 94.2
  }
}
```


Threat Management

GET /api/v1/threats

Query Parameters: - `severity`: Filter by severity (critical, high, medium, low) - `status`: Filter by status (active, resolved, investigating) - `limit`: Number of results (default: 50) - `offset`: Pagination offset

Response:

```
{
  "threats": [
    {
      "id": "threat_001",
      "timestamp": "2024-12-24T10:30:00Z",
      "severity": "high",
      "status": "active",
      "type": "malware_detection",
      "source": {
        "ip": "192.168.1.100",
        "hostname": "workstation-01",
        "protocol": "modbus"
      },
      "target": {
        "ip": "192.168.1.10",
        "hostname": "plc-01",
        "protocol": "modbus"
      },
      "confidence": 0.92,
      "description": "Suspicious Modbus function code detected",
      "ai_analysis": {
        "model": "ensemble_classifier",
        "features": ["unusual_function_code", "timing_anomaly"],
        "explanation": "The detected pattern shows characteristics of known
malware families targeting industrial control systems."
      },
      "recommended_actions": [
        "isolate_affected_system",
        "update_security_policies",
        "deploy_additional_monitoring"
      ]
    }
  ],
  "pagination": {
    "total": 342,
    "limit": 50,
    "offset": 0,
    "has_more": true
  }
}
```


POST /api/v1/threats/{threat_id}/respond

Request:


```
{
  "action": "isolate_system",
  "parameters": {
    "isolation_level": "network",
    "duration": 3600,
    "notify_admin": true
  }
}
```

Agent Management

GET /api/v1/agents

Response:

```
{
  "agents": [
    {
      "id": "agent_001",
      "type": "eco_agent",
      "status": "active",
      "deployment": {
        "target": "192.168.1.0/24",
        "mode": "stealth",
        "deployed_at": "2024-12-24T08:00:00Z"
      },
      "capabilities": [
        "network_discovery",
        "vulnerability_scanning",
        "traffic_analysis"
      ],
      "current_mission": {
        "id": "mission_123",
        "type": "reconnaissance",
        "progress": 75,
        "estimated_completion": "2024-12-24T12:00:00Z"
      },
      "metrics": {
        "systems_discovered": 45,
        "vulnerabilities_found": 8,
        "data_collected_mb": 12.5
      }
    }
  ]
}
```

POST /api/v1/agents/deploy

Request:

```
{
  "agent_type": "eco_agent",
  "target_network": "192.168.1.0/24",
  "deployment_mode": "stealth",
  "mission_config": {
    "scan_intensity": "low",
    "stealth_level": "high",
    "reporting_interval": 300
  }
}
```

Protocol Analysis

GET /api/v1/protocols/{protocol}/traffic

Response:


```
{
  "protocol": "modbus",
  "timeframe": {
    "start": "2024-12-24T10:00:00Z",
    "end": "2024-12-24T11:00:00Z"
  },
  "statistics": {
    "total_packets": 15420,
    "unique_sources": 12,
    "unique_destinations": 8,
    "function_codes": {
      "read_coils": 8500,
      "read_holding_registers": 4200,
      "write_single_coil": 1800,
      "write_multiple_registers": 920
    }
  },
  "anomalies": [
    {
      "timestamp": "2024-12-24T10:15:30Z",
      "type": "unusual_function_code",
      "severity": "medium",
      "description": "Function code 0x90 detected (non-standard)",
      "source": "192.168.1.100",
      "destination": "192.168.1.10"
    }
  ]
}
```


AI and ML APIs

Predictive Intelligence

POST /api/v1/ai/predict

Request:

```
{
  "data_type": "network_traffic",
  "timeframe_hours": 24,
  "features": {
    "packet_rate": 1500,
    "protocol_distribution": {
      "modbus": 0.6,
      "opcua": 0.3,
      "dnp3": 0.1
    },
    "anomaly_score": 0.15
  }
}
```

Response:

```
{
  "predictions": [
    {
      "timeframe": "1h",
      "threat_probability": 0.12,
      "confidence": 0.89,
      "threat_types": {
        "malware": 0.08,
        "dos_attack": 0.03,
        "data_exfiltration": 0.01
      }
    },
    {
      "timeframe": "24h",
      "threat_probability": 0.34,
      "confidence": 0.76,
      "threat_types": {
        "malware": 0.20,
        "dos_attack": 0.10,
        "data_exfiltration": 0.04
      }
    }
  ],
  "model_info": {
    "model_type": "lstm_ensemble",
    "training_data_points": 1000000,
    "last_updated": "2024-12-20T00:00:00Z"
  }
}
```

Explainable AI

POST /api/v1/ai/explain

Request:

```
{
  "prediction_id": "pred_123",
  "explanation_type": "shap",
  "detail_level": "comprehensive"
}
```

Response:

```
{
  "explanation": {
    "method": "shap",
    "feature_importance": [
      {
        "feature": "packet_rate_anomaly",
        "importance": 0.45,
        "contribution": "positive",
        "description": "Packet rate 3.2x higher than baseline"
      },
      {
        "feature": "protocol_timing",
        "importance": 0.32,
        "contribution": "positive",
        "description": "Unusual timing patterns in Modbus communications"
      }
    ],
    "decision_path": [
      "High packet rate detected",
      "Timing anomaly confirmed",
      "Pattern matches known attack signatures",
      "Confidence threshold exceeded"
    ],
    "visualization_url": "/api/v1/ai/explain/pred_123/visualization"
  }
}
```

Quantum Security APIs

Quantum Key Distribution

POST /api/v1/quantum/qkd/establish

Request:

```
{
  "peer_id": "node_002",
  "protocol": "bb84",
  "key_length": 256,
  "security_level": "high"
}
```

Response:

```
{
  "session_id": "qkd_session_001",
  "status": "established",
  "key_id": "qkey_12345",
  "security_metrics": {
    "quantum_bit_error_rate": 0.02,
    "key_generation_rate": 1000,
    "security_parameter": 128
  }
}
```

WebSocket APIs

Real-time Threat Feed

WebSocket: /ws/threats

Connection:

```
const ws = new WebSocket('ws://localhost:5000/ws/threats');

ws.onmessage = function(event) {
    const threat = JSON.parse(event.data);
    console.log('New threat:', threat);
};
```

Message Format:

```
{
  "type": "threat_detected",
  "timestamp": "2024-12-24T10:30:00Z",
  "threat": {
    "id": "threat_002",
    "severity": "critical",
    "type": "malware_detection",
    "confidence": 0.95,
    "source": "192.168.1.100",
    "target": "192.168.1.10"
  }
}
```

Database Schema

Overview

Enhanced SICA uses SQLite for development and testing, with PostgreSQL support for production deployments. The database schema is designed for high performance and scalability.

Core Tables

1. Threats Table

```
CREATE TABLE threats (
    id VARCHAR(50) PRIMARY KEY,
    timestamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    severity VARCHAR(20) NOT NULL CHECK (severity IN ('critical', 'high',
'medium', 'low')),
    status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('active',
'resolved', 'investigating', 'false_positive')),
    type VARCHAR(50) NOT NULL,
    source_ip INET,
    source_hostname VARCHAR(255),
    target_ip INET,
    target_hostname VARCHAR(255),
    protocol VARCHAR(20),
    confidence DECIMAL(3,2) CHECK (confidence >= 0 AND confidence <= 1),
    description TEXT,
    raw_data JSONB,
    ai_analysis JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes for performance
CREATE INDEX idx_threats_timestamp ON threats(timestamp);
CREATE INDEX idx_threats_severity ON threats(severity);
CREATE INDEX idx_threats_status ON threats(status);
CREATE INDEX idx_threats_source_ip ON threats(source_ip);
CREATE INDEX idx_threats_type ON threats(type);
```

2. Agents Table

```sql
CREATE TABLE agents (
    id VARCHAR(50) PRIMARY KEY,
    type VARCHAR(30) NOT NULL,
    status VARCHAR(20) NOT NULL DEFAULT 'inactive' CHECK (status IN ('active',
'inactive', 'deploying', 'error')),
    deployment_target VARCHAR(100),
    deployment_mode VARCHAR(20) DEFAULT 'normal',
    capabilities JSONB,
    configuration JSONB,
    metrics JSONB,
    last_heartbeat TIMESTAMP,
    deployed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_agents_status ON agents(status);
CREATE INDEX idx_agents_type ON agents(type);
CREATE INDEX idx_agents_last_heartbeat ON agents(last_heartbeat);
```

3. Protocol Traffic Table

```sql
CREATE TABLE protocol_traffic (
    id BIGSERIAL PRIMARY KEY,
    timestamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    protocol VARCHAR(20) NOT NULL,
    source_ip INET NOT NULL,
    destination_ip INET NOT NULL,
    source_port INTEGER,
    destination_port INTEGER,
    function_code INTEGER,
    data_length INTEGER,
    packet_data BYTEA,
    anomaly_score DECIMAL(3,2),
    is_anomaly BOOLEAN DEFAULT FALSE,
    processed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Partitioning by date for performance
CREATE TABLE protocol_traffic_y2024m12 PARTITION OF protocol_traffic
    FOR VALUES FROM ('2024-12-01') TO ('2025-01-01');

-- Indexes
CREATE INDEX idx_protocol_traffic_timestamp ON protocol_traffic(timestamp);
CREATE INDEX idx_protocol_traffic_protocol ON protocol_traffic(protocol);
CREATE INDEX idx_protocol_traffic_anomaly ON protocol_traffic(is_anomaly);
```

4. System Events Table

```sql
CREATE TABLE system_events (
    id BIGSERIAL PRIMARY KEY,
    timestamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    event_type VARCHAR(50) NOT NULL,
    severity VARCHAR(20) NOT NULL,
    component VARCHAR(50),
    message TEXT,
    details JSONB,
    user_id VARCHAR(50),
    session_id VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_system_events_timestamp ON system_events(timestamp);
CREATE INDEX idx_system_events_type ON system_events(event_type);
CREATE INDEX idx_system_events_severity ON system_events(severity);
```

5. AI Models Table

```sql
CREATE TABLE ai_models (
    id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    type VARCHAR(50) NOT NULL,
    version VARCHAR(20) NOT NULL,
    status VARCHAR(20) DEFAULT 'inactive',
    configuration JSONB,
    performance_metrics JSONB,
    training_data_info JSONB,
    model_file_path VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_trained TIMESTAMP
);
```

Advanced Schema Features

1. Cyber Stem Cells Schema

```
CREATE TABLE cyber_stem_cells (
    id VARCHAR(50) PRIMARY KEY,
    cell_type VARCHAR(30) NOT NULL CHECK (cell_type IN ('stem', 'immune',
'neural', 'security', 'protocol', 'quantum', 'swarm', 'twin')),
    status VARCHAR(20) DEFAULT 'healthy',
    health_score DECIMAL(3,2) DEFAULT 1.00,
    generation INTEGER DEFAULT 0,
    parent_cell_id VARCHAR(50),
    capabilities JSONB,
    memory_data JSONB,
    last_regeneration TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (parent_cell_id) REFERENCES cyber_stem_cells(id)
);
```

2. Immune Memory Schema

```
CREATE TABLE immune_memory (
    id VARCHAR(50) PRIMARY KEY,
    pathogen_signature VARCHAR(255) NOT NULL,
    pathogen_type VARCHAR(50),
    antibody_data JSONB,
    effectiveness_score DECIMAL(3,2),
    encounter_count INTEGER DEFAULT 1,
    first_encounter TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_encounter TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    memory_strength DECIMAL(3,2) DEFAULT 1.00
);

CREATE UNIQUE INDEX idx_immune_memory_signature ON
immune_memory(pathogen_signature);
```

3. Quantum Keys Schema

```
CREATE TABLE quantum_keys (
    id VARCHAR(50) PRIMARY KEY,
    session_id VARCHAR(100) NOT NULL,
    peer_id VARCHAR(50) NOT NULL,
    key_data BYTEA NOT NULL,
    key_length INTEGER NOT NULL,
    protocol VARCHAR(20) NOT NULL,
    security_level VARCHAR(20),
    qber DECIMAL(4,3), -- Quantum Bit Error Rate
    generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP,
    status VARCHAR(20) DEFAULT 'active'
);
```

Database Performance Optimization

1. Indexing Strategy

```
-- Composite indexes for common queries
CREATE INDEX idx_threats_severity_timestamp ON threats(severity, timestamp
DESC);
CREATE INDEX idx_protocol_traffic_protocol_timestamp ON
protocol_traffic(protocol, timestamp DESC);

-- Partial indexes for active records
CREATE INDEX idx_active_threats ON threats(timestamp DESC) WHERE status =
'active';
CREATE INDEX idx_active_agents ON agents(last_heartbeat DESC) WHERE status =
'active';

-- GIN indexes for JSONB columns
CREATE INDEX idx_threats_ai_analysis_gin ON threats USING GIN (ai_analysis);
CREATE INDEX idx_agents_capabilities_gin ON agents USING GIN (capabilities);
```

2. Partitioning Strategy

```
-- Time-based partitioning for large tables
CREATE TABLE protocol_traffic (
    id BIGSERIAL,
    timestamp TIMESTAMP NOT NULL,
    -- other columns...
) PARTITION BY RANGE (timestamp);

-- Monthly partitions
CREATE TABLE protocol_traffic_y2024m12 PARTITION OF protocol_traffic
    FOR VALUES FROM ('2024-12-01') TO ('2025-01-01');
```

3. Data Retention Policies

```
 -- Automated cleanup procedures
CREATE OR REPLACE FUNCTION cleanup_old_data()
RETURNS void AS ```math

BEGIN
    -- Delete old resolved threats (older than 90 days)
    DELETE FROM threats
    WHERE status = 'resolved'
    AND updated_at < NOW() - INTERVAL '90 days';

    -- Archive old protocol traffic (older than 30 days)
    INSERT INTO protocol_traffic_archive
    SELECT * FROM protocol_traffic
    WHERE timestamp < NOW() - INTERVAL '30 days';

    DELETE FROM protocol_traffic
    WHERE timestamp < NOW() - INTERVAL '30 days';

    -- Update statistics
    ANALYZE threats;
    ANALYZE protocol_traffic;
END;

``` LANGUAGE plpgsql;

-- Schedule cleanup job
SELECT cron.schedule('cleanup-old-data', '0 2 * * *', 'SELECT
cleanup_old_data();');
```

Security Protocols

Authentication and Authorization

1. Multi-Factor Authentication (MFA)

Enhanced SICA implements TOTP-based MFA using the following protocol:
```

```python
class MFAManager:
    def __init__(self):
        self.secret_key = self._generate_secret()
        self.window_size = 1  # Allow 1 time step tolerance

    def generate_qr_code(self, username: str) -> str:
        """Generate QR code for MFA setup"""
        issuer = "Enhanced SICA"
        uri = f"otpauth://totp/{issuer}:{username}?secret=
{self.secret_key}&issuer={issuer}"
        return self._generate_qr_code(uri)

    def verify_token(self, token: str, secret: str) -> bool:
        """Verify TOTP token"""
        import pyotp
        totp = pyotp.TOTP(secret)
        return totp.verify(token, valid_window=self.window_size)
```

2. Role-Based Access Control (RBAC)

```python
class Permission(Enum):
    READ_THREATS = "read:threats"
    WRITE_THREATS = "write:threats"
    DEPLOY_AGENTS = "deploy:agents"
    MANAGE_SYSTEM = "manage:system"
    ADMIN_ALL = "admin:all"

class Role:
    VIEWER = [Permission.READ_THREATS]
    OPERATOR = [Permission.READ_THREATS, Permission.WRITE_THREATS]
    ADMIN = [Permission.READ_THREATS, Permission.WRITE_THREATS,
             Permission.DEPLOY_AGENTS, Permission.MANAGE_SYSTEM]
    SUPER_ADMIN = [Permission.ADMIN_ALL]

@require_permission(Permission.DEPLOY_AGENTS)
def deploy_agent(agent_config: AgentConfig):
    """Deploy new agent - requires deploy permission"""
    pass
```

Encryption Standards

1. Data at Rest Encryption

```python
class DataEncryption:
    def __init__(self):
        self.cipher_suite = Fernet(self._load_encryption_key())

    def encrypt_sensitive_data(self, data: str) -> bytes:
        """Encrypt sensitive data using AES-256"""
        return self.cipher_suite.encrypt(data.encode())

    def decrypt_sensitive_data(self, encrypted_data: bytes) -> str:
        """Decrypt sensitive data"""
        return self.cipher_suite.decrypt(encrypted_data).decode()

    def _load_encryption_key(self) -> bytes:
        """Load encryption key from secure storage"""
        # Key derivation using PBKDF2
        password = os.environ['SICA_MASTER_PASSWORD']
        salt = os.environ['SICA_SALT']
        return PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=32,
            salt=salt.encode(),
            iterations=100000,
        ).derive(password.encode())
```

2. Data in Transit Encryption

```python
class SecureCommunication:
    def __init__(self):
        self.context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH)
        self.context.check_hostname = False
        self.context.verify_mode = ssl.CERT_REQUIRED

    async def secure_send(self, data: dict, endpoint: str) -> dict:
        """Send data over encrypted channel"""
        async with aiohttp.ClientSession(
            connector=aiohttp.TCPConnector(ssl=self.context)
        ) as session:
            async with session.post(
                endpoint,
                json=data,
                headers={"Authorization": f"Bearer {self.get_jwt_token()}"}
            ) as response:
                return await response.json()
```

Quantum Security Implementation

1. Post-Quantum Cryptography

```python
class PostQuantumCrypto:
    def __init__(self):
        # CRYSTALS-Kyber for key encapsulation
        self.kem = Kyber512()
        # CRYSTALS-Dilithium for digital signatures
        self.signature = Dilithium2()

    def generate_keypair(self) -> Tuple[bytes, bytes]:
        """Generate post-quantum key pair"""
        public_key, private_key = self.kem.keygen()
        return public_key, private_key

    def encapsulate_secret(self, public_key: bytes) -> Tuple[bytes, bytes]:
        """Encapsulate shared secret"""
        ciphertext, shared_secret = self.kem.encaps(public_key)
        return ciphertext, shared_secret

    def sign_message(self, message: bytes, private_key: bytes) -> bytes:
        """Create post-quantum digital signature"""
        return self.signature.sign(message, private_key)
```

2. Quantum Key Distribution (QKD)

```python
class BB84Protocol:
    def __init__(self):
        self.bases = ['rectilinear', 'diagonal']
        self.qber_threshold = 0.11  # Quantum Bit Error Rate threshold

    async def establish_quantum_channel(self, peer_id: str) -> QuantumChannel:
        """Establish QKD channel using BB84 protocol"""
        # Step 1: Alice prepares random bits and bases
        alice_bits = self._generate_random_bits(1000)
        alice_bases = self._generate_random_bases(1000)

        # Step 2: Alice sends qubits to Bob
        qubits = self._prepare_qubits(alice_bits, alice_bases)
        await self._send_qubits(qubits, peer_id)

        # Step 3: Bob measures with random bases
        bob_bases = self._generate_random_bases(1000)
        bob_bits = await self._receive_measurements(peer_id)

        # Step 4: Basis reconciliation
        matching_bases = self._compare_bases(alice_bases, bob_bases)
        sifted_key = self._sift_key(alice_bits, bob_bits, matching_bases)

        # Step 5: Error estimation
        qber = self._estimate_qber(sifted_key)
        if qber > self.qber_threshold:
            raise SecurityError("QBER too high - possible eavesdropping")

        # Step 6: Privacy amplification
        final_key = self._privacy_amplification(sifted_key)

        return QuantumChannel(peer_id, final_key, qber)
```

Zero-Knowledge Protocols

1. ZK-SNARK Implementation

```python
class ZKProof:
    def __init__(self):
        self.circuit = self._load_circuit()
        self.proving_key = self._load_proving_key()
        self.verification_key = self._load_verification_key()

    def generate_proof(self, secret_input: int, public_input: int) -> ZKProof:
        """Generate zero-knowledge proof"""
        witness = self._compute_witness(secret_input, public_input)
        proof = self.circuit.prove(witness, self.proving_key)
        return ZKProof(proof, public_input)

    def verify_proof(self, proof: ZKProof) -> bool:
        """Verify zero-knowledge proof"""
        return self.circuit.verify(
            proof.proof,
            proof.public_input,
            self.verification_key
        )
```

Secure Agent Communication

1. Agent Authentication Protocol

```
class AgentAuthProtocol:
    def __init__(self):
        self.agent_certificates = {}
        self.revoked_certificates = set()

    async def authenticate_agent(self, agent_id: str, certificate: bytes) ->
bool:
        """Authenticate agent using certificate"""
        # Verify certificate signature
        if not self._verify_certificate_signature(certificate):
            return False

        # Check certificate revocation
        if certificate in self.revoked_certificates:
            return False

        # Verify agent identity
        cert_agent_id = self._extract_agent_id(certificate)
        if cert_agent_id != agent_id:
            return False

        # Store valid certificate
        self.agent_certificates[agent_id] = certificate
        return True

    def revoke_agent_certificate(self, agent_id: str):
        """Revoke agent certificate"""
        if agent_id in self.agent_certificates:
            cert = self.agent_certificates[agent_id]
            self.revoked_certificates.add(cert)
            del self.agent_certificates[agent_id]
```

Performance Tuning

System Optimization

1. CPU Optimization

Multi-processing Configuration:

```python
class SICAEngine:
    def __init__(self):
        self.cpu_count = multiprocessing.cpu_count()
        self.worker_processes = min(self.cpu_count - 1, 8)
        self.thread_pool = ThreadPoolExecutor(max_workers=self.cpu_count * 2)

    async def process_threats_parallel(self, threats: List[ThreatData]):
        """Process threats using multiple processes"""
        chunk_size = len(threats) // self.worker_processes
        chunks = [threats[i:i + chunk_size] for i in range(0, len(threats),
chunk_size)]

        with ProcessPoolExecutor(max_workers=self.worker_processes) as
executor:
            futures = [executor.submit(self._process_threat_chunk, chunk) for
chunk in chunks]
            results = await asyncio.gather(*[asyncio.wrap_future(f) for f in
futures])

        return [item for sublist in results for item in sublist]
```

CPU Affinity Settings:

```bash
# Set CPU affinity for Enhanced SICA processes
taskset -c 0-3 python src/core/enhanced_sica_engine.py
taskset -c 4-7 python src/ai/predictive_intelligence.py
```

2. Memory Optimization

Memory Pool Management:

```python
class MemoryManager:
    def __init__(self):
        self.object_pools = {
            'threat_objects': ObjectPool(ThreatData, initial_size=1000),
            'packet_objects': ObjectPool(PacketData, initial_size=5000),
            'analysis_objects': ObjectPool(AnalysisResult, initial_size=500)
        }

    def get_object(self, object_type: str):
        """Get object from pool to avoid allocation overhead"""
        return self.object_pools[object_type].get()

    def return_object(self, obj, object_type: str):
        """Return object to pool for reuse"""
        obj.reset()  # Clear object state
        self.object_pools[object_type].put(obj)
```

Memory Monitoring:

```python
import psutil
import gc

class MemoryMonitor:
    def __init__(self):
        self.memory_threshold = 0.85  # 85% memory usage threshold

    def monitor_memory_usage(self):
        """Monitor and optimize memory usage"""
        memory_percent = psutil.virtual_memory().percent / 100

        if memory_percent > self.memory_threshold:
            # Force garbage collection
            gc.collect()

            # Clear caches
            self._clear_caches()

            # Log memory warning
            logger.warning(f"High memory usage: {memory_percent:.1%}")

    def _clear_caches(self):
        """Clear internal caches to free memory"""
        # Clear threat analysis cache
        ThreatAnalyzer.clear_cache()

        # Clear protocol parser cache
        ProtocolParser.clear_cache()
```

3. I/O Optimization

Asynchronous I/O Configuration:

```python
class AsyncIOOptimizer:
    def __init__(self):
        # Configure event loop for high performance
        if sys.platform == 'linux':
            # Use uvloop for better performance on Linux
            import uvloop
            asyncio.set_event_loop_policy(uvloop.EventLoopPolicy())

        self.loop = asyncio.new_event_loop()
        asyncio.set_event_loop(self.loop)

    async def batch_database_operations(self, operations: List[DatabaseOp]):
        """Batch database operations for better performance"""
        async with self.db_pool.acquire() as conn:
            async with conn.transaction():
                await asyncio.gather(*[op.execute(conn) for op in operations])
```

Database Performance

1. Query Optimization

Optimized Threat Queries:

```sql
-- Optimized threat retrieval with proper indexing
EXPLAIN ANALYZE
SELECT t.id, t.timestamp, t.severity, t.type, t.confidence,
       t.source_ip, t.target_ip, t.description
FROM threats t
WHERE t.status = 'active'
  AND t.timestamp >= NOW() - INTERVAL '24 hours'
  AND t.severity IN ('critical', 'high')
ORDER BY t.timestamp DESC, t.severity DESC
LIMIT 100;

-- Index to support this query
CREATE INDEX CONCURRENTLY idx_threats_active_recent
ON threats(status, timestamp DESC, severity)
WHERE status = 'active';
```

Connection Pooling:

```python
class DatabasePool:
    def __init__(self):
        self.pool = asyncpg.create_pool(
            host='localhost',
            database='enhanced_sica',
            user='sica_user',
            password='secure_password',
            min_size=10,
            max_size=50,
            command_timeout=60,
            server_settings={
                'jit': 'off',  # Disable JIT for consistent performance
                'shared_preload_libraries': 'pg_stat_statements'
            }
        )

    async def execute_optimized_query(self, query: str, *args):
        """Execute query with connection pooling"""
        async with self.pool.acquire() as conn:
            # Use prepared statements for better performance
            stmt = await conn.prepare(query)
            return await stmt.fetch(*args)
```

2. Caching Strategy

Multi-Level Caching:

```python
class CacheManager:
    def __init__(self):
        # L1 Cache: In-memory LRU cache
        self.l1_cache = LRUCache(maxsize=1000)

        # L2 Cache: Redis distributed cache
        self.l2_cache = redis.Redis(
            host='localhost',
            port=6379,
            db=0,
            decode_responses=True,
            socket_keepalive=True,
            socket_keepalive_options={}
        )

    async def get_cached_threat_analysis(self, threat_id: str) ->
Optional[ThreatAnalysis]:
        """Get threat analysis with multi-level caching"""
        # Check L1 cache first
        result = self.l1_cache.get(threat_id)
        if result:
            return result

        # Check L2 cache
        cached_data = await self.l2_cache.get(f"threat:{threat_id}")
        if cached_data:
            result = ThreatAnalysis.from_json(cached_data)
            self.l1_cache[threat_id] = result
            return result

        return None

    async def cache_threat_analysis(self, threat_id: str, analysis:
ThreatAnalysis):
        """Cache threat analysis at multiple levels"""
        # Store in L1 cache
        self.l1_cache[threat_id] = analysis

        # Store in L2 cache with TTL
        await self.l2_cache.setex(
            f"threat:{threat_id}",
            3600,  # 1 hour TTL
            analysis.to_json()
        )
```

Network Performance

1. Protocol Optimization

High-Performance Packet Processing:

```python
class PacketProcessor:
    def __init__(self):
        self.packet_queue = asyncio.Queue(maxsize=10000)
        self.processing_workers = []

    async def start_processing(self):
        """Start high-performance packet processing"""
        # Create multiple worker coroutines
        for i in range(multiprocessing.cpu_count()):
            worker = asyncio.create_task(self._packet_worker(f"worker-{i}"))
            self.processing_workers.append(worker)

    async def _packet_worker(self, worker_id: str):
        """High-performance packet processing worker"""
        while True:
            try:
                # Get packet from queue with timeout
                packet = await asyncio.wait_for(
                    self.packet_queue.get(),
                    timeout=1.0
                )

                # Process packet efficiently
                await self._process_packet_fast(packet)

                # Mark task as done
                self.packet_queue.task_done()

            except asyncio.TimeoutError:
                continue
            except Exception as e:
                logger.error(f"Worker {worker_id} error: {e}")
```

2. Load Balancing

Nginx Configuration for High Performance:

```
upstream sica_backend {
    least_conn;
    keepalive 32;

    server sica-core-1:5000 weight=3 max_fails=3 fail_timeout=30s;
    server sica-core-2:5000 weight=3 max_fails=3 fail_timeout=30s;
    server sica-core-3:5000 weight=2 max_fails=3 fail_timeout=30s;
}

server {
    listen 80;
    server_name enhanced-sica.local;

    # Performance optimizations
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    keepalive_requests 1000;

    # Compression
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;
    gzip_types text/plain text/css application/json application/javascript;

    # Caching
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    location /api/ {
        proxy_pass http://sica_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        # Performance settings
        proxy_buffering on;
        proxy_buffer_size 128k;
        proxy_buffers 4 256k;
        proxy_busy_buffers_size 256k;

        # Timeouts
        proxy_connect_timeout 5s;
        proxy_send_timeout 10s;
        proxy_read_timeout 10s;
    }
}
```

Monitoring and Metrics

1. Performance Metrics Collection

```python
class PerformanceMonitor:
    def __init__(self):
        self.metrics = {
            'requests_per_second': Counter(),
            'response_time': Histogram(),
            'active_connections': Gauge(),
            'memory_usage': Gauge(),
            'cpu_usage': Gauge()
        }

    @contextmanager
    def measure_response_time(self):
        """Context manager to measure response time"""
        start_time = time.time()
        try:
            yield
        finally:
            response_time = time.time() - start_time
            self.metrics['response_time'].observe(response_time)

    async def collect_system_metrics(self):
        """Collect system performance metrics"""
        while True:
            # CPU usage
            cpu_percent = psutil.cpu_percent(interval=1)
            self.metrics['cpu_usage'].set(cpu_percent)

            # Memory usage
            memory = psutil.virtual_memory()
            self.metrics['memory_usage'].set(memory.percent)

            # Network connections
            connections = len(psutil.net_connections())
            self.metrics['active_connections'].set(connections)

            await asyncio.sleep(10)  # Collect every 10 seconds
```

Integration Specifications

SIEM Integration

1. Syslog Integration

```python
class SyslogIntegration:
    def __init__(self, siem_host: str, siem_port: int = 514):
        self.siem_host = siem_host
        self.siem_port = siem_port
        self.logger = logging.getLogger('sica.siem')

        # Configure syslog handler
        handler = SysLogHandler(address=(siem_host, siem_port))
        formatter = logging.Formatter(
            'Enhanced-SICA: %(levelname)s - %(message)s'
        )
        handler.setFormatter(formatter)
        self.logger.addHandler(handler)

    def send_threat_alert(self, threat: ThreatData):
        """Send threat alert to SIEM"""
        alert_message = {
            'timestamp': threat.timestamp.isoformat(),
            'threat_id': threat.id,
            'severity': threat.severity,
            'source_ip': str(threat.source_ip),
            'target_ip': str(threat.target_ip),
            'description': threat.description,
            'confidence': threat.confidence
        }

        self.logger.warning(json.dumps(alert_message))
```

2. CEF (Common Event Format) Integration

```
class CEFIntegration:
    def __init__(self):
        self.device_vendor = "Enhanced SICA"
        self.device_product = "Cybersecurity Platform"
        self.device_version = "1.0"

    def format_cef_message(self, threat: ThreatData) -> str:
        """Format threat as CEF message"""
        # CEF Header
        cef_header = f"CEF:0|{self.device_vendor}|{self.device_product}|
{self.device_version}"

        # Event details
        signature_id = threat.type
        name = threat.description
        severity = self._map_severity_to_cef(threat.severity)

        # Extensions
        extensions = [
            f"src={threat.source_ip}",
            f"dst={threat.target_ip}",
            f"spt={threat.source_port}",
            f"dpt={threat.target_port}",
            f"proto={threat.protocol}",
            f"act=detected",
            f"cnt=1",
            f"cs1={threat.confidence}",
            f"cs1Label=Confidence"
        ]

        return f"{cef_header}|{signature_id}|{name}|{severity}|{'
'.join(extensions)}"
```

API Integration Examples

1. REST API Client

```python
class SICAAPIClient:
    def __init__(self, base_url: str, api_key: str):
        self.base_url = base_url
        self.api_key = api_key
        self.session = aiohttp.ClientSession(
            headers={"Authorization": f"Bearer {api_key}"}
        )

    async def get_active_threats(self, limit: int = 50) -> List[dict]:
        """Get active threats from Enhanced SICA"""
        async with self.session.get(
            f"{self.base_url}/api/v1/threats",
            params={"status": "active", "limit": limit}
        ) as response:
            data = await response.json()
            return data["threats"]

    async def deploy_agent(self, target_network: str, config: dict) -> dict:
        """Deploy new ECO agent"""
        payload = {
            "agent_type": "eco_agent",
            "target_network": target_network,
            "deployment_mode": "stealth",
            "mission_config": config
        }

        async with self.session.post(
            f"{self.base_url}/api/v1/agents/deploy",
            json=payload
        ) as response:
            return await response.json()
```

2. Webhook Integration

```python
class WebhookManager:
    def __init__(self):
        self.webhooks = {}

    def register_webhook(self, event_type: str, url: str, secret: str):
        """Register webhook for specific events"""
        self.webhooks[event_type] = {
            'url': url,
            'secret': secret
        }

    async def send_webhook(self, event_type: str, data: dict):
        """Send webhook notification"""
        if event_type not in self.webhooks:
            return

        webhook = self.webhooks[event_type]

        # Create signature for verification
        signature = hmac.new(
            webhook['secret'].encode(),
            json.dumps(data).encode(),
            hashlib.sha256
        ).hexdigest()

        headers = {
            'Content-Type': 'application/json',
            'X-SICA-Signature': f'sha256={signature}',
            'X-SICA-Event': event_type
        }

        async with aiohttp.ClientSession() as session:
            async with session.post(
                webhook['url'],
                json=data,
                headers=headers
            ) as response:
                if response.status != 200:
                    logger.error(f"Webhook failed: {response.status}")
```

Cloud Platform Integration

1. AWS Integration

```python
class AWSIntegration:
    def __init__(self):
        self.s3_client = boto3.client('s3')
        self.cloudwatch = boto3.client('cloudwatch')
        self.sns = boto3.client('sns')

    async def backup_to_s3(self, data: bytes, key: str):
        """Backup data to AWS S3"""
        try:
            self.s3_client.put_object(
                Bucket='enhanced-sica-backups',
                Key=key,
                Body=data,
                ServerSideEncryption='AES256'
            )
        except Exception as e:
            logger.error(f"S3 backup failed: {e}")

    async def send_cloudwatch_metrics(self, metrics: dict):
        """Send metrics to CloudWatch"""
        metric_data = []

        for metric_name, value in metrics.items():
            metric_data.append({
                'MetricName': metric_name,
                'Value': value,
                'Unit': 'Count',
                'Timestamp': datetime.utcnow()
            })

        self.cloudwatch.put_metric_data(
            Namespace='Enhanced-SICA',
            MetricData=metric_data
        )
```

2. Kubernetes Integration

```yaml
# Enhanced SICA Kubernetes Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: enhanced-sica
  labels:
    app: enhanced-sica
spec:
  replicas: 3
  selector:
    matchLabels:
      app: enhanced-sica
  template:
    metadata:
      labels:
        app: enhanced-sica
    spec:
      containers:
      - name: sica-core
        image: enhanced-sica/core:1.0
        ports:
        - containerPort: 5000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: sica-secrets
              key: database-url
        - name: REDIS_URL
          valueFrom:
            configMapKeyRef:
              name: sica-config
              key: redis-url
        resources:
          requests:
            memory: "1Gi"
            cpu: "500m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 5000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 5000
          initialDelaySeconds: 5
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: enhanced-sica-service
spec:
  selector:
```

```
    app: enhanced-sica
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
  type: LoadBalancer
```

Document Information: - **Version**: 1.0 - **Last Updated**: December 2024 - **Document Owner**: Enhanced SICA Development Team - **Classification**: Technical - **Distribution**: Internal/Partner