Sarthi AI Agent Development Platform (SADP)

Production-Ready Independent Service Architecture

Overview

The Sarthi AI Agent Development Platform (SADP) is a standalone, enterprise-grade microservice that provides comprehensive AI agent development, evaluation, and management capabilities as an external API service for the main Sarthi healthcare application.

Architectural Improvements & Enhancements

1. Event-Driven Architecture with CQRS

```
┌─────────────────────────────────────────────────────────────────┐
│             Sarthi Healthcare Application                  │
├─────────────────────────────────────────────────────────────────┤
│ Patient Portal │ Provider Dashboard │ Admin Console │ Mobile │
└─────────────────────────────────────────────────────────────────┘
      │          │
      ▼          ▼
┌─────────────────────────────────────────────────────────────────┐
│         API Gateway (Kong/Istio) + Rate Limiting          │
├─────────────────────────────────────────────────────────────────┤
│        Authentication Layer (Auth0/Keycloak)              │
└─────────────────────────────────────────────────────────────────┘
      │          │
      ▼          ▼
┌─────────────────────────────────────────────────────────────────┐
│        Sarthi AI Agent Development Platform (SADP)          │
├─────────────────────────────────────────────────────────────────┤
│              Command Bus (CQRS)                       │
├─────────────────────────────────────────────────────────────────┤
│ Agent Runtime │ Evaluation │ Development │ Monitoring &   │
│ Service      │ Service    │ Service     │ Analytics      │
├─────────────────────────────────────────────────────────────────┤
│ • Agent Exec    │ • Test Gen    │ • Prompt Mgmt │ • Performance    │
│ • Orchestration │ • Evaluation  │ • A/B Testing │ • Real-time Mon  │
│ • Capability    │ • Benchmarks  │ • CI/CD       │ • Alerts         │
│   Discovery     │ • Compliance  │ • Deployment  │ • Analytics      │
│ • Circuit       │ • Regression  │ • Version Mgmt│ • Cost Tracking  │
│   Breakers      │   Testing     │ • Blue/Green  │ • Anomaly Detect │
└─────────────────────────────────────────────────────────────────┘
                 │
                 ▼
┌─────────────────────────────────────────────────────────────────┐
│              Event Streaming Layer                    │
├─────────────────────────────────────────────────────────────────┤
│  Apache Kafka  │ Redis Streams│ Event Store │ Message Router │
│  (Primary)     │ (Fast Cache) │ (Audit)     │ (Dead Letter Q) │
└─────────────────────────────────────────────────────────────────┘
                 │
                 ▼
┌─────────────────────────────────────────────────────────────────┐
│            Data & Infrastructure Layer                 │
├─────────────────────────────────────────────────────────────────┤
│  Database    │  Message    │  Storage    │  External APIs │
│  Cluster      │  Queue      │  Services   │  & Integrations  │
├───────────────┼─────────────┼─────────────┼──────────────────┤
│ • PostgreSQL  │ • Redis     │ • GCS Buckets│ • Claude API     │
│   (Primary)   │   Cluster   │ • File System│ • OpenAI API     │
│ • CockroachDB │ • Kafka     │ • CDN        │ • Healthcare APIs│
│   (Distributed)│ • RabbitMQ │ • Object Store│ • FHIR Services │
│ • MongoDB     │   (Fallback)│ • Backup     │ • EHR Systems    │
│   (Documents) │ • NATS      │   Storage    │ • Monitoring SaaS│
│ • InfluxDB    │   (Streaming)│ • Artifact  │ • Observability  │
│   (Time Series)│            │   Registry   │   Stack          │
└───────────────┴─────────────┴─────────────┴──────────────────┘
```

2. Enhanced Service Mesh Architecture

Service Discovery & Communication

```yaml
```

```yaml
# Istio Service Mesh Configuration
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: sadp-routing
spec:
  hosts:
  - sadp-api
  http:
  - match:
    - headers:
        version:
          exact: "v2"
    route:
    - destination:
        host: sadp-api
        subset: v2
      weight: 20
    - destination:
        host: sadp-api
        subset: v1
      weight: 80
  - route:
    - destination:
        host: sadp-api
        subset: v1

---
# Circuit Breaker Configuration
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: sadp-circuit-breaker
spec:
  host: sadp-api
  trafficPolicy:
    circuitBreaker:
      consecutiveErrors: 5
      interval: 30s
      baseEjectionTime: 30s
      maxEjectionPercent: 50
    retryPolicy:
```

```
    attempts: 3
    perTryTimeout: 2s
```

Sarthi AI Agent Development Platform (SADP)

Google Healthcare Platform Integration Architecture

Overview

The Sarthi AI Agent Development Platform (SADP) is a standalone, enterprise-grade microservice built specifically for **Google Cloud Healthcare APIs** and **Google AI (Gemini)** models. It provides comprehensive AI agent development, evaluation, and management capabilities as an external API service for the main Sarthi healthcare application, leveraging Google's healthcare-specific infrastructure and AI capabilities.

Google Cloud Healthcare Architecture

High-Level Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│              Sarthi Healthcare Application                     │
├─────────────────────────────────────────────────────────────────┤
│ Patient Portal │ Provider Dashboard │ Admin Console │ Mobile │
└────────────────┬────────────────────────┬───────────────────────┘
       │              │
       ▼              ▼
┌─────────────────────────────────────────────────────────────────┐
│      Google Cloud Load Balancer + Cloud Armor (WAF)        │
├─────────────────────────────────────────────────────────────────┤
│            Google Cloud API Gateway                       │
└────────────────┬────────────────────────┬───────────────────────┘
       │              │
       ▼              ▼
┌─────────────────────────────────────────────────────────────────┐
│        Sarthi AI Agent Development Platform (SADP)         │
│              (Google Cloud Healthcare API)                │
├─────────────────────────────────────────────────────────────────┤
│              Google Service Mesh (Istio)                 │
├────────────────┬──────────────┬──────────────┬──────────────────┤
│ Agent Runtime │ Evaluation │ Development │ Monitoring &   │
│ Service       │ Service    │ Service     │ Analytics     │
├────────────────┼──────────────┼──────────────┼──────────────────┤
│ • Gemini Exec │ • Test Gen  │ • Prompt Mgmt │ • Cloud Mon     │
│ • Healthcare  │ • Evaluation │ • A/B Testing │ • Real-time Mon  │
│   API Integ   │ • Benchmarks │ • CI/CD      │ • Alerts        │
│ • FHIR Proc   │ • Compliance │ • Deployment │ • Analytics      │
│ • HL7 Support │ • Regression │ • Version Mgmt│ • Cost Tracking  │
│ • Circuit     │   Testing   │ • Blue/Green │ • Anomaly Detect │
│   Breakers    │             │             │                 │
└────────────────┴──────────────┴──────────────┴──────────────────┘
                   │
                   ▼
┌─────────────────────────────────────────────────────────────────┐
│           Google Cloud Healthcare APIs                    │
├────────────────┬──────────────┬──────────────┬──────────────────┤
│  FHIR Store    │ DICOM Store │ HL7v2 Store │ Consent APIs    │
│  (R4/STU3)     │ (Medical    │ (Legacy     │ (Privacy Mgmt)  │
│               │ Imaging)    │ Messages)   │               │
└────────────────┴──────────────┴──────────────┴──────────────────┘
                   │
                   ▼
┌─────────────────────────────────────────────────────────────────┐
```

```
│              Google AI & Data Services              │
├─────────────────┬─────────────────┬─────────────────┬─────────────────┤
│ Gemini Pro │ AutoML │ Healthcare │ Vertex AI │
│ (Primary AI) │ (Custom │ NLP API │ (ML Ops) │
│ │ Models) │ (Medical │ │
│ │ │ Entity) │ │
└─────────────────┴─────────────────┴─────────────────┴─────────────────┘
                    │
                    ▼
┌─────────────────────────────────────────────────────────────────────┐
│              Google Cloud Infrastructure              │
├─────────────────┬─────────────────┬─────────────────┬─────────────────┤
│ Database │ Message │ Storage │ Security & │
│ Services │ Services │ Services │ Compliance │
├─────────────────┼─────────────────┼─────────────────┼─────────────────┤
│ • Cloud SQL │ • Pub/Sub │ • Cloud │ • Cloud IAM │
│ (PostgreSQL) │ (Healthcare │ Storage │ • Cloud KMS │
│ • Firestore │ Events) │ • Filestore │ • VPC Service │
│ (Documents) │ • Cloud Tasks │ • Memorystore │ Controls │
│ • BigQuery │ (Workflows) │ (Redis) │ • Cloud Armor │
│ (Analytics) │ • Eventarc │ • BigTable │ • Binary Auth │
│ • Spanner │ (Events) │ (Time │ • Cloud Audit │
│ (Global DB) │ │ Series) │ Logs │
└─────────────────┴─────────────────┴─────────────────┴─────────────────┘
```

Google Cloud Healthcare Platform Integration

1. Google Cloud Healthcare APIs Integration

python

```python
# Google Cloud Healthcare API Integration
class GoogleHealthcareAPIClient:
    """Native integration with Google Cloud Healthcare APIs"""

    def __init__(self, project_id: str, location: str, dataset_id: str):
        self.project_id = project_id
        self.location = location
        self.dataset_id = dataset_id
        self.healthcare_client = healthcare.HealthcareServiceClient()
        self.fhir_client = self.setup_fhir_client()
        self.hl7v2_client = self.setup_hl7v2_client()
        self.dicom_client = self.setup_dicom_client()
        self.consent_client = self.setup_consent_client()

    def setup_fhir_client(self) -> FHIRStoreClient:
        """Setup FHIR R4 store client for structured healthcare data"""
        fhir_store_path = self.healthcare_client.fhir_store_path(
            self.project_id, self.location, self.dataset_id, "sarthi-fhir-store"
        )
        return FHIRStoreClient(fhir_store_path)

    async def store_patient_data(self, patient_resource: dict) -> str:
        """Store patient data in FHIR format"""
        try:
            # Validate FHIR resource
            validated_resource = await self.validate_fhir_resource(
                patient_resource, "Patient"
            )

            # Store in Google Cloud Healthcare FHIR store
            response = await self.fhir_client.create_resource(
                parent=self.fhir_store_path,
                body=validated_resource
            )

            # Log to Cloud Audit Logs
            await self.log_healthcare_data_access(
                operation="create_patient",
                resource_id=response.name,
                data_classification="PHI"
            )

            return response.name
        except Exception as e:
            await self.handle_healthcare_api_error(e, "store_patient_data")
            raise

    async def retrieve_patient_consent(self, patient_id: str) -> ConsentRecord:
        """Retrieve patient consent using Google Cloud Healthcare Consent API"""
        try:
            consent_store_path = self.healthcare_client.consent_store_path(
                self.project_id, self.location, self.dataset_id, "sarthi-consent-store"
            )

            # Query consent records
            consent_response = await self.consent_client.list_consents(
                parent=consent_store_path,
                filter=f'user_id="{patient_id}"'
            )

            return ConsentRecord.from_google_response(consent_response)

        except Exception as e:
            await self.handle_healthcare_api_error(e, "retrieve_patient_consent")
            raise

    async def process_hl7v2_message(self, hl7_message: str) -> ProcessedMessage:
        """Process HL7v2 messages for legacy system integration"""
        try:
            hl7v2_store_path = self.healthcare_client.hl7v2_store_path(
                self.project_id, self.location, self.dataset_id, "sarthi-hl7v2-store"
            )

            # Ingest HL7v2 message
            message_response = await self.hl7v2_client.ingest_message(
                parent=hl7v2_store_path,
                message={"data": base64.b64encode(hl7_message.encode()).decode()}
            )

            # Extract structured data
            parsed_data = await self.parse_hl7v2_message(message_response)

            return ProcessedMessage(
                message_id=message_response.name,
                parsed_data=parsed_data,
                processing_timestamp=datetime.utcnow()
            )

        except Exception as e:
            await self.handle_healthcare_api_error(e, "process_hl7v2_message")
            raise

# Google Cloud Healthcare Data Pipeline
class HealthcareDataPipeline:
    """Healthcare data processing pipeline using Google Cloud services"""

    def __init__(self):
        self.healthcare_client = GoogleHealthcareAPIClient(
            project_id="sarthi-healthcare-platform",
            location="us-central1",
            dataset_id="sarthi-production-dataset"
        )
        self.nlp_client = HealthcareNLPClient()
        self.automl_client = AutoMLClient()

    async def process_clinical_document(self, document: ClinicalDocument) -> ProcessedClinicalData:
        """Process clinical documents using Google Healthcare NLP API"""

        # Extract medical entities using Healthcare NLP API
        nlp_response = await self.nlp_client.analyze_entities(
            document_content=document.content,
            license_type="HEALTHCARE"
        )

        # Extract structured data
        medical_entities = self.extract_medical_entities(nlp_response)

        # Store in FHIR format
        fhir_resources = await self.convert_to_fhir_resources(medical_entities)

        stored_resources = []
        for resource in fhir_resources:
            resource_id = await self.healthcare_client.store_fhir_resource(resource)
            stored_resources.append(resource_id)

        return ProcessedClinicalData(
            document_id=document.id,
            medical_entities=medical_entities,
            fhir_resources=stored_resources,
            processing_timestamp=datetime.utcnow()
    )

class HealthcareNLPClient:
    """Google Cloud Healthcare NLP API client"""

    def __init__(self):
        self.nlp_service = language.LanguageServiceClient()

    async def analyze_entities(self, document_content: str,
                               license_type: str = "HEALTHCARE") -> dict:
        """Analyze medical entities in clinical text"""

        # Configure healthcare-specific NLP
        document = language.Document(
            content=document_content,
            type_=language.Document.Type.PLAIN_TEXT,
            language="en"
        )

        # Use healthcare-licensed NLP
        features = language.AnnotateTextRequest.Features(
            extract_entities=True,
            extract_entity_sentiment=False,
            extract_syntax=False,
            classify_text=False,
            extract_document_sentiment=False
        )

        response = await self.nlp_service.annotate_text(
            request={
                "document": document,
                "features": features,
                "encoding_type": language.EncodingType.UTF8
            }
        )

        # Extract healthcare-specific entities
        medical_entities = self.extract_healthcare_entities(response.entities)

        return {
            "entities": medical_entities,
            "confidence_scores": self.calculate_confidence_scores(response.entities),
            "processing_metadata": {
                "api_version": "healthcare_nlp_v1",
            "license_type": license_type,
            "language": "en"
        }
    }
```

2. Google AI (Gemini) Integration

```
python
```

```python
# Google AI (Gemini) Integration for Healthcare
class GoogleAIGeminiClient:
    """Native integration with Google AI Gemini models for healthcare"""

    def __init__(self):
        self.vertex_ai_client = aiplatform.gapic.PredictionServiceClient()
        self.project_id = "sarthi-healthcare-platform"
        self.location = "us-central1"
        self.gemini_endpoint = self.setup_gemini_endpoint()

    def setup_gemini_endpoint(self) -> str:
        """Setup Gemini Pro endpoint for healthcare applications"""
        return f"projects/{self.project_id}/locations/{self.location}/publishers/google/models/gemini-pro-healthcare"

    async def execute_healthcare_agent(self, agent_config: HealthcareAgentConfig,
                                      input_data: dict) -> GeminiResponse:
        """Execute healthcare AI agent using Gemini Pro"""

        try:
            # Prepare healthcare-specific prompt
            healthcare_prompt = await self.prepare_healthcare_prompt(
                agent_config, input_data
            )

            # Configure Gemini for healthcare use
            gemini_request = {
                "endpoint": self.gemini_endpoint,
                "instances": [{
                    "prompt": healthcare_prompt,
                    "parameters": {
                        "temperature": agent_config.temperature,
                        "max_output_tokens": agent_config.max_tokens,
                        "top_p": agent_config.top_p,
                        "top_k": agent_config.top_k,
                        "safety_settings": self.get_healthcare_safety_settings(),
                        "healthcare_mode": True,  # Enable healthcare-specific features
                        "phi_protection": True,   # Enable PHI protection
                        "medical_accuracy": "high"  # Prioritize medical accuracy
                    }
                }]
            }

            # Execute with healthcare monitoring
            start_time = time.time()
            response = await self.vertex_ai_client.predict(
                request=gemini_request
            )
            execution_time = (time.time() - start_time) * 1000

            # Validate healthcare response
            validated_response = await self.validate_healthcare_response(
                response, agent_config
            )

            # Log healthcare AI usage
            await self.log_healthcare_ai_usage(
                agent_config=agent_config,
                input_data=input_data,
                response=validated_response,
                execution_time=execution_time
            )

            return GeminiResponse(
                content=validated_response.predictions[0]["content"],
                safety_ratings=validated_response.predictions[0]["safety_ratings"],
                execution_time=execution_time,
                model_version="gemini-pro-healthcare",
                healthcare_validated=True,
                phi_detected=validated_response.phi_analysis.phi_detected,
                medical_accuracy_score=validated_response.medical_accuracy_score
            )

        except Exception as e:
            await self.handle_gemini_error(e, agent_config)
            raise

    def get_healthcare_safety_settings(self) -> list:
        """Get healthcare-specific safety settings for Gemini"""
        return [
            {
                "category": "HARM_CATEGORY_MEDICAL_ADVICE",
                "threshold": "BLOCK_MEDIUM_AND_ABOVE"
            },
            {
                "category": "HARM_CATEGORY_HEALTH_CLAIMS",
                "threshold": "BLOCK_LOW_AND_ABOVE"
            },
            {
                "category": "HARM_CATEGORY_PHI_EXPOSURE",
                "threshold": "BLOCK_NONE"  # We handle PHI separately
            },
            {
                "category": "HARM_CATEGORY_MEDICAL_MISINFORMATION",
                "threshold": "BLOCK_LOW_AND_ABOVE"
            }
        ]

    async def prepare_healthcare_prompt(self, agent_config: HealthcareAgentConfig,
                                      input_data: dict) -> str:
        """Prepare healthcare-optimized prompt for Gemini"""

        base_prompt = f"""
        You are a Google AI assistant specialized in healthcare applications for the Sarthi platform.

        Healthcare Context:
        - Agent Type: {agent_config.agent_type}
        - Medical Specialty: {agent_config.medical_specialty}
        - Compliance Level: HIPAA, GDPR healthcare
        - Safety Level: Maximum medical safety protocols

        Instructions:
        {agent_config.instructions}

        Medical Guidelines:
        - Follow evidence-based medical practices
        - Cite relevant medical literature when applicable
        - Maintain patient confidentiality and privacy
        - Never provide definitive diagnoses
        - Always recommend healthcare provider consultation
        - Use appropriate medical terminology

        Input Data:
        {json.dumps(input_data, indent=2)}

        Please provide a response following these healthcare standards and the specific agent instructions.
        """

        return base_prompt

    async def validate_healthcare_response(self, response: any,
                                         agent_config: HealthcareAgentConfig) -> ValidatedResponse:
```

```
        """Validate Gemini response for healthcare compliance"""

        validation_result = ValidatedResponse()

        # Extract response content
        response_content = response.predictions[0]["content"]

        # PHI detection and protection
        phi_analysis = await self.analyze_phi_in_response(response_content)
        validation_result.phi_analysis = phi_analysis

        if phi_analysis.phi_detected and not agent_config.allow_phi_output:
            # Redact PHI from response
            response_content = await self.redact_phi_from_response(
                response_content, phi_analysis.phi_locations
            )

        # Medical accuracy validation
        medical_accuracy_score = await self.validate_medical_accuracy(
            response_content, agent_config.medical_specialty
        )
        validation_result.medical_accuracy_score = medical_accuracy_score

        # Safety validation
        safety_validation = await self.validate_healthcare_safety(response_content)
        validation_result.safety_validation = safety_validation

        # Update response with validated content
        response.predictions[0]["content"] = response_content
        validation_result.predictions = response.predictions

        return validation_result

# Google AI Model Manager for Healthcare
class GoogleAIModelManager:
    """Manage Google AI models for healthcare applications"""

    def __init__(self):
        self.vertex_ai = aiplatform
        self.model_registry = VertexAIModelRegistry()
        self.automl_client = AutoMLClient()

    async def deploy_custom_healthcare_model(self, model_config: HealthcareModelConfig) -> str:
        """Deploy custom healthcare model using Vertex AI"""
        # Create custom healthcare model
        model = aiplatform.Model.upload(
            display_name=model_config.model_name,
            artifact_uri=model_config.model_artifact_uri,
            serving_container_image_uri=model_config.container_image_uri,
            serving_container_health_route="/health",
            serving_container_predict_route="/predict",
            labels={
                "healthcare": "true",
                "hipaa_compliant": "true",
                "medical_specialty": model_config.medical_specialty,
                "sarthi_agent": model_config.agent_type
            }
        )

        # Deploy with healthcare-specific configuration
        endpoint = model.deploy(
            endpoint_display_name=f"{model_config.model_name}-endpoint",
            machine_type="n1-standard-4",
            min_replica_count=2,
            max_replica_count=10,
            accelerator_type="NVIDIA_TESLA_T4",
            accelerator_count=1,
            encryption_spec_key_name=model_config.kms_key_name,  # Healthcare encryption
            enable_request_response_logging=True,  # For compliance
            request_response_logging_sampling_rate=1.0  # Log all for audit
        )

        # Configure healthcare monitoring
        await self.setup_healthcare_model_monitoring(endpoint, model_config)

        return endpoint.resource_name

    async def setup_healthcare_model_monitoring(self, endpoint: aiplatform.Endpoint,
                                              model_config: HealthcareModelConfig):
        """Setup comprehensive monitoring for healthcare models"""

        # Create model monitoring job
        monitoring_job = aiplatform.ModelDeploymentMonitoringJob.create(
            display_name=f"{model_config.model_name}-monitoring",
            endpoint=endpoint,
            logging_sampling_strategy=aiplatform.SamplingStrategy(
                random_sample_config=aiplatform.RandomSampleConfig(
                sample_rate=1.0  # Monitor all requests for healthcare
            )
        ),
        model_deployment_monitoring_objective_configs=[
            aiplatform.ModelDeploymentMonitoringObjectiveConfig(
                deployed_model_id=endpoint.list_models()[0].id,
                objective_config=aiplatform.ModelMonitoringObjectiveConfig(
                    training_dataset=aiplatform.InputDataConfig(
                        dataset=model_config.training_dataset_uri
                    ),
                    training_prediction_skew_detection_config=aiplatform.TrainingPredictionSkewDetectionConfig(
                        skew_thresholds={
                            "medical_accuracy": 0.05,  # Low tolerance for medical accuracy drift
                            "confidence_score": 0.1,
                            "response_quality": 0.05
                        }
                    )
                )
            )
        ],
        model_deployment_monitoring_schedule_config=aiplatform.ModelDeploymentMonitoringScheduleConfig(
            monitor_interval=3600  # Check every hour for healthcare models
        )
    )

    return monitoring_job

# Google Cloud Healthcare Event Processing
class HealthcareEventProcessor:
    """Process healthcare events using Google Cloud Pub/Sub and Eventarc"""

    def __init__(self):
        self.publisher_client = pubsub_v1.PublisherClient()
        self.subscriber_client = pubsub_v1.SubscriberClient()
        self.healthcare_topic = f"projects/{PROJECT_ID}/topics/sarthi-healthcare-events"

    async def publish_healthcare_event(self, event: HealthcareEvent) -> str:
        """Publish healthcare event to Google Cloud Pub/Sub"""

        # Prepare healthcare event message
        event_message = {
            "event_id": str(uuid.uuid4()),
            "event_type": event.event_type,
            "timestamp": datetime.utcnow().isoformat(),
            "patient_id": event.patient_id,
            "facility_id": event.facility_id,
            "data_classification": event.data_classification,
            "phi_level": event.phi_level,
            "event_data": event.data
        }

        # Encrypt PHI if present
        if event.phi_level > 0:
            event_message["event_data"] = await self.encrypt_healthcare_data(
                event.data, event.kms_key_name
            )

        # Publish with healthcare attributes
        message_data = json.dumps(event_message).encode("utf-8")
        future = self.publisher_client.publish(
            self.healthcare_topic,
            message_data,
            event_type=event.event_type,
            phi_level=str(event.phi_level),
            facility_id=event.facility_id,
            data_classification=event.data_classification
        )

        message_id = await future.result()

        # Log healthcare event publication
        await self.log_healthcare_event_publication(event, message_id)

        return message_id

    async def process_healthcare_event(self, message: pubsub_v1.PubsubMessage) -> ProcessingResult:
        """Process incoming healthcare event"""

        try:
            # Parse healthcare event
            event_data = json.loads(message.data.decode("utf-8"))

            # Validate healthcare event structure
            validated_event = await self.validate_healthcare_event(event_data)

            # Decrypt PHI if present
            if validated_event.phi_level > 0:
                decrypted_data = await self.decrypt_healthcare_data(
                validated_event.event_data, validated_event.kms_key_name
            )
            validated_event.event_data = decrypted_data

        # Route to appropriate healthcare processor
        processor_result = await self.route_healthcare_event(validated_event)

        # Log successful processing
        await self.log_healthcare_event_processing(validated_event, processor_result)

        return ProcessingResult(
            success=True,
            event_id=validated_event.event_id,
            processing_time=processor_result.processing_time,
            output_data=processor_result.output_data
        )

    except Exception as e:
        await self.handle_healthcare_event_error(e, message)
        raise
```

3. Google Cloud Technology Stack

yaml

```yaml
# Google Cloud Healthcare Technology Stack
google_cloud_services:
  compute_and_containers:
    primary: "Google Kubernetes Engine (GKE Autopilot)"
    serverless: "Cloud Run (Healthcare compliant)"
    functions: "Cloud Functions 2nd Gen"
    batch_processing: "Cloud Batch"

  ai_and_ml:
    primary_llm: "Gemini Pro (Healthcare)"
    secondary_llm: "Gemini Pro Vision (Medical Imaging)"
    custom_models: "Vertex AI Custom Models"
    automl: "Vertex AI AutoML (Healthcare)"
    nlp: "Healthcare Natural Language API"
    document_ai: "Document AI (Healthcare)"

  healthcare_apis:
    fhir_store: "Cloud Healthcare FHIR Store (R4/STU3)"
    dicom_store: "Cloud Healthcare DICOM Store"
    hl7v2_store: "Cloud Healthcare HL7v2 Store"
    consent_management: "Cloud Healthcare Consent API"
    de_identification: "Cloud Healthcare De-identification API"

  databases_and_storage:
    primary_db: "Cloud SQL for PostgreSQL (Healthcare)"
    document_db: "Firestore (Healthcare mode)"
    analytics_db: "BigQuery (Healthcare)"
    global_db: "Cloud Spanner (Multi-region)"
    cache: "Memorystore for Redis (HA)"
    time_series: "Cloud Bigtable"
    object_storage: "Cloud Storage (Healthcare)"

  messaging_and_events:
    pub_sub: "Cloud Pub/Sub (Healthcare events)"
    workflows: "Cloud Workflows (Healthcare processes)"
    tasks: "Cloud Tasks (Background jobs)"
    eventarc: "Eventarc (Event-driven architecture)"
    scheduler: "Cloud Scheduler"

  security_and_compliance:
    identity: "Cloud Identity and Access Management (IAM)"
    key_management: "Cloud Key Management Service (KMS)"
    secret_management: "Secret Manager"
    vpc_security: "VPC Service Controls"
    armor: "Cloud Armor (WAF)"
    binary_authorization: "Binary Authorization"
    audit_logs: "Cloud Audit Logs"
    dlp: "Cloud Data Loss Prevention API"

  monitoring_and_observability:
    monitoring: "Cloud Monitoring (Healthcare dashboards)"
    logging: "Cloud Logging (Healthcare audit)"
    tracing: "Cloud Trace"
    profiler: "Cloud Profiler"
    error_reporting: "Error Reporting"

  networking:
    load_balancer: "Cloud Load Balancing"
    cdn: "Cloud CDN"
    dns: "Cloud DNS"
    interconnect: "Cloud Interconnect (Healthcare partners)"

  development_and_deployment:
    build: "Cloud Build (Healthcare CI/CD)"
    deploy: "Cloud Deploy"
    artifact_registry: "Artifact Registry"
    source_repos: "Cloud Source Repositories"

# Google Cloud Healthcare Configuration
google_healthcare_config:
  project_id: "sarthi-healthcare-platform"
  location: "us-central1"  # HIPAA compliant region

  healthcare_dataset:
    name: "sarthi-production-dataset"
    time_zone: "UTC"

  fhir_stores:
    - name: "sarthi-fhir-r4-store"
      version: "R4"
      enable_update_create: true
      disable_referential_integrity: false

  dicom_stores:
    - name: "sarthi-dicom-store"
      notification_config:
        pubsub_topic: "projects/sarthi-healthcare-platform/topics/dicom-events"
```

```
hl7v2_stores:
  - name: "sarthi-hl7v2-store"
    parser_config:
      allow_null_header: false
      segment_terminator: "\\r"

consent_stores:
  - name: "sarthi-consent-store"
    enable_consent_create_on_update: true
    default_consent_ttl: "315360000s"  # 10 years

security_config:
  kms_key_name: "projects/sarthi-healthcare-platform/locations/us-central1/keyRings/sarthi-healthcare/cryptoKeys/sa
  audit_log_config:
    log_type: "ADMIN_READ"
    exempted_members: []
```

4. Google Cloud Deployment Configuration

```
yaml
```

```yaml
# Google Cloud Healthcare Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sadp-api
  namespace: sarthi-healthcare
  labels:
    app: sadp-api
    healthcare: "true"
    hipaa-compliant: "true"
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sadp-api
  template:
    metadata:
      labels:
        app: sadp-api
        healthcare: "true"
      annotations:
        # Google Cloud specific annotations
        run.googleapis.com/vpc-access-connector: "sarthi-vpc-connector"
        run.googleapis.com/vpc-access-egress: "private-ranges-only"
    spec:
      serviceAccountName: sadp-healthcare-service-account
      securityContext:
        runAsNonRoot: true
        runAsUser: 65534
        fsGroup: 65534
      containers:
      - name: sadp-api
        image: gcr.io/sarthi-healthcare-platform/sadp-api:v1.0.0
        ports:
        - containerPort: 8080
          name: http
        env:
        # Google Cloud Healthcare API configuration
        - name: GOOGLE_CLOUD_PROJECT
          value: "sarthi-healthcare-platform"
        - name: HEALTHCARE_DATASET_ID
          value: "sarthi-production-dataset"
        - name: HEALTHCARE_LOCATION
```

```
 value: "us-central1"

# Gemini AI configuration
- name: GEMINI_MODEL_ENDPOINT
 value: "projects/sarthi-healthcare-platform/locations/us-central1/publishers/google/models/gemini-pro-healthca
- name: VERTEX_AI_PROJECT
 value: "sarthi-healthcare-platform"
- name: VERTEX_AI_LOCATION
 value: "us-central1"

# Google Cloud services configuration
- name: CLOUD_SQL_CONNECTION_NAME
 valueFrom:
  secretKeyRef:
   name: sarthi-db-secret
   key: connection-name
- name: REDIS_HOST
 valueFrom:
  secretKeyRef:
   name: sarthi-redis-secret
   key: host
- name: PUBSUB_TOPIC
 value: "projects/sarthi-healthcare-platform/topics/sarthi-healthcare-events"

# Security and compliance
- name: KMS_KEY_NAME
 value: "projects/sarthi-healthcare-platform/locations/us-central1/keyRings/sarthi-healthcare/cryptoKeys/sarthi-p
- name: HEALTHCARE_COMPLIANCE_MODE
 value: "HIPAA"

resources:
 requests:
  memory: "1Gi"
  cpu: "500m"
 limits:
  memory: "2Gi"
  cpu: "1000m"

livenessProbe:
 httpGet:
  path: /health
  port: 8080
 initialDelaySeconds: 30
 periodSeconds: 10
```

```yaml
        readinessProbe:
          httpGet:
            path: /ready
            port: 8080
          initialDelaySeconds: 5
          periodSeconds: 5

        # Google Cloud security
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          readOnlyRootFilesystem: true

---
# Google Cloud specific service configuration
apiVersion: v1
kind: Service
metadata:
  name: sadp-api-service
  namespace: sarthi-healthcare
  annotations:
    cloud.google.com/backend-config: '{"default": "sadp-backend-config"}'
    cloud.google.com/load-balancer-type: "External"
spec:
  type: LoadBalancer
  loadBalancerSourceRanges:
  - "10.0.0.0/8"  # Internal VPC only
  selector:
    app: sadp-api
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP

---
# Google Cloud Backend Configuration
apiVersion: cloud.google.com/v1
kind: BackendConfig
metadata:
  name: sadp-backend-config
  namespace: sarthi-healthcare
```

```
spec:
  healthCheck:
    checkIntervalSec: 10
    timeoutSec: 5
    healthyThreshold: 2
    unhealthyThreshold: 3
    type: HTTP
    requestPath: /health
  sessionAffinity:
    affinityType: "CLIENT_IP"
  timeoutSec: 300
  connectionDraining:
    drainingTimeoutSec: 60
  # Healthcare-specific security
  securityPolicy:
    name: "sarthi-healthcare-security-policy"
  iap:
    enabled: true
    oauthclientCredentials:
      secretName: "oauth-client-secret"

---
# Google Cloud IAM Service Account
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sadp-healthcare-service-account
  namespace: sarthi-healthcare
  annotations:
    iam.gke.io/gcp-service-account: sadp-healthcare@sarthi-healthcare-platform.iam.gserviceaccount.com
```

4. Advanced API Specifications with OpenAPI 3.0

```
yaml
```

```yaml
openapi: 3.0.3
info:
  title: Sarthi AI Agent Development Platform API
  version: 1.0.0
  description: |
    Enterprise-grade API for AI agent management, evaluation, and execution.
    Supports healthcare-specific workflows with HIPAA compliance.
  contact:
    name: Sarthi Platform Team
    url: https://sarthi.com/support
    email: support@sarthi.com
  license:
    name: Commercial License
    url: https://sarthi.com/license

servers:
  - url: https://sadp.sarthi.com/api/v1
    description: Production server
  - url: https://sadp-staging.sarthi.com/api/v1
    description: Staging server

security:
  - BearerAuth: []
  - ApiKeyAuth: []

paths:
  /agents/{agentName}/execute:
    post:
      summary: Execute AI agent
      description: |
        Execute a specific AI agent with input data and configuration.
        Supports healthcare-specific agents with HIPAA compliance.
      operationId: executeAgent
      tags:
        - Agent Runtime
      parameters:
        - name: agentName
          in: path
          required: true
          schema:
            type: string
            enum: [
              "document_processor", "clinical_agent", "billing_agent",
          "voice_agent", "health_assistant", "medication_entry",
          "referral_processing", "lab_result_entry"
        ]
      example: "clinical_agent"
    - name: X-Trace-ID
      in: header
      schema:
        type: string
        format: uuid
      description: Unique trace ID for request tracking
    - name: X-Facility-ID
      in: header
      schema:
        type: string
      description: Healthcare facility identifier
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/AgentExecutionRequest'
        examples:
          clinical_agent_example:
            summary: Clinical agent execution
            value:
              input_data:
                patient_id: "SARTHI-PT-001"
                primary_diagnosis: "Type 2 Diabetes Mellitus"
                patient_age: 55
                comorbidities: ["Hypertension", "Obesity"]
              context:
                facility_id: "SARTHI-CLINIC-001"
                provider_id: "DR-SMITH-001"
                session_id: "sess_123456"
              options:
                timeout: 30000
                priority: "high"
                trace_enabled: true
  responses:
    '200':
      description: Agent execution successful
      content:
        application/json:
          schema:
              $ref: '#/components/schemas/AgentExecutionResponse'
      '400':
        $ref: '#/components/responses/BadRequest'
      '401':
        $ref: '#/components/responses/Unauthorized'
      '429':
        $ref: '#/components/responses/RateLimited'
      '500':
        $ref: '#/components/responses/InternalError'

/workflows/execute:
  post:
    summary: Execute multi-agent workflow
    description: |
      Execute a complex workflow involving multiple AI agents
      with orchestration and dependency management.
    operationId: executeWorkflow
    tags:
      - Workflow Orchestration
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/WorkflowExecutionRequest'
    responses:
      '200':
        description: Workflow execution initiated
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/WorkflowExecutionResponse'
      '202':
        description: Workflow accepted for asynchronous processing
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/AsyncWorkflowResponse'

/evaluation/agents/{agentName}/test:
  post:
    summary: Run comprehensive agent evaluation
    description: |
      Execute comprehensive testing and evaluation of an AI agent
```

```
        against predefined test suites and benchmarks.
      operationId: evaluateAgent
      tags:
        - Evaluation & Testing
      parameters:
        - name: agentName
          in: path
          required: true
          schema:
            type: string
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/AgentEvaluationRequest'
      responses:
        '200':
          description: Evaluation completed
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/AgentEvaluationResponse'
        '202':
          description: Evaluation initiated asynchronously
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/AsyncEvaluationResponse'

  /monitoring/agents/{agentName}/metrics:
    get:
      summary: Get agent performance metrics
      description: |
        Retrieve real-time and historical performance metrics
        for a specific AI agent.
      operationId: getAgentMetrics
      tags:
        - Monitoring & Analytics
      parameters:
        - name: agentName
          in: path
          required: true
          schema:
```

```yaml
          type: string
        - name: timeRange
          in: query
          schema:
            type: string
            enum: ["1h", "24h", "7d", "30d"]
            default: "24h"
        - name: metrics
          in: query
          schema:
            type: array
            items:
              type: string
              enum: ["accuracy", "latency", "error_rate", "cost", "throughput"]
          style: form
          explode: false
        - name: aggregation
          in: query
          schema:
            type: string
            enum: ["mean", "median", "p95", "p99", "sum"]
            default: "mean"
      responses:
        '200':
          description: Metrics retrieved successfully
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/AgentMetricsResponse'

components:
  securitySchemes:
    BearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
    ApiKeyAuth:
      type: apiKey
      in: header
      name: X-API-Key

  schemas:
    AgentExecutionRequest:
      type: object
      required:
        - input_data
      properties:
        input_data:
          type: object
          description: Agent-specific input data
          additionalProperties: true
        context:
          $ref: '#/components/schemas/ExecutionContext'
        options:
          $ref: '#/components/schemas/ExecutionOptions'

    ExecutionContext:
      type: object
      properties:
        facility_id:
          type: string
          description: Healthcare facility identifier
        provider_id:
          type: string
          description: Healthcare provider identifier
        patient_id:
          type: string
          description: Patient identifier (if applicable)
        session_id:
          type: string
          description: Session identifier for tracking
        compliance_level:
          type: string
          enum: ["HIPAA", "GDPR", "STANDARD"]
          default: "HIPAA"

    ExecutionOptions:
      type: object
      properties:
        timeout:
          type: integer
          minimum: 1000
          maximum: 300000
          default: 30000
          description: Timeout in milliseconds
        priority:
          type: string
          enum: ["low", "normal", "high", "urgent"]
      default: "normal"
    trace_enabled:
      type: boolean
      default: false
      description: Enable detailed execution tracing
    cache_enabled:
      type: boolean
      default: true
      description: Enable response caching
    fallback_enabled:
      type: boolean
      default: true
      description: Enable fallback mechanisms

AgentExecutionResponse:
  type: object
  required:
    - execution_id
    - agent_name
    - status
    - result
  properties:
    execution_id:
      type: string
      format: uuid
      description: Unique execution identifier
    agent_name:
      type: string
      description: Name of the executed agent
    status:
      type: string
      enum: ["completed", "failed", "timeout", "cancelled"]
    execution_time:
      type: integer
      description: Execution time in milliseconds
    result:
      type: object
      description: Agent execution result
      additionalProperties: true
    metadata:
      $ref: '#/components/schemas/ExecutionMetadata'
    compliance:
      $ref: '#/components/schemas/ComplianceInfo'
    performance:
```

```yaml
            $ref: '#/components/schemas/PerformanceMetrics'
          error:
            $ref: '#/components/schemas/ErrorInfo'

    ExecutionMetadata:
      type: object
      properties:
        prompt_version:
          type: string
          description: Version of the prompt used
        model_info:
          type: object
          properties:
            provider:
              type: string
              example: "anthropic"
            model:
              type: string
              example: "claude-sonnet-4-20250514"
            version:
              type: string
        tokens_used:
          type: object
          properties:
            input:
              type: integer
            output:
              type: integer
            total:
              type: integer
        cost:
          type: number
          format: decimal
          description: Execution cost in USD
        cache_hit:
          type: boolean
          description: Whether response was served from cache

    ComplianceInfo:
      type: object
      properties:
        hipaa_compliant:
          type: boolean
          description: HIPAA compliance status
    audit_trail_id:
      type: string
      description: Audit trail identifier
    phi_processed:
      type: boolean
      description: Whether PHI was processed
    encryption_used:
      type: boolean
      description: Whether data was encrypted
    retention_policy:
      type: string
      description: Data retention policy applied

PerformanceMetrics:
  type: object
  properties:
    accuracy_score:
      type: number
      minimum: 0
      maximum: 100
      description: Accuracy score (0-100)
    confidence_score:
      type: number
      minimum: 0
      maximum: 1
      description: Confidence score (0-1)
    latency_percentile:
      type: string
      enum: ["p50", "p90", "p95", "p99"]
      description: Latency percentile classification
    quality_score:
      type: number
      minimum: 0
      maximum: 100
      description: Overall quality score

ErrorInfo:
  type: object
  properties:
    error_code:
      type: string
      description: Error code for categorization
    error_message:
      type: string
```

```yaml
        description: Human-readable error message
      error_details:
        type: object
        description: Detailed error information
        additionalProperties: true
      retry_possible:
        type: boolean
        description: Whether the operation can be retried
      suggested_action:
        type: string
        description: Suggested action to resolve the error

responses:
  BadRequest:
    description: Invalid request parameters
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ErrorResponse'

  Unauthorized:
    description: Authentication required
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ErrorResponse'

  RateLimited:
    description: Rate limit exceeded
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ErrorResponse'
    headers:
      Retry-After:
        schema:
          type: integer
        description: Seconds to wait before retrying

  InternalError:
    description: Internal server error
    content:
      application/json:
        schema:
```

```
        $ref: '#/components/schemas/ErrorResponse'

    ErrorResponse:
      type: object
      required:
        - error
        - message
        - timestamp
      properties:
        error:
          type: string
          description: Error type
        message:
          type: string
          description: Error message
        details:
          type: object
          description: Additional error details
          additionalProperties: true
        timestamp:
          type: string
          format: date-time
          description: Error timestamp
        trace_id:
          type: string
          description: Request trace ID for debugging
        support_id:
          type: string
          description: Support ticket ID for complex issues
```

5. Google Cloud Security & Compliance Framework

python

```python
# Google Cloud Healthcare Security Implementation
class GoogleCloudHealthcareSecurity:
    """Google Cloud native security for healthcare applications"""

    def __init__(self):
        self.iam_client = iam.IamPolicyManagementServiceClient()
        self.kms_client = kms.KeyManagementServiceClient()
        self.dlp_client = dlp.DlpServiceClient()
        self.audit_logger = GoogleCloudAuditLogger()
        self.vpc_sc_client = accesscontextmanager.AccessContextManagerClient()

    async def authenticate_with_google_iam(self, request: Request) -> GoogleServiceContext:
        """Google Cloud IAM-based authentication for healthcare APIs"""

        # Extract Google Cloud Identity token
        id_token = self.extract_google_identity_token(request)

        # Verify token with Google Cloud IAM
        try:
            # Validate JWT token
            claims = id_token.verify_oauth2_token(
                id_token,
                google.auth.transport.requests.Request(),
                audience=self.get_expected_audience()
            )

            # Get service account details
            service_account = await self.get_service_account_details(
                claims["email"]
            )

            # Validate healthcare permissions
            healthcare_roles = await self.get_healthcare_iam_roles(service_account)

            return GoogleServiceContext(
                service_account_email=claims["email"],
                project_id=claims["aud"].split("/")[1],
                iam_roles=healthcare_roles,
                healthcare_permissions=self.extract_healthcare_permissions(healthcare_roles),
                authenticated_at=datetime.utcnow()
            )

        except ValueError as e:
            await self.audit_logger.log_authentication_failure(
                error="invalid_token",
                request_details=self.sanitize_request_for_logging(request)
            )
            raise InvalidAuthenticationError("Invalid Google Cloud identity token")

    async def authorize_healthcare_action(self, context: GoogleServiceContext,
                                          resource: str, action: str) -> bool:
        """Google Cloud IAM-based authorization for healthcare operations"""

        # Check Google Cloud IAM permissions
        required_permission = f"healthcare.{resource}.{action}"

        try:
            # Test IAM permissions
            permissions_response = await self.iam_client.test_iam_permissions(
                resource=f"projects/{context.project_id}/locations/us-central1/datasets/sarthi-production-dataset",
                permissions=[required_permission]
            )

            has_permission = required_permission in permissions_response.permissions

            if not has_permission:
                await self.audit_logger.log_authorization_failure(
                    service_account=context.service_account_email,
                    resource=resource,
                    action=action,
                    reason="insufficient_iam_permissions"
                )
                return False

            # Additional healthcare-specific checks
            if not await self.validate_healthcare_context(context, resource, action):
                return False

            return True

        except Exception as e:
            await self.audit_logger.log_authorization_error(e, context)
            return False

    async def encrypt_phi_with_google_kms(self, phi_data: dict,
                                          patient_id: str) -> EncryptedData:
        """Encrypt PHI using Google Cloud KMS with healthcare keys"""
```

```
# Get healthcare KMS key
key_name = f"projects/{self.project_id}/locations/us-central1/keyRings/sarthi-healthcare/cryptoKeys/sarthi-phi-key

# Detect PHI fields
phi_fields = await self.detect_phi_with_google_dlp(phi_data)

encrypted_data = phi_data.copy()
encryption_metadata = []

for field_path, field_value in phi_fields.items():
    # Encrypt each PHI field separately
    plaintext = json.dumps(field_value).encode('utf-8')

    # Additional authenticated data for context
    additional_data = f"patient_id:{patient_id},field:{field_path}".encode('utf-8')

    # Encrypt with Google Cloud KMS
    encrypt_response = await self.kms_client.encrypt(
        request={
            "name": key_name,
            "plaintext": plaintext,
            "additional_authenticated_data": additional_data
        }
    )

    # Store encrypted data
    encrypted_data[field_path] = {
        "encrypted_value": base64.b64encode(encrypt_response.ciphertext).decode('utf-8'),
        "key_version": encrypt_response.name,
        "encryption_algorithm": "GOOGLE_SYMMETRIC_ENCRYPTION"
    }

    encryption_metadata.append({
        "field_path": field_path,
        "key_version": encrypt_response.name,
        "patient_id": patient_id,
        "encrypted_at": datetime.utcnow().isoformat()
    })

# Log encryption for audit
await self.audit_logger.log_phi_encryption(
    patient_id=patient_id,
    fields_encrypted=list(phi_fields.keys()),
            key_version=key_name,
            encryption_metadata=encryption_metadata
        )

        return EncryptedData(
            encrypted_data=encrypted_data,
            encryption_metadata=encryption_metadata,
            phi_fields_count=len(phi_fields)
        )

    async def detect_phi_with_google_dlp(self, data: dict) -> dict:
        """Detect PHI using Google Cloud Data Loss Prevention API"""

        # Configure DLP for healthcare PHI detection
        inspect_config = {
            "info_types": [
                {"name": "PERSON_NAME"},
                {"name": "PHONE_NUMBER"},
                {"name": "EMAIL_ADDRESS"},
                {"name": "US_SOCIAL_SECURITY_NUMBER"},
                {"name": "DATE_OF_BIRTH"},
                {"name": "MEDICAL_RECORD_NUMBER"},
                {"name": "US_HEALTHCARE_NPI"},
                {"name": "CREDIT_CARD_NUMBER"},
                {"name": "US_BANK_ACCOUNT"},
                {"name": "IBAN_CODE"},
                # Healthcare-specific info types
                {"name": "US_DEA_NUMBER"},
                {"name": "ICD9_CODE"},
                {"name": "ICD10_CODE"},
                {"name": "US_PASSPORT"},
                {"name": "US_DRIVERS_LICENSE_NUMBER"}
            ],
            "custom_info_types": [
                {
                    "info_type": {"name": "PATIENT_ID"},
                    "regex": {
                        "pattern": r"SARTHI-PT-\d{6}"
                    }
                }
            ],
            "min_likelihood": "POSSIBLE",
            "limits": {
                "max_findings_per_info_type": 100
            },
            "include_quote": True
        }

        # Prepare data for inspection
        content_item = {
            "value": json.dumps(data)
        }

        # Call Google Cloud DLP
        dlp_response = await self.dlp_client.inspect_content(
            request={
                "parent": f"projects/{self.project_id}/locations/global",
                "inspect_config": inspect_config,
                "item": content_item
            }
        )

        # Extract PHI findings
        phi_fields = {}
        for finding in dlp_response.result.findings:
            field_path = self.extract_field_path_from_quote(finding.quote, data)
            phi_fields[field_path] = {
                "info_type": finding.info_type.name,
                "likelihood": finding.likelihood.name,
                "quote": finding.quote,
                "location": finding.location
            }

        return phi_fields

    async def apply_google_vpc_service_controls(self, request: Request,
                                                context: GoogleServiceContext) -> bool:
        """Apply Google Cloud VPC Service Controls for healthcare data"""

        # Check if request is within authorized VPC Service Controls perimeter
        perimeter_name = f"accessPolicies/{self.access_policy_id}/servicePerimeters/sarthi-healthcare-perimeter"

        try:
            # Get current service perimeter
            perimeter = await self.vpc_sc_client.get_service_perimeter(
                name=perimeter_name
            )
            # Validate request context against perimeter
            request_context = {
                "origin": {
                    "ip_address": request.client.host,
                    "user_agent": request.headers.get("user-agent")
                },
                "destination": {
                    "service": "healthcare.googleapis.com",
                    "method": request.method,
                    "resource": request.url.path
                }
            }

            # Check perimeter restrictions
            if not self.validate_perimeter_access(perimeter, request_context, context):
                await self.audit_logger.log_vpc_service_control_violation(
                    perimeter_name=perimeter_name,
                    request_context=request_context,
                    service_context=context
                )
                return False

            return True

        except Exception as e:
            await self.audit_logger.log_vpc_service_control_error(e, context)
            return False

class GoogleCloudHealthcareCompliance:
    """Google Cloud native compliance for healthcare"""

    def __init__(self):
        self.audit_log_client = logging.Client()
        self.healthcare_client = healthcare.HealthcareServiceClient()
        self.bigquery_client = bigquery.Client()
        self.dlp_client = dlp.DlpServiceClient()

    async def log_healthcare_api_access(self, access_log: HealthcareAccessLog):
        """Log healthcare API access to Google Cloud Audit Logs"""

        # Prepare structured audit log entry
        audit_entry = {
            "protoPayload": {
                "@type": "type.googleapis.com/google.cloud.audit.AuditLog",
        "serviceName": "sarthi-sadp.googleapis.com",
        "methodName": f"/{access_log.api_version}/{access_log.method}",
        "resourceName": access_log.resource_name,
        "authenticationInfo": {
            "principalEmail": access_log.service_account_email,
            "serviceAccountKeyName": access_log.service_account_key_name
        },
        "authorizationInfo": [
            {
                "resource": access_log.resource_name,
                "permission": access_log.required_permission,
                "granted": access_log.permission_granted
            }
        ],
        "requestMetadata": {
            "callerIp": access_log.caller_ip,
            "callerSuppliedUserAgent": access_log.user_agent,
            "requestAttributes": {
                "time": access_log.timestamp.isoformat(),
                "reason": "HEALTHCARE_API_ACCESS"
            }
        },
        "request": {
            "healthcare_operation": access_log.operation_type,
            "phi_accessed": access_log.phi_accessed,
            "patient_count": access_log.patient_count,
            "data_classification": access_log.data_classification
        },
        "response": {
            "status": access_log.response_status,
            "processing_time_ms": access_log.processing_time
        }
    },
    "insertId": str(uuid.uuid4()),
    "resource": {
        "type": "healthcare_dataset",
        "labels": {
            "project_id": self.project_id,
            "location": "us-central1",
            "dataset_id": "sarthi-production-dataset"
        }
    },
    "timestamp": access_log.timestamp.isoformat(),
    "severity": "INFO",
        "labels": {
            "healthcare_compliance": "HIPAA",
            "phi_level": str(access_log.phi_level),
            "facility_id": access_log.facility_id
        }
    }

    # Write to Google Cloud Audit Logs
    self.audit_log_client.write_entries([audit_entry])

    # Also store in BigQuery for analytics
    await self.store_compliance_data_in_bigquery(access_log)

async def generate_hipaa_audit_report(self, report_config: HIPAAAuditReportConfig) -> AuditReport:
    """Generate HIPAA audit report using Google Cloud services"""

    # Query audit logs from BigQuery
    audit_query = f"""
    SELECT
        timestamp,
        protoPayload.authenticationInfo.principalEmail as service_account,
        protoPayload.resourceName as resource,
        protoPayload.request.healthcare_operation as operation,
        protoPayload.request.phi_accessed as phi_accessed,
        protoPayload.request.patient_count as patient_count,
        protoPayload.response.status as status,
        labels.facility_id as facility_id
    FROM `{self.project_id}.sarthi_audit_logs.healthcare_access_logs`
    WHERE timestamp >= @start_date
    AND timestamp <= @end_date
    AND labels.healthcare_compliance = 'HIPAA'
    ORDER BY timestamp DESC
    """

    query_job = self.bigquery_client.query(
        audit_query,
        job_config=bigquery.QueryJobConfig(
            query_parameters=[
                bigquery.ScalarQueryParameter("start_date", "TIMESTAMP", report_config.start_date),
                bigquery.ScalarQueryParameter("end_date", "TIMESTAMP", report_config.end_date)
            ]
        )
    )
```

```
audit_results = query_job.result()

# Analyze audit data
report_data = {
    "report_period": {
        "start_date": report_config.start_date.isoformat(),
        "end_date": report_config.end_date.isoformat()
    },
    "total_api_calls": 0,
    "phi_access_events": 0,
    "unique_patients_accessed": set(),
    "service_accounts_used": set(),
    "facilities_accessed": set(),
    "compliance_violations": [],
    "access_patterns": {}
}

for row in audit_results:
    report_data["total_api_calls"] += 1

    if row.phi_accessed:
        report_data["phi_access_events"] += 1
        report_data["unique_patients_accessed"].add(row.patient_count)

    report_data["service_accounts_used"].add(row.service_account)
    report_data["facilities_accessed"].add(row.facility_id)

    # Check for compliance violations
    if await self.detect_compliance_violation(row):
        report_data["compliance_violations"].append({
            "timestamp": row.timestamp,
            "violation_type": "UNAUTHORIZED_PHI_ACCESS",
            "service_account": row.service_account,
            "resource": row.resource
        })

# Convert sets to counts for JSON serialization
report_data["unique_patients_count"] = len(report_data["unique_patients_accessed"])
report_data["unique_service_accounts"] = len(report_data["service_accounts_used"])
report_data["unique_facilities"] = len(report_data["facilities_accessed"])

# Remove sets (not JSON serializable)
del report_data["unique_patients_accessed"]
del report_data["service_accounts_used"]
```

```python
        del report_data["facilities_accessed"]

        return AuditReport(
            report_id=str(uuid.uuid4()),
            generated_at=datetime.utcnow(),
            report_type="HIPAA_COMPLIANCE",
            data=report_data,
            compliance_status="COMPLIANT" if not report_data["compliance_violations"] else "VIOLATIONS_DETECTED"
        )

# Google Cloud Healthcare API Key Management
class GoogleCloudAPIKeyManager:
    """Google Cloud native API key management for healthcare services"""

    def __init__(self):
        self.api_keys_client = apikeys_v2.ApiKeysClient()
        self.iam_client = iam.IamPolicyManagementServiceClient()
        self.secret_manager_client = secretmanager.SecretManagerServiceClient()

    async def create_healthcare_api_key(self, key_config: HealthcareAPIKeyConfig) -> GoogleHealthcareAPIKey:
        """Create Google Cloud API key for healthcare services"""

        # Create API key with healthcare-specific restrictions
        api_key_request = {
            "parent": f"projects/{self.project_id}/locations/global",
            "api_key": {
                "display_name": key_config.display_name,
                "restrictions": {
                    "api_targets": [
                        {
                            "service": "healthcare.googleapis.com",
                            "methods": key_config.allowed_methods
                        },
                        {
                            "service": "aiplatform.googleapis.com",
                            "methods": ["predict", "explain"]
                        }
                    ],
                    "server_key_restrictions": {
                        "allowed_ips": key_config.allowed_ip_ranges
                    }
                },
                "annotations": {
                    "healthcare_compliant": "true",
                "facility_id": key_config.facility_id,
                "service_type": key_config.service_type,
                "phi_access_level": str(key_config.phi_access_level)
            }
        }
    }

    # Create the API key
    operation = await self.api_keys_client.create_key(request=api_key_request)
    api_key_response = await operation.result()

    # Store key metadata in Secret Manager
    secret_data = {
        "api_key_id": api_key_response.name,
        "key_string": api_key_response.key_string,
        "created_at": datetime.utcnow().isoformat(),
        "facility_id": key_config.facility_id,
        "phi_access_level": key_config.phi_access_level
    }

    secret_name = f"projects/{self.project_id}/secrets/sadp-api-key-{key_config.service_type}-{key_config.facility_id}"

    await self.secret_manager_client.create_secret(
        request={
            "parent": f"projects/{self.project_id}",
            "secret_id": f"sadp-api-key-{key_config.service_type}-{key_config.facility_id}",
            "secret": {
                "replication": {
                    "user_managed": {
                        "replicas": [
                            {"location": "us-central1"},
                            {"location": "us-east1"}  # Backup region
                        ]
                    }
                },
                "labels": {
                    "healthcare": "true",
                    "facility_id": key_config.facility_id,
                    "service_type": key_config.service_type
                }
            }
        }
    )
```

```python
        # Add secret version
        await self.secret_manager_client.add_secret_version(
            request={
                "parent": secret_name,
                "payload": {"data": json.dumps(secret_data).encode("utf-8")}
            }
        )

        return GoogleHealthcareAPIKey(
            api_key_id=api_key_response.name,
            key_string=api_key_response.key_string,
            restrictions=api_key_response.restrictions,
            facility_id=key_config.facility_id,
            phi_access_level=key_config.phi_access_level,
            created_at=datetime.utcnow()
        )

    async def validate_healthcare_api_key(self, api_key_string: str) -> Optional[HealthcareAPIKeyContext]:
        """Validate Google Cloud API key for healthcare operations"""

        try:
            # Look up API key
            lookup_response = await self.api_keys_client.lookup_key(
                request={"key_string": api_key_string}
            )

            api_key = lookup_response.parent

            # Get key details
            key_details = await self.api_keys_client.get_key(name=api_key)

            # Validate key is active and not expired
            if key_details.state != apikeys_v2.Key.State.ACTIVE:
                return None

            # Extract healthcare context from annotations
            annotations = key_details.annotations

            return HealthcareAPIKeyContext(
                api_key_id=key_details.name,
                facility_id=annotations.get("facility_id"),
                service_type=annotations.get("service_type"),
                phi_access_level=int(annotations.get("phi_access_level", "0")),
                healthcare_compliant=annotations.get("healthcare_compliant") == "true",
                allowed_services=self.extract_allowed_services(key_details.restrictions),
                key_created_at=key_details.create_time
            )

        except Exception as e:
            # Log validation attempt
            await self.audit_logger.log_api_key_validation_error(e, api_key_string)
            return None

# Updated Google Cloud Security Configuration
GOOGLE_CLOUD_SECURITY_CONFIG = {
    "authentication": {
        "method": "google_cloud_iam",
        "service_account_auth": True,
        "identity_token_validation": True,
        "api_key_validation": True
    },
    "authorization": {
        "iam_permissions": True,
        "healthcare_role_validation": True,
        "vpc_service_controls": True,
        "resource_level_permissions": True
    },
    "encryption": {
        "kms_provider": "google_cloud_kms",
        "key_ring": "sarthi-healthcare",
        "phi_encryption_key": "sarthi-phi-key",
        "field_level_encryption": True,
        "encryption_at_rest": True,
        "encryption_in_transit": True
    },
    "compliance": {
        "phi_detection": "google_cloud_dlp",
        "audit_logging": "google_cloud_audit_logs",
        "data_residency": "us_central1",
        "vpc_service_controls": True,
        "binary_authorization": True
    },
    "monitoring": {
        "security_monitoring": "google_cloud_security_command_center",
        "audit_analysis": "google_cloud_bigquery",
        "threat_detection": "google_cloud_armor",
        "anomaly_detection": "google_cloud_monitoring"
    }
```

~~~
}
```## Google Cloud Healthcare API Specifications

### Enhanced API Specifications for Google Cloud Healthcare

```yaml
openapi: 3.0.3
info:
  title: Sarthi AI Agent Development Platform API - Google Healthcare
  version: 1.0.0
  description: |
    Enterprise-grade API for AI agent management using Google Cloud Healthcare APIs
    and Google AI (Gemini) models. Supports healthcare-specific workflows with
    Google Cloud native HIPAA compliance.
  contact:
    name: Sarthi Platform Team
    url: https://sarthi.com/support
    email: support@sarthi.com
  license:
    name: Commercial License
    url: https://sarthi.com/license

servers:
  - url: https://sadp.sarthi.com/api/v1
    description: Production server (Google Cloud)
  - url: https://sadp-staging.sarthi.com/api/v1
    description: Staging server (Google Cloud)

security:
  - GoogleCloudAuth: []
  - GoogleAPIKey: []

paths:
  /agents/{agentName}/execute:
    post:
      summary: Execute AI agent with Google AI (Gemini)
      description: |
        Execute a healthcare AI agent using Google Cloud Healthcare APIs
        and Google AI Gemini models with HIPAA compliance.
      operationId: executeHealthcareAgent
      tags:
        - Google Healthcare Agents
      parameters:
        - name: agentName
~~~

```
        in: path
        required: true
        schema:
          type: string
          enum: [
            "gemini_clinical_agent", "gemini_billing_agent",
            "healthcare_nlp_agent", "fhir_processing_agent",
            "medical_imaging_agent", "hl7_processing_agent"
          ]
        example: "gemini_clinical_agent"
      - name: X-Google-Healthcare-Dataset
        in: header
        required: true
        schema:
          type: string
        description: Google Cloud Healthcare dataset ID
        example: "projects/sarthi-healthcare-platform/locations/us-central1/datasets/sarthi-production-dataset"
      - name: X-Google-Project-ID
        in: header
        required: true
        schema:
          type: string
        description: Google Cloud Project ID
        example: "sarthi-healthcare-platform"
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/GoogleHealthcareAgentRequest'
          examples:
            gemini_clinical_example:
              summary: Gemini clinical agent execution
              value:
                input_data:
                  patient_fhir_resource:
                    resourceType: "Patient"
                    id: "SARTHI-PT-001"
                    name: [{"family": "Doe", "given": ["Jane"]}]
                  clinical_context:
                    primary_diagnosis: "Type 2 Diabetes Mellitus"
                    icd10_code: "E11.9"
                  google_healthcare_context:
                    fhir_store: "projects/sarthi-healthcare-platform/locations/us-central1/datasets/sarthi-production-dataset/
                consent_store: "projects/sarthi-healthcare-platform/locations/us-central1/datasets/sarthi-production-data
              gemini_config:
                model: "gemini-pro-healthcare"
                temperature: 0.2
                max_tokens: 2048
                safety_settings: "healthcare_maximum"
              google_cloud_options:
                use_healthcare_nlp: true
                phi_detection_enabled: true
                audit_logging_level: "FULL"
      responses:
        '200':
          description: Healthcare agent execution successful
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/GoogleHealthcareAgentResponse'
        '400':
          $ref: '#/components/responses/BadRequest'
        '401':
          $ref: '#/components/responses/Unauthorized'
        '403':
          $ref: '#/components/responses/GoogleCloudPermissionDenied'

  /healthcare/fhir/process:
    post:
      summary: Process FHIR resources with Google Healthcare APIs
      description: |
        Process FHIR resources using Google Cloud Healthcare FHIR stores
        with integrated AI analysis via Gemini models.
      operationId: processFHIRResources
      tags:
        - Google Healthcare FHIR
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/FHIRProcessingRequest'
      responses:
        '200':
          description: FHIR processing completed
          content:
            application/json:
```

```yaml
              schema:
                $ref: '#/components/schemas/FHIRProcessingResponse'

  /healthcare/hl7v2/process:
    post:
      summary: Process HL7v2 messages with Google Healthcare APIs
      description: |
        Process HL7v2 messages using Google Cloud Healthcare HL7v2 stores
        with legacy system integration.
      operationId: processHL7v2Messages
      tags:
        - Google Healthcare HL7v2
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/HL7v2ProcessingRequest'
      responses:
        '200':
          description: HL7v2 processing completed
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/HL7v2ProcessingResponse'

  /healthcare/consent/validate:
    post:
      summary: Validate patient consent with Google Healthcare Consent API
      description: |
        Validate patient consent for data access using Google Cloud
        Healthcare Consent API with privacy controls.
      operationId: validatePatientConsent
      tags:
        - Google Healthcare Consent
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ConsentValidationRequest'
      responses:
        '200':
          description: Consent validation completed
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ConsentValidationResponse'

  /google-ai/gemini/healthcare:
    post:
      summary: Direct Gemini healthcare model access
      description: |
        Direct access to Google AI Gemini models configured for
        healthcare applications with enhanced safety settings.
      operationId: geminiHealthcareInference
      tags:
        - Google AI Gemini
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/GeminiHealthcareRequest'
      responses:
        '200':
          description: Gemini inference completed
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/GeminiHealthcareResponse'

components:
  securitySchemes:
    GoogleCloudAuth:
      type: oauth2
      flows:
        clientCredentials:
          tokenUrl: https://oauth2.googleapis.com/token
          scopes:
            https://www.googleapis.com/auth/cloud-healthcare: "Access Google Cloud Healthcare APIs"
            https://www.googleapis.com/auth/cloud-platform: "Access Google Cloud Platform"

    GoogleAPIKey:
      type: apiKey
      in: header
      name: X-Goog-Api-Key
```

```yaml
schemas:
  GoogleHealthcareAgentRequest:
    type: object
    required:
      - input_data
      - gemini_config
    properties:
      input_data:
        type: object
        description: Healthcare-specific input data
        properties:
          patient_fhir_resource:
            $ref: '#/components/schemas/FHIRResource'
          clinical_context:
            type: object
            properties:
              primary_diagnosis:
                type: string
                example: "Type 2 Diabetes Mellitus"
              icd10_code:
                type: string
                example: "E11.9"
              symptoms:
                type: array
                items:
                  type: string
          google_healthcare_context:
            $ref: '#/components/schemas/GoogleHealthcareContext'
      gemini_config:
        $ref: '#/components/schemas/GeminiConfig'
      google_cloud_options:
        $ref: '#/components/schemas/GoogleCloudOptions'

  GoogleHealthcareContext:
    type: object
    properties:
      fhir_store:
        type: string
        description: Google Cloud Healthcare FHIR store path
        example: "projects/sarthi-healthcare-platform/locations/us-central1/datasets/sarthi-production-dataset/fhirStore
      dicom_store:
        type: string
        description: Google Cloud Healthcare DICOM store path
      hl7v2_store:
```

```
      type: string
      description: Google Cloud Healthcare HL7v2 store path
    consent_store:
      type: string
      description: Google Cloud Healthcare Consent store path
    dataset_id:
      type: string
      example: "sarthi-production-dataset"
    location:
      type: string
      example: "us-central1"

GeminiConfig:
  type: object
  required:
    - model
  properties:
    model:
      type: string
      enum: ["gemini-pro-healthcare", "gemini-pro-vision-healthcare"]
      default: "gemini-pro-healthcare"
    temperature:
      type: number
      minimum: 0.0
      maximum: 2.0
      default: 0.2
      description: Lower temperature for healthcare accuracy
    max_tokens:
      type: integer
      minimum: 1
      maximum: 8192
      default: 2048
    top_p:
      type: number
      minimum: 0.0
      maximum: 1.0
      default: 0.8
    top_k:
      type: integer
      minimum: 1
      maximum: 40
      default: 40
    safety_settings:
      type: string
      enum: ["healthcare_maximum", "healthcare_standard", "standard"]
      default: "healthcare_maximum"
      description: Healthcare-specific safety configurations
    healthcare_mode:
      type: boolean
      default: true
      description: Enable healthcare-specific features
    phi_protection:
      type: boolean
      default: true
      description: Enable PHI protection mechanisms

GoogleCloudOptions:
  type: object
  properties:
    use_healthcare_nlp:
      type: boolean
      default: true
      description: Use Google Healthcare NLP API for entity extraction
    phi_detection_enabled:
      type: boolean
      default: true
      description: Enable Google Cloud DLP for PHI detection
    audit_logging_level:
      type: string
      enum: ["NONE", "BASIC", "FULL"]
      default: "FULL"
      description: Google Cloud Audit Logs level
    kms_encryption:
      type: boolean
      default: true
      description: Use Google Cloud KMS for encryption
    vpc_service_controls:
      type: boolean
      default: true
      description: Apply VPC Service Controls

GoogleHealthcareAgentResponse:
  type: object
  required:
    - execution_id
    - agent_name
    - status
    - result
```

```yaml
  properties:
    execution_id:
      type: string
      format: uuid
      description: Unique execution identifier
    agent_name:
      type: string
      description: Name of the executed healthcare agent
    status:
      type: string
      enum: ["completed", "failed", "timeout", "cancelled"]
    execution_time:
      type: integer
      description: Execution time in milliseconds
    result:
      type: object
      description: Healthcare agent execution result
      properties:
        clinical_analysis:
          type: string
          description: Clinical analysis from Gemini
        confidence_score:
          type: number
          minimum: 0
          maximum: 1
        medical_entities:
          type: array
          items:
            $ref: '#/components/schemas/MedicalEntity'
        recommendations:
          type: array
          items:
            type: string
    google_cloud_metadata:
      $ref: '#/components/schemas/GoogleCloudMetadata'
    gemini_metadata:
      $ref: '#/components/schemas/GeminiMetadata'
    compliance:
      $ref: '#/components/schemas/GoogleHealthcareCompliance'
    performance:
      $ref: '#/components/schemas/PerformanceMetrics'

GoogleCloudMetadata:
  type: object
```

```yaml
      properties:
        project_id:
          type: string
          example: "sarthi-healthcare-platform"
        region:
          type: string
          example: "us-central1"
        healthcare_dataset:
          type: string
          example: "sarthi-production-dataset"
        services_used:
          type: array
          items:
            type: string
          example: ["healthcare.googleapis.com", "aiplatform.googleapis.com"]
        kms_key_used:
          type: string
          example: "projects/sarthi-healthcare-platform/locations/us-central1/keyRings/sarthi-healthcare/cryptoKeys/sarth

    GeminiMetadata:
      type: object
      properties:
        model_version:
          type: string
          example: "gemini-pro-healthcare-001"
        model_endpoint:
          type: string
          example: "us-central1-aiplatform.googleapis.com"
        tokens_used:
          type: object
          properties:
            input_tokens:
              type: integer
            output_tokens:
              type: integer
            total_tokens:
              type: integer
        safety_ratings:
          type: array
          items:
            type: object
            properties:
              category:
                type: string
              probability:
                type: string
              blocked:
                type: boolean
        cost:
          type: number
          format: decimal
          description: Execution cost in USD

    GoogleHealthcareCompliance:
      type: object
      properties:
        hipaa_compliant:
          type: boolean
          description: HIPAA compliance status
        google_cloud_audit_logged:
          type: boolean
          description: Whether logged to Google Cloud Audit Logs
        phi_detected:
          type: boolean
          description: Whether PHI was detected by Google Cloud DLP
        phi_encrypted:
          type: boolean
          description: Whether PHI was encrypted with Google Cloud KMS
        consent_validated:
          type: boolean
          description: Whether patient consent was validated
        vpc_service_controls_applied:
          type: boolean
          description: Whether VPC Service Controls were applied
        data_residency:
          type: string
          example: "us-central1"
          description: Google Cloud data residency location

    FHIRResource:
      type: object
      description: FHIR R4 resource structure
      properties:
        resourceType:
          type: string
          example: "Patient"
        id:
          type: string
```

```yaml
      example: "SARTHI-PT-001"
  additionalProperties: true

MedicalEntity:
  type: object
  properties:
    entity_type:
      type: string
      example: "MEDICAL_CONDITION"
    text:
      type: string
      example: "Type 2 Diabetes"
    confidence:
      type: number
      minimum: 0
      maximum: 1
    source:
      type: string
      enum: ["gemini", "healthcare_nlp", "manual"]

FHIRProcessingRequest:
  type: object
  required:
    - fhir_resources
    - processing_type
  properties:
    fhir_resources:
      type: array
      items:
        $ref: '#/components/schemas/FHIRResource'
    processing_type:
      type: string
      enum: ["store", "search", "analyze", "validate"]
    fhir_store_path:
      type: string
      description: Google Cloud Healthcare FHIR store path
    analysis_config:
      type: object
      properties:
        use_gemini_analysis:
          type: boolean
          default: true
        extract_medical_entities:
          type: boolean
          default: true

FHIRProcessingResponse:
  type: object
  properties:
    processing_id:
      type: string
      format: uuid
    fhir_resources_processed:
      type: integer
    stored_resource_ids:
      type: array
      items:
        type: string
    analysis_results:
      type: object
      description: Results from Gemini analysis if requested
    google_healthcare_metadata:
      $ref: '#/components/schemas/GoogleCloudMetadata'

HL7v2ProcessingRequest:
  type: object
  required:
    - hl7_message
    - processing_type
  properties:
    hl7_message:
      type: string
      description: Base64 encoded HL7v2 message
    processing_type:
      type: string
      enum: ["ingest", "parse", "convert_to_fhir"]
    hl7v2_store_path:
      type: string
      description: Google Cloud Healthcare HL7v2 store path
    conversion_config:
      type: object
      properties:
        target_fhir_version:
          type: string
          enum: ["R4", "STU3"]
          default: "R4"

ConsentValidationRequest:
  type: object
  required:
    - patient_id
    - data_access_request
  properties:
    patient_id:
      type: string
      example: "SARTHI-PT-001"
    data_access_request:
      type: object
      properties:
        data_types:
          type: array
          items:
            type: string
        purpose:
          type: string
          example: "treatment"
        requesting_organization:
          type: string
    consent_store_path:
      type: string
      description: Google Cloud Healthcare Consent store path

GeminiHealthcareRequest:
  type: object
  required:
    - prompt
    - gemini_config
  properties:
    prompt:
      type: string
      description: Healthcare-specific prompt for Gemini
    gemini_config:
      $ref: '#/components/schemas/GeminiConfig'
    context:
      type: object
      properties:
        medical_specialty:
          type: string
          example: "cardiology"
        patient_context:
          type: object
          description: Anonymized patient context
```

```
responses:
  GoogleCloudPermissionDenied:
    description: Google Cloud IAM permission denied
    content:
      application/json:
        schema:
          type: object
          properties:
            error:
              type: string
              example: "PERMISSION_DENIED"
            message:
              type: string
              example: "Service account lacks required Google Cloud Healthcare permissions"
            required_permissions:
              type: array
              items:
                type: string
              example: ["healthcare.datasets.get", "healthcare.fhirStores.search"]
            google_cloud_project:
              type: string
              example: "sarthi-healthcare-platform"
```

Google Cloud Integration Examples

Complete Integration Example

```
python
```

```python
# Complete Google Cloud Healthcare Integration Example
async def main():
    """Complete example of Google Cloud Healthcare Platform integration"""

    print("🏥 GOOGLE CLOUD HEALTHCARE PLATFORM INTEGRATION")
    print("=" * 60)

    # Initialize Google Cloud Healthcare components
    google_healthcare_client = GoogleHealthcareAPIClient(
        project_id="sarthi-healthcare-platform",
        location="us-central1",
        dataset_id="sarthi-production-dataset"
    )

    gemini_client = GoogleAIGeminiClient()
    security_client = GoogleCloudHealthcareSecurity()
    compliance_client = GoogleCloudHealthcareCompliance()

    print("\n1️⃣ GOOGLE CLOUD AUTHENTICATION")
    print("-" * 40)

    # Authenticate with Google Cloud IAM
    mock_request = create_mock_request_with_google_auth()
    service_context = await security_client.authenticate_with_google_iam(mock_request)
    print(f"✅ Authenticated: {service_context.service_account_email}")
    print(f"   Project: {service_context.project_id}")
    print(f"   Healthcare Permissions: {len(service_context.healthcare_permissions)}")

    print("\n2️⃣ FHIR RESOURCE PROCESSING")
    print("-" * 40)

    # Create sample FHIR patient resource
    patient_resource = {
        "resourceType": "Patient",
        "id": "SARTHI-PT-001",
        "name": [{"family": "Smith", "given": ["John"]}],
        "birthDate": "1970-05-15",
        "gender": "male",
        "identifier": [
            {
                "system": "http://sarthi.com/patient-id",
                "value": "SARTHI-PT-001"
            }
    ]
}

# Store in Google Cloud Healthcare FHIR store
stored_resource_id = await google_healthcare_client.store_patient_data(patient_resource)
print(f"✅ FHIR Patient stored: {stored_resource_id}")

print("\n3️⃣ GEMINI HEALTHCARE AI PROCESSING")
print("-" * 40)

# Configure healthcare agent for Gemini
healthcare_agent_config = HealthcareAgentConfig(
    agent_type="gemini_clinical_agent",
    medical_specialty="primary_care",
    temperature=0.2,
    max_tokens=2048,
    allow_phi_output=False,
    safety_level="healthcare_maximum"
)

# Prepare clinical input
clinical_input = {
    "patient_fhir_resource": patient_resource,
    "chief_complaint": "Routine annual physical examination",
    "vital_signs": {
        "blood_pressure": "120/80",
        "heart_rate": 72,
        "temperature": "98.6°F",
        "weight": "180 lbs"
    },
    "medical_history": ["Hypertension", "Type 2 Diabetes"]
}

# Execute Gemini healthcare analysis
gemini_response = await gemini_client.execute_healthcare_agent(
    healthcare_agent_config, clinical_input
)

print(f"✅ Gemini Analysis Complete:")
print(f"   Model: {gemini_response.model_version}")
print(f"   Execution Time: {gemini_response.execution_time}ms")
print(f"   Healthcare Validated: {gemini_response.healthcare_validated}")
print(f"   Medical Accuracy Score: {gemini_response.medical_accuracy_score}")
print(f"   PHI Detected: {gemini_response.phi_detected}")
print("\n4️⃣ GOOGLE CLOUD DLP PHI DETECTION")
print("-" * 40)

# Test PHI detection with Google Cloud DLP
test_data = {
    "patient_name": "John Smith",
    "ssn": "123-45-6789",
    "medical_record_number": "MR-123456",
    "diagnosis": "Type 2 Diabetes Mellitus"
}

phi_fields = await security_client.detect_phi_with_google_dlp(test_data)
print(f"✅ PHI Detection Complete:")
print(f"   PHI Fields Found: {len(phi_fields)}")
for field, details in phi_fields.items():
    print(f"   - {field}: {details['info_type']} ({details['likelihood']})")

print("\n5️⃣ GOOGLE CLOUD KMS ENCRYPTION")
print("-" * 40)

# Encrypt PHI using Google Cloud KMS
encrypted_data = await security_client.encrypt_phi_with_google_kms(
    test_data, patient_id="SARTHI-PT-001"
)

print(f"✅ PHI Encryption Complete:")
print(f"   Fields Encrypted: {encrypted_data.phi_fields_count}")
print(f"   Encryption Key: Google Cloud KMS")
print(f"   Key Ring: sarthi-healthcare")

print("\n6️⃣ HEALTHCARE NLP PROCESSING")
print("-" * 40)

# Process clinical text with Healthcare NLP API
clinical_text = """
Patient presents with chest pain and shortness of breath.
History of hypertension and diabetes mellitus type 2.
Current medications include metformin 1000mg BID and lisinopril 10mg daily.
Vital signs: BP 140/90, HR 88, RR 18, Temp 98.4°F.
"""

nlp_client = HealthcareNLPClient()
nlp_results = await nlp_client.analyze_entities(clinical_text)
```

```
    print(f"✅ Healthcare NLP Complete:")
    print(f"   Medical Entities Found: {len(nlp_results['entities'])}")
    print(f"   API Version: {nlp_results['processing_metadata']['api_version']}")

    print("\n7️⃣ HL7V2 MESSAGE PROCESSING")
    print("-" * 40)

    # Sample HL7v2 message
    hl7_message = """MSH|^~\\&|EPIC|SARTHI|||20250819120000||ADT^A01|123456|P|2.5
PID|1||SARTHI-PT-001^^^MR^MR||SMITH^JOHN^||19700515|M|||123 MAIN ST^^ANYTOWN^ST^12345^USA||(555)1
PV1|1|I|2000^2012^01||||1234567890^ATTENDING^PHYSICIAN||SUR||||||1234567890^ATTENDING^PHYSICIAN|INP|A||

    processed_message = await google_healthcare_client.process_hl7v2_message(hl7_message)
    print(f"✅ HL7v2 Processing Complete:")
    print(f"   Message ID: {processed_message.message_id}")
    print(f"   Processing Time: {processed_message.processing_timestamp}")

    print("\n8️⃣ CONSENT VALIDATION")
    print("-" * 40)

    # Validate patient consent
    consent_record = await google_healthcare_client.retrieve_patient_consent("SARTHI-PT-001")
    print(f"✅ Consent Validation:")
    print(f"   Consent Status: {consent_record.status if consent_record else 'Not Found'}")

    print("\n9️⃣ COMPLIANCE AUDIT LOGGING")
    print("-" * 40)

    # Generate healthcare audit log
    audit_log = HealthcareAccessLog(
        service_account_email=service_context.service_account_email,
        resource_name="projects/sarthi-healthcare-platform/locations/us-central1/datasets/sarthi-production-dataset",
        operation_type="FHIR_PATIENT_CREATE",
        phi_accessed=True,
        patient_count=1,
        data_classification="PHI",
        phi_level=3,
        facility_id="SARTHI-CLINIC-001"
    )

    await compliance_client.log_healthcare_api_access(audit_log)
    print(f"✅ Audit Logging Complete:")
    print(f"   Logged to Google Cloud Audit Logs")
```

```python
print(f"   Stored in BigQuery for analytics")

print("\n🔟 COMPREHENSIVE HIPAA AUDIT REPORT")
print("-" * 40)

# Generate HIPAA audit report
report_config = HIPAAAuditReportConfig(
    start_date=datetime.utcnow() - timedelta(days=30),
    end_date=datetime.utcnow(),
    include_phi_access=True,
    include_compliance_violations=True
)

audit_report = await compliance_client.generate_hipaa_audit_report(report_config)
print(f"✅ HIPAA Audit Report Generated:")
print(f"   Report ID: {audit_report.report_id}")
print(f"   Compliance Status: {audit_report.compliance_status}")
print(f"   Total API Calls: {audit_report.data['total_api_calls']}")
print(f"   PHI Access Events: {audit_report.data['phi_access_events']}")

print("\n🎯 GOOGLE CLOUD HEALTHCARE INTEGRATION SUMMARY")
print("=" * 60)
print("✅ Google Cloud IAM authentication successful")
print("✅ Google Cloud Healthcare FHIR store integration")
print("✅ Google AI Gemini healthcare models")
print("✅ Google Cloud DLP PHI detection")
print("✅ Google Cloud KMS encryption")
print("✅ Healthcare NLP API integration")
print("✅ HL7v2 message processing")
print("✅ Patient consent validation")
print("✅ Google Cloud Audit Logs compliance")
print("✅ Comprehensive HIPAA audit reporting")

print(f"\n📊 GOOGLE CLOUD NATIVE BENEFITS:")
print(f"   • Native HIPAA compliance with Google Cloud Healthcare APIs")
print(f"   • Advanced PHI protection with Google Cloud DLP and KMS")
print(f"   • Healthcare-optimized Gemini models")
print(f"   • Comprehensive audit trails with Cloud Audit Logs")
print(f"   • VPC Service Controls for data protection")
print(f"   • Google Cloud Security Command Center integration")

return {
    "google_cloud_integration": "successful",
    "gemini_healthcare_models": "operational",
            "healthcare_apis": "integrated",
            "compliance_framework": "hipaa_compliant",
            "security_controls": "enterprise_grade"
        }

if __name__ == "__main__":
    # Run comprehensive Google Cloud Healthcare integration
    asyncio.run(main())
```

Benefits of Google Cloud Healthcare Focus

1. Native Healthcare Compliance

- ✅ **Built-in HIPAA compliance** with Google Cloud Healthcare APIs
- ✅ **PHI protection** with Google Cloud DLP and KMS
- ✅ **Audit trails** with Google Cloud Audit Logs
- ✅ **Data residency** controls with Google Cloud regions

2. Advanced AI Capabilities

- ✅ **Healthcare-optimized Gemini** models with medical safety
- ✅ **Medical entity extraction** with Healthcare NLP API
- ✅ **Medical imaging analysis** with Gemini Pro Vision
- ✅ **Custom healthcare models** with Vertex AI

3. Comprehensive Healthcare APIs

- ✅ **FHIR R4/STU3 support** with Cloud Healthcare FHIR stores
- ✅ **HL7v2 integration** for legacy systems
- ✅ **DICOM support** for medical imaging
- ✅ **Consent management** with Healthcare Consent API

4. Enterprise Security

- ✅ **Google Cloud IAM** for fine-grained access control
- ✅ **VPC Service Controls** for data protection
- ✅ **Binary Authorization** for container security
- ✅ **Cloud Armor** for DDoS protection

5. Operational Excellence

- ✅ **Managed services** reduce operational overhead
- ✅ **Auto-scaling** with GKE Autopilot
- ✅ **Monitoring** with Cloud Monitoring and Error Reporting
- ✅ **Cost optimization** with Google Cloud pricing models

This Google Cloud Healthcare-focused architecture provides a robust, compliant, and scalable foundation for the Sarthi AI Agent Development Platform, leveraging Google's healthcare-specific infrastructure and AI capabilities.

6. Advanced Error Handling & Resilience

```python
```

```python
# Circuit Breaker and Retry Patterns
class SADPResilienceFramework:
    """Advanced resilience patterns for SADP"""

    def __init__(self):
        self.circuit_breakers = {}
        self.retry_policies = {
            "claude_api": ExponentialBackoff(
                initial_delay=1.0,
                max_delay=60.0,
                multiplier=2.0,
                max_retries=3
            ),
            "database": LinearBackoff(
                delay=0.5,
                max_retries=5
            )
        }

    async def execute_with_resilience(self,
                                operation_name: str,
                                operation: Callable,
                                *args, **kwargs) -> Any:
        """Execute operation with full resilience patterns"""

        # Circuit breaker check
        circuit_breaker = self.get_circuit_breaker(operation_name)
        if circuit_breaker.is_open():
            raise CircuitBreakerOpenError(f"Circuit breaker open for {operation_name}")

        # Retry with backoff
        retry_policy = self.retry_policies.get(operation_name)

        for attempt in range(retry_policy.max_retries + 1):
            try:
                result = await operation(*args, **kwargs)
                circuit_breaker.record_success()
                return result

            except RetryableError as e:
                circuit_breaker.record_failure()

                if attempt < retry_policy.max_retries:
                    delay = retry_policy.calculate_delay(attempt)
                    await asyncio.sleep(delay)
                    continue
                else:
                    raise e

            except NonRetryableError as e:
                circuit_breaker.record_failure()
                raise e

    def get_circuit_breaker(self, operation_name: str) -> CircuitBreaker:
        """Get or create circuit breaker for operation"""
        if operation_name not in self.circuit_breakers:
            self.circuit_breakers[operation_name] = CircuitBreaker(
                failure_threshold=5,
                timeout=30.0,
                expected_exception=Exception
            )
        return self.circuit_breakers[operation_name]

# Advanced Error Classifications
class SADPErrorHandler:
    """Intelligent error handling and classification"""

    ERROR_CLASSIFICATIONS = {
        "claude_api_errors": {
            "rate_limit": {"retryable": True, "severity": "medium"},
            "invalid_request": {"retryable": False, "severity": "low"},
            "server_error": {"retryable": True, "severity": "high"},
            "timeout": {"retryable": True, "severity": "medium"}
        },
        "validation_errors": {
            "schema_violation": {"retryable": False, "severity": "low"},
            "phi_leak_detected": {"retryable": False, "severity": "critical"},
            "compliance_violation": {"retryable": False, "severity": "critical"}
        },
        "infrastructure_errors": {
            "database_connection": {"retryable": True, "severity": "high"},
            "cache_miss": {"retryable": True, "severity": "low"},
            "service_unavailable": {"retryable": True, "severity": "high"}
        }
    }

    def classify_error(self, error: Exception) -> ErrorClassification:
```

```
    """Intelligent error classification"""
    error_type = type(error).__name__
    error_message = str(error)

    # Use ML-based classification for unknown errors
    if error_type not in self.ERROR_CLASSIFICATIONS:
        return self.ml_classify_error(error_message)

    classification = self.ERROR_CLASSIFICATIONS[error_type]

    return ErrorClassification(
        error_type=error_type,
        retryable=classification["retryable"],
        severity=classification["severity"],
        recommended_action=self.get_recommended_action(error),
        user_message=self.get_user_friendly_message(error)
    )
```

7. Performance Optimization & Caching

python

```python
# Advanced Caching Strategy
class SADPCacheManager:
    """Multi-level caching with intelligent invalidation"""

    def __init__(self):
        self.l1_cache = TTLCache(maxsize=1000, ttl=300)  # 5 min in-memory
        self.l2_cache = redis.Redis(host="redis-cluster")  # 1 hour Redis
        self.l3_cache = self.setup_cdn_cache()  # 24 hour CDN

    async def get_cached_response(self, cache_key: str) -> Optional[dict]:
        """Multi-level cache retrieval"""

        # L1: In-memory cache (fastest)
        if cache_key in self.l1_cache:
            return self.l1_cache[cache_key]

        # L2: Redis cache (fast)
        cached_data = await self.l2_cache.get(cache_key)
        if cached_data:
            data = json.loads(cached_data)
            self.l1_cache[cache_key] = data  # Promote to L1
            return data

        # L3: CDN cache (for static responses)
        if self.is_static_response(cache_key):
            return await self.get_from_cdn(cache_key)

        return None

    async def cache_response(self, cache_key: str, data: dict, ttl: int = 3600):
        """Intelligent response caching"""

        # Determine cache levels based on data characteristics
        cache_levels = self.determine_cache_levels(data)

        if "l1" in cache_levels:
            self.l1_cache[cache_key] = data

        if "l2" in cache_levels:
            await self.l2_cache.setex(
                cache_key,
                ttl,
                json.dumps(data, cls=SADPJSONEncoder)
        )

        if "l3" in cache_levels and self.is_publicly_cacheable(data):
            await self.cache_to_cdn(cache_key, data, ttl)

    def determine_cache_levels(self, data: dict) -> List[str]:
        """Determine appropriate cache levels for data"""
        cache_levels = []

        # Always cache non-PHI data in L1
        if not self.contains_phi(data):
            cache_levels.extend(["l1", "l2"])

        # Cache static responses in CDN
        if self.is_static_content(data):
            cache_levels.append("l3")

        return cache_levels

# Connection Pooling and Resource Management
class SADPResourceManager:
    """Advanced resource management for external services"""

    def __init__(self):
        self.claude_client_pool = self.create_claude_pool()
        self.db_connection_pool = self.create_db_pool()
        self.redis_connection_pool = self.create_redis_pool()

    def create_claude_pool(self) -> ClientPool:
        """Create optimized Claude API client pool"""
        return ClientPool(
            client_factory=lambda: anthropic.Anthropic(),
            max_size=50,
            min_size=10,
            acquire_timeout=30.0,
            max_lifetime=3600.0,
            health_check_interval=60.0
        )

    async def get_claude_client(self) -> anthropic.Anthropic:
        """Get Claude client with automatic retry and failover"""
        for attempt in range(3):
            try:
                client = await self.claude_client_pool.acquire()
        # Health check
        if await self.health_check_claude_client(client):
            return client
        else:
            await self.claude_client_pool.discard(client)

    except Exception as e:
        if attempt == 2:  # Last attempt
            raise ClaudeClientUnavailableError(
                "Unable to acquire healthy Claude client"
            )
        await asyncio.sleep(2 ** attempt)  # Exponential backoff
```

8. Advanced Monitoring & Observability

python

```python
# Comprehensive Monitoring Stack
class SADPMonitoringStack:
    """Production-grade monitoring and observability"""

    def __init__(self):
        self.prometheus_registry = CollectorRegistry()
        self.metrics_collector = self.setup_metrics()
        self.tracer = self.setup_distributed_tracing()
        self.logger = self.setup_structured_logging()

    def setup_metrics(self) -> MetricsCollector:
        """Setup comprehensive metrics collection"""

        # Business metrics
        agent_execution_counter = Counter(
            'sadp_agent_executions_total',
            'Total number of agent executions',
            ['agent_name', 'status', 'facility_id']
        )

        agent_execution_duration = Histogram(
            'sadp_agent_execution_duration_seconds',
            'Agent execution duration in seconds',
            ['agent_name'],
            buckets=(0.1, 0.5, 1.0, 2.5, 5.0, 10.0, 30.0, 60.0)
        )

        agent_accuracy_gauge = Gauge(
            'sadp_agent_accuracy_score',
            'Current agent accuracy score',
            ['agent_name']
        )

        # Infrastructure metrics
        claude_api_requests = Counter(
            'sadp_claude_api_requests_total',
            'Total Claude API requests',
            ['method', 'status_code']
        )

        claude_api_latency = Histogram(
            'sadp_claude_api_latency_seconds',
            'Claude API request latency',
            buckets=(0.1, 0.25, 0.5, 1.0, 2.5, 5.0, 10.0)
        )

        # Cost metrics
        claude_token_usage = Counter(
            'sadp_claude_tokens_total',
            'Total Claude tokens consumed',
            ['agent_name', 'token_type']
        )

        daily_cost_gauge = Gauge(
            'sadp_daily_cost_usd',
            'Daily operational cost in USD',
            ['service']
        )

        return MetricsCollector(
            agent_execution_counter,
            agent_execution_duration,
            agent_accuracy_gauge,
            claude_api_requests,
            claude_api_latency,
            claude_token_usage,
            daily_cost_gauge
        )

    def setup_distributed_tracing(self) -> Tracer:
        """Setup distributed tracing with Jaeger"""
        config = Config(
            config={
                'sampler': {'type': 'const', 'param': 1},
                'logging': True,
                'reporter_batch_size': 1,
            },
            service_name='sadp-api',
            validate=True,
        )
        return config.initialize_tracer()

    def setup_structured_logging(self) -> Logger:
        """Setup structured logging with correlation IDs"""
        logger = structlog.get_logger()
        structlog.configure(
            processors=[
            structlog.stdlib.filter_by_level,
            structlog.stdlib.add_logger_name,
            structlog.stdlib.add_log_level,
            structlog.processors.TimeStamper(fmt="iso"),
            structlog.processors.StackInfoRenderer(),
            structlog.processors.format_exc_info,
            structlog.processors.UnicodeDecoder(),
            structlog.processors.JSONRenderer()
        ],
        context_class=dict,
        logger_factory=structlog.stdlib.LoggerFactory(),
        wrapper_class=structlog.stdlib.BoundLogger,
        cache_logger_on_first_use=True,
    )
    return logger

# Advanced Alerting System
class SADPAlertManager:
    """Intelligent alerting with context-aware notifications"""

    def __init__(self):
        self.alert_rules = self.load_alert_rules()
        self.notification_channels = self.setup_notification_channels()
        self.escalation_policies = self.load_escalation_policies()

    async def evaluate_alerts(self, metrics: dict):
        """Evaluate metrics against alert rules"""

        for rule in self.alert_rules:
            if await self.evaluate_rule(rule, metrics):
                alert = Alert(
                    rule_name=rule.name,
                    severity=rule.severity,
                    message=rule.format_message(metrics),
                    context=self.build_alert_context(metrics),
                    timestamp=datetime.utcnow()
                )

                await self.send_alert(alert)

    async def send_alert(self, alert: Alert):
        """Send alert through appropriate channels"""

        # Determine notification channels based on severity
        channels = self.get_channels_for_severity(alert.severity)

        # Check for alert fatigue and deduplication
        if not await self.should_send_alert(alert):
            return

        # Send notifications
        tasks = []
        for channel in channels:
            if channel.type == "slack":
                tasks.append(self.send_slack_alert(alert, channel))
            elif channel.type == "pagerduty":
                tasks.append(self.send_pagerduty_alert(alert, channel))
            elif channel.type == "email":
                tasks.append(self.send_email_alert(alert, channel))

        await asyncio.gather(*tasks, return_exceptions=True)

    def load_alert_rules(self) -> List[AlertRule]:
        """Load alert rules from configuration"""
        return [
            AlertRule(
                name="agent_accuracy_degradation",
                condition="agent_accuracy < 90",
                severity="warning",
                cooldown=300  # 5 minutes
            ),
            AlertRule(
                name="claude_api_high_latency",
                condition="claude_api_p95_latency > 10",
                severity="critical",
                cooldown=60  # 1 minute
            ),
            AlertRule(
                name="high_error_rate",
                condition="error_rate > 0.05",
                severity="critical",
                cooldown=120  # 2 minutes
            ),
            AlertRule(
                name="cost_spike",
                condition="hourly_cost > daily_budget / 24 * 2",
                severity="warning",
                cooldown=1800  # 30 minutes
    )
  ]
```

9. Advanced Configuration Management

```
python
```

```python
# Environment-specific Configuration
class SADPConfig:
    """Comprehensive configuration management"""

    def __init__(self, environment: str = "production"):
        self.environment = environment
        self.config = self.load_config()
        self.secrets = self.load_secrets()

    def load_config(self) -> dict:
        """Load environment-specific configuration"""
        base_config = {
            "api": {
                "host": "0.0.0.0",
                "port": 8000,
                "workers": 4,
                "timeout": 300,
                "max_request_size": "50MB"
            },
            "database": {
                "pool_size": 20,
                "max_overflow": 30,
                "pool_timeout": 30,
                "pool_recycle": 3600
            },
            "redis": {
                "max_connections": 50,
                "socket_timeout": 5,
                "socket_connect_timeout": 5,
                "retry_on_timeout": True
            },
            "claude_api": {
                "timeout": 60,
                "max_retries": 3,
                "rate_limit": 100,  # requests per minute
                "max_tokens": 4000
            },
            "security": {
                "jwt_expiration": 3600,
                "mfa_required_for": [
                    "/api/v1/development/*",
                    "/api/v1/evaluation/*/deploy"
                ],
                "encryption_algorithm": "AES-256-GCM",
                "password_policy": {
                    "min_length": 12,
                    "require_uppercase": True,
                    "require_lowercase": True,
                    "require_numbers": True,
                    "require_symbols": True
                }
            },
            "compliance": {
                "hipaa_enabled": True,
                "audit_retention_days": 2555,  # 7 years
                "phi_encryption_required": True,
                "data_residency": "US",
                "backup_encryption": True
            },
            "monitoring": {
                "metrics_retention_days": 90,
                "log_retention_days": 365,
                "alert_cooldown_seconds": 300,
                "health_check_interval": 30
            }
        }

        # Environment-specific overrides
        env_overrides = {
            "development": {
                "api": {"workers": 1, "debug": True},
                "database": {"pool_size": 5},
                "security": {"jwt_expiration": 86400},  # 24 hours
                "compliance": {"hipaa_enabled": False}
            },
            "staging": {
                "api": {"workers": 2},
                "database": {"pool_size": 10},
                "claude_api": {"rate_limit": 50}
            },
            "production": {
                "api": {"workers": 4, "debug": False},
                "database": {"pool_size": 20},
                "claude_api": {"rate_limit": 200}
            }
        }
```

```
        config = base_config.copy()
        if self.environment in env_overrides:
            config.update(env_overrides[self.environment])

        return config

    def load_secrets(self) -> dict:
        """Load secrets from secure storage"""
        if self.environment == "development":
            return self.load_from_env()
        else:
            return self.load_from_vault()

    def load_from_vault(self) -> dict:
        """Load secrets from HashiCorp Vault or Google Secret Manager"""
        # Implementation for production secret management
        pass
```

10. Deployment and DevOps Enhancements

yaml

```
# Advanced Helm Chart Structure
sadp-platform/
├── Chart.yaml
├── values.yaml
├── values-dev.yaml
├── values-staging.yaml
├── values-prod.yaml
└── templates/
    ├── deployment.yaml
    ├── service.yaml
    ├── ingress.yaml
    ├── configmap.yaml
    ├── secrets.yaml
    ├── hpa.yaml
    ├── pdb.yaml
    ├── networkpolicy.yaml
    ├── rbac.yaml
    ├── servicemonitor.yaml
    └── tests/
        ├── test-connection.yaml
        └── test-health.yaml

# values-prod.yaml (Production Configuration)
global:
  environment: production
  imageTag: "v1.0.0"
  imagePullPolicy: IfNotPresent

replicaCount: 3

image:
  repository: gcr.io/sarthi-platform/sadp-api
  tag: v1.0.0

service:
  type: ClusterIP
  port: 80
  targetPort: 8000

ingress:
  enabled: true
  className: nginx
  annotations:
```

```yaml
    cert-manager.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/rate-limit: "100"
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
  hosts:
    - host: sadp.sarthi.com
      paths:
        - path: /
          pathType: Prefix
  tls:
    - secretName: sadp-tls
      hosts:
        - sadp.sarthi.com

autoscaling:
  enabled: true
  minReplicas: 3
  maxReplicas: 20
  targetCPUUtilizationPercentage: 70
  targetMemoryUtilizationPercentage: 80

podDisruptionBudget:
  enabled: true
  minAvailable: 2

resources:
  limits:
    cpu: 2000m
    memory: 4Gi
  requests:
    cpu: 500m
    memory: 1Gi

nodeSelector:
  node-type: compute-optimized

tolerations:
  - key: "sadp-workload"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"

affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
```

```
    - weight: 100
      podAffinityTerm:
        labelSelector:
          matchExpressions:
          - key: app.kubernetes.io/name
            operator: In
            values:
            - sadp-api
        topologyKey: kubernetes.io/hostname

# CI/CD Pipeline with GitLab CI
.gitlab-ci.yml: |
  stages:
    - test
    - build
    - security-scan
    - deploy-staging
    - integration-tests
    - deploy-production
    - post-deploy-tests

  variables:
    DOCKER_REGISTRY: gcr.io/sarthi-platform
    DOCKER_DRIVER: overlay2
    DOCKER_TLS_CERTDIR: "/certs"

  before_script:
    - echo $GCP_SERVICE_ACCOUNT_KEY | base64 -d > gcp-key.json
    - gcloud auth activate-service-account --key-file gcp-key.json
    - gcloud config set project sarthi-platform

  test:
    stage: test
    image: python:3.11
    services:
      - postgres:15
      - redis:7
    script:
      - pip install poetry
      - poetry install
      - poetry run pytest tests/ --cov=src/ --cov-report=xml
      - poetry run mypy src/
      - poetry run black --check src/
      - poetry run isort --check-only src/
```

```yaml
  coverage: '/TOTAL.+ ([0-9]{1,3}%)/'
  artifacts:
    reports:
      coverage_report:
        coverage_format: cobertura
        path: coverage.xml

build:
  stage: build
  image: docker:20.10.16
  services:
    - docker:20.10.16-dind
  script:
    - docker build -t $DOCKER_REGISTRY/sadp-api:$CI_COMMIT_SHA .
    - docker push $DOCKER_REGISTRY/sadp-api:$CI_COMMIT_SHA
  only:
    - main
    - develop

security-scan:
  stage: security-scan
  image: aquasec/trivy:latest
  script:
    - trivy image --exit-code 1 --severity HIGH,CRITICAL $DOCKER_REGISTRY/sadp-api:$CI_COMMIT_SHA
  allow_failure: false

deploy-staging:
  stage: deploy-staging
  image: google/cloud-sdk:alpine
  script:
    - gcloud container clusters get-credentials staging-cluster --region us-central1
    - helm upgrade --install sadp-staging ./helm/sadp-platform
      --namespace sadp-staging
      --values ./helm/sadp-platform/values-staging.yaml
      --set image.tag=$CI_COMMIT_SHA
  environment:
    name: staging
    url: https://sadp-staging.sarthi.com
  only:
    - develop

integration-tests:
  stage: integration-tests
  image: python:3.11
```

```
    script:
      - pip install pytest requests
      - pytest tests/integration/ --base-url=https://sadp-staging.sarthi.com
    dependencies:
      - deploy-staging

  deploy-production:
    stage: deploy-production
    image: google/cloud-sdk:alpine
    script:
      - gcloud container clusters get-credentials production-cluster --region us-central1
      - helm upgrade --install sadp-production ./helm/sadp-platform
        --namespace sadp-production
        --values ./helm/sadp-platform/values-prod.yaml
        --set image.tag=$CI_COMMIT_SHA
    environment:
      name: production
      url: https://sadp.sarthi.com
    when: manual
    only:
      - main

  post-deploy-tests:
    stage: post-deploy-tests
    image: python:3.11
    script:
      - pip install pytest requests
      - pytest tests/smoke/ --base-url=https://sadp.sarthi.com
    dependencies:
      - deploy-production
```

Key Improvements Summary

1. Enterprise Architecture Patterns

- ✅ **CQRS + Event Sourcing** for better scalability
- ✅ **Service Mesh (Istio)** for advanced traffic management
- ✅ **Circuit Breakers** for resilience
- ✅ **Multi-level caching** for performance

2. Production-Grade Security

- ✅ **Multi-factor authentication** for sensitive operations

- ✅ **Fine-grained authorization** with Open Policy Agent
- ✅ **Advanced encryption** with key rotation
- ✅ **Comprehensive audit logging** for HIPAA compliance

3. Advanced Observability

- ✅ **Distributed tracing** with Jaeger
- ✅ **Comprehensive metrics** with Prometheus
- ✅ **Intelligent alerting** with context awareness
- ✅ **Structured logging** with correlation IDs

4. Robust Error Handling

- ✅ **Intelligent error classification** with ML
- ✅ **Advanced retry patterns** with backoff strategies
- ✅ **Graceful degradation** mechanisms
- ✅ **Automatic recovery** procedures

5. Scalability & Performance

- ✅ **Connection pooling** for external services
- ✅ **Resource optimization** algorithms
- ✅ **Horizontal auto-scaling** based on metrics
- ✅ **Cost optimization** tracking and alerts

This enhanced architecture provides enterprise-grade capabilities that will scale with your Sarthi platform while maintaining the highest standards of security, compliance, and performance. It's designed to be implementation-ready for Claude CLI with clear specifications and comprehensive documentation.

Service Architecture

1. API Gateway Layer

FastAPI-based REST API with the following endpoints:

```python
```

```
# Agent Runtime APIs
POST   /api/v1/agents/{agent_name}/execute
GET    /api/v1/agents/{agent_name}/capabilities
POST   /api/v1/workflows/execute
GET    /api/v1/workflows/{workflow_id}/status

# Evaluation APIs
POST   /api/v1/evaluation/agents/{agent_name}/test
GET    /api/v1/evaluation/agents/{agent_name}/results
POST   /api/v1/evaluation/test-suites/generate
GET    /api/v1/evaluation/benchmarks

# Development APIs
POST   /api/v1/development/prompts/deploy
POST   /api/v1/development/prompts/ab-test
GET    /api/v1/development/prompts/{prompt_id}/performance
POST   /api/v1/development/ci-cd/trigger

# Monitoring APIs
GET    /api/v1/monitoring/agents/{agent_name}/metrics
POST   /api/v1/monitoring/alerts/configure
GET    /api/v1/monitoring/dashboard/{dashboard_id}
GET    /api/v1/monitoring/performance/reports
```

2. Core Services

Agent Runtime Service

```python
```

```
class AgentRuntimeService:
    """Core service for executing AI agents"""

    async def execute_agent(self, agent_name: str, input_data: dict,
                          context: dict = None) -> dict:
        """Execute specified agent with input data"""

    async def execute_workflow(self, workflow_definition: dict,
                             input_data: dict) -> dict:
        """Execute multi-agent workflow"""

    def get_agent_capabilities(self, agent_name: str) -> dict:
        """Get agent capabilities and metadata"""

    def register_agent(self, agent_config: dict) -> str:
        """Register new agent with the platform"""
```

Evaluation Service

```python
class EvaluationService:
    """Service for agent evaluation and testing"""

    async def evaluate_agent(self, agent_name: str, test_cases: list) -> dict:
        """Run comprehensive agent evaluation"""

    async def generate_test_suite(self, agent_name: str,
                                test_categories: list) -> dict:
        """Generate automated test suites"""

    def get_evaluation_results(self, evaluation_id: str) -> dict:
        """Get detailed evaluation results"""

    def compare_agent_versions(self, agent_name: str,
                             version_a: str, version_b: str) -> dict:
        """Compare performance between agent versions"""
```

Development Service

```python
```

```
class DevelopmentService:
    """Service for agent development and deployment"""

    def deploy_prompt(self, prompt_config: dict) -> dict:
        """Deploy new prompt version"""

    async def setup_ab_test(self, test_config: dict) -> str:
        """Setup A/B testing for prompt optimization"""

    def trigger_ci_cd(self, agent_name: str, changes: dict) -> dict:
        """Trigger CI/CD pipeline for agent updates"""

    def get_deployment_status(self, deployment_id: str) -> dict:
        """Get deployment status and metrics"""
```

Monitoring Service

```
python

class MonitoringService:
    """Service for real-time monitoring and analytics"""

    def track_agent_execution(self, execution_data: dict) -> None:
        """Track agent execution metrics"""

    def get_performance_metrics(self, agent_name: str,
                              time_range: str) -> dict:
        """Get performance metrics for specified time range"""

    def configure_alerts(self, alert_config: dict) -> str:
        """Configure performance alerts"""

    def generate_analytics_report(self, report_config: dict) -> dict:
        """Generate comprehensive analytics reports"""
```

Implementation Stack

Backend Technology Stack

```
yaml
```

```
# Core Framework
api_framework: "FastAPI 0.104+"
async_runtime: "Python 3.11+ with asyncio"
web_server: "Uvicorn with Gunicorn"

# Database Layer
primary_db: "PostgreSQL 15+"  # Agent configs, evaluations, metadata
cache_layer: "Redis 7+"       # Session cache, real-time data
time_series: "InfluxDB 2.0"   # Performance metrics, monitoring
document_store: "MongoDB 6+"  # Test cases, reports, logs

# Message Queue & Streaming
message_queue: "Apache Kafka"  # Event streaming, workflow orchestration
task_queue: "Celery with Redis" # Background task processing
real_time: "WebSocket connections" # Live monitoring dashboards

# Storage & CDN
file_storage: "Google Cloud Storage" # Prompt templates, test datasets
cdn: "Google Cloud CDN"       # Static assets, cached responses

# External APIs
ai_provider: "Anthropic Claude API"
healthcare_apis: "FHIR R4, HL7"
monitoring: "Google Cloud Monitoring"
logging: "Google Cloud Logging"

# Security & Compliance
authentication: "OAuth 2.0 + JWT"
authorization: "RBAC with Casbin"
encryption: "AES-256 at rest, TLS 1.3 in transit"
compliance: "HIPAA, SOC 2 Type II"
```

Container Architecture

```yaml
```

```yaml
# Docker Compose Structure
version: '3.8'
services:
  # API Gateway
  sadp-gateway:
    image: "nginx:alpine"
    ports: ["80:80", "443:443"]

  # Core Services
  sadp-api:
    image: "sadp/api:latest"
    replicas: 3
    environment:
      - DATABASE_URL=postgresql://...
      - REDIS_URL=redis://...
      - CLAUDE_API_KEY=${CLAUDE_API_KEY}

  sadp-worker:
    image: "sadp/worker:latest"
    replicas: 5
    environment:
      - CELERY_BROKER=redis://...

  # Data Layer
  postgres:
    image: "postgres:15-alpine"
    environment:
      - POSTGRES_DB=sadp
      - POSTGRES_USER=sadp
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: "redis:7-alpine"
    volumes:
      - redis_data:/data

  mongodb:
    image: "mongo:6"
    volumes:
      - mongo_data:/data/db

  influxdb:
```

```
  image: "influxdb:2.0"
  volumes:
    - influx_data:/var/lib/influxdb2

# Message Queue
kafka:
  image: "confluentinc/cp-kafka:latest"
  environment:
    - KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181

zookeeper:
  image: "confluentinc/cp-zookeeper:latest"
```

API Specifications

Agent Execution API

```
yaml
```

```yaml
# POST /api/v1/agents/{agent_name}/execute
request:
  agent_name: "clinical_agent"
  input_data:
    patient_id: "SARTHI-PT-001"
    primary_diagnosis: "Type 2 Diabetes Mellitus"
    patient_age: 55
    comorbidities: ["Hypertension", "Obesity"]
  context:
    facility_id: "SARTHI-CLINIC-001"
    provider_id: "DR-SMITH-001"
    session_id: "sess_123456"
  options:
    timeout: 30000  # 30 seconds
    priority: "high"
    trace_enabled: true

response:
  execution_id: "exec_789012"
  agent_name: "clinical_agent"
  status: "completed"
  execution_time: 2847  # milliseconds
  result:
    treatment_plan: "Comprehensive treatment plan..."
    confidence_score: 0.94
    clinical_recommendations: [...]
  metadata:
    prompt_version: "v1.2.3"
    claude_model: "claude-sonnet-4-20250514"
    tokens_used: {"input": 1250, "output": 2890}
    cost: 0.045  # USD
  compliance:
    hipaa_compliant: true
    audit_trail_id: "audit_345678"
    phi_processed: true
  performance:
    accuracy_score: 95.2
    latency_percentile: "p95"
    quality_metrics: {...}
```

Workflow Execution API

```
yaml
```

```yaml
# POST /api/v1/workflows/execute
request:
  workflow_name: "patient_intake_complete"
  input_data:
    patient_id: "SARTHI-PT-002"
    intake_form_image: "base64_encoded_image"
    insurance_info: {...}
  workflow_definition:
    steps:
      - agent: "document_processor"
        input_mapping: {"document_image": "intake_form_image"}
        output_key: "extracted_data"
      - agent: "clinical_agent"
        input_mapping: {"clinical_data": "extracted_data"}
        output_key: "treatment_plan"
      - agent: "billing_agent"
        input_mapping: {"encounter_data": "extracted_data"}
        output_key: "billing_claim"
    parallel_steps: ["clinical_agent", "billing_agent"]
  options:
    timeout: 60000
    priority: "normal"

response:
  workflow_id: "wf_456789"
  status: "completed"
  total_execution_time: 8934  # milliseconds
  steps_completed: 3
  results:
    document_processing:
      execution_time: 4200
      result: {...}
    clinical_analysis:
      execution_time: 3100
      result: {...}
    billing_verification:
      execution_time: 1634
      result: {...}
  workflow_metrics:
    total_cost: 0.127
    accuracy_aggregate: 94.7
    performance_rating: "excellent"
```

Evaluation API

```yaml
# POST /api/v1/evaluation/agents/{agent_name}/test
request:
  agent_name: "clinical_agent"
  test_suite_id: "clinical_comprehensive_v1"
  evaluation_config:
    test_categories: ["basic_functionality", "edge_cases", "compliance"]
    performance_targets:
      accuracy: 95
      latency: 5000
      compliance_score: 100
  options:
    parallel_execution: true
    detailed_reporting: true

response:
  evaluation_id: "eval_789123"
  status: "in_progress"
  estimated_completion: "2025-08-19T10:45:00Z"
  test_cases_total: 47
  test_cases_completed: 0
  real_time_url: "wss://sadp.sarthi.com/evaluations/eval_789123/stream"
```

Monitoring API

```yaml
```

```
# GET /api/v1/monitoring/agents/{agent_name}/metrics
request:
  agent_name: "clinical_agent"
  time_range: "24h"
  metrics: ["accuracy", "latency", "error_rate", "cost"]
  aggregation: "mean"

response:
  agent_name: "clinical_agent"
  time_range: "2025-08-18T10:00:00Z to 2025-08-19T10:00:00Z"
  metrics:
    accuracy:
      current: 94.7
      trend: "stable"
      target: 95.0
      status: "warning"
    latency:
      current: 2847
      p95: 4200
      target: 5000
      status: "healthy"
    error_rate:
      current: 0.023
      trend: "improving"
      target: 0.05
      status: "healthy"
    cost:
      current: 0.045
      daily_total: 127.34
      trend: "increasing"
  alerts:
    active: 1
    details: ["Accuracy below target threshold"]
  recommendations:
    - "Consider prompt optimization to improve accuracy"
    - "Monitor cost trends for potential optimization"
```

Integration with Main Sarthi Application

SDK for Sarthi Application

```
python
```

```python
# Sarthi Application Integration SDK
class SarthiAISDK:
    def __init__(self, api_base_url: str, api_key: str):
        self.api_base_url = api_base_url
        self.api_key = api_key
        self.session = httpx.AsyncClient()

    async def execute_agent(self, agent_name: str, input_data: dict, **kwargs) -> dict:
        """Execute AI agent via SADP"""
        response = await self.session.post(
            f"{self.api_base_url}/api/v1/agents/{agent_name}/execute",
            json={"input_data": input_data, **kwargs},
            headers={"Authorization": f"Bearer {self.api_key}"}
        )
        return response.json()

    async def execute_workflow(self, workflow_name: str, input_data: dict) -> dict:
        """Execute multi-agent workflow"""
        response = await self.session.post(
            f"{self.api_base_url}/api/v1/workflows/execute",
            json={"workflow_name": workflow_name, "input_data": input_data},
            headers={"Authorization": f"Bearer {self.api_key}"}
        )
        return response.json()

    async def get_agent_performance(self, agent_name: str, time_range: str = "1h") -> dict:
        """Get real-time agent performance metrics"""
        response = await self.session.get(
            f"{self.api_base_url}/api/v1/monitoring/agents/{agent_name}/metrics",
            params={"time_range": time_range},
            headers={"Authorization": f"Bearer {self.api_key}"}
        )
        return response.json()

# Usage in Sarthi Application
class SarthiPatientService:
    def __init__(self):
        self.ai_sdk = SarthiAISDK(
            api_base_url="https://sadp.sarthi.com",
            api_key=os.getenv("SADP_API_KEY")
        )

    async def process_patient_intake(self, patient_data: dict) -> dict:
        """Process patient intake using SADP agents"""

        # Execute patient intake workflow
        workflow_result = await self.ai_sdk.execute_workflow(
            workflow_name="patient_intake_complete",
            input_data=patient_data
        )

        # Extract results
        document_analysis = workflow_result["results"]["document_processing"]
        clinical_summary = workflow_result["results"]["clinical_analysis"]
        billing_verification = workflow_result["results"]["billing_verification"]

        return {
            "patient_id": patient_data["patient_id"],
            "intake_summary": clinical_summary,
            "billing_ready": billing_verification,
            "processing_time": workflow_result["total_execution_time"],
            "ai_confidence": workflow_result["workflow_metrics"]["accuracy_aggregate"]
        }

    async def get_ai_performance_dashboard(self) -> dict:
        """Get AI performance metrics for admin dashboard"""

        agents = ["clinical_agent", "billing_agent", "document_processor"]
        performance_data = {}

        for agent in agents:
            metrics = await self.ai_sdk.get_agent_performance(agent, "24h")
            performance_data[agent] = metrics

        return performance_data
```

Deployment Architecture

Google Cloud Platform Deployment

```yaml
```

```yaml
# GCP Services Architecture
gcp_services:
  compute:
    - service: "Google Kubernetes Engine (GKE)"
      purpose: "Container orchestration for SADP services"
      configuration:
        node_pools: 3
        auto_scaling: true
        preemptible_nodes: true

    - service: "Google Cloud Run"
      purpose: "Serverless API endpoints for low-latency requests"
      configuration:
        concurrency: 100
        timeout: 300s

  storage:
    - service: "Google Cloud SQL (PostgreSQL)"
      purpose: "Primary database for agent configs and metadata"
      configuration:
        high_availability: true
        automatic_backups: true

    - service: "Google Cloud Storage"
      purpose: "Prompt templates, test datasets, reports"
      configuration:
        storage_class: "STANDARD"
        versioning: true

  networking:
    - service: "Google Cloud Load Balancer"
      purpose: "API gateway and traffic distribution"
      configuration:
        ssl_termination: true
        health_checks: true

    - service: "Google Cloud CDN"
      purpose: "Cache static assets and frequent responses"

  monitoring:
    - service: "Google Cloud Monitoring"
      purpose: "Infrastructure and application monitoring"
    - service: "Google Cloud Logging"
      purpose: "Centralized logging and audit trails"

  security:
    - service: "Google Cloud IAM"
      purpose: "Access control and service authentication"

    - service: "Google Cloud KMS"
      purpose: "Encryption key management"
```

Kubernetes Deployment Manifests

```
yaml
```

```yaml
# SADP API Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sadp-api
  namespace: sadp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sadp-api
  template:
    metadata:
      labels:
        app: sadp-api
    spec:
      containers:
      - name: sadp-api
        image: gcr.io/sarthi-platform/sadp-api:v1.0.0
        ports:
        - containerPort: 8000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: sadp-secrets
              key: database-url
        - name: CLAUDE_API_KEY
          valueFrom:
            secretKeyRef:
              name: sadp-secrets
              key: claude-api-key
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8000
        initialDelaySeconds: 30
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /ready
          port: 8000
        initialDelaySeconds: 5
        periodSeconds: 5

---
# SADP Service
apiVersion: v1
kind: Service
metadata:
  name: sadp-api-service
  namespace: sadp
spec:
  selector:
    app: sadp-api
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8000
  type: ClusterIP

---
# SADP Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: sadp-ingress
  namespace: sadp
  annotations:
    kubernetes.io/ingress.class: "gce"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  tls:
  - hosts:
    - sadp.sarthi.com
    secretName: sadp-tls
  rules:
  - host: sadp.sarthi.com
    http:
      paths:
```

```
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: sadp-api-service
            port:
              number: 80
```

Security & Compliance

Authentication & Authorization

```python
# JWT-based authentication with RBAC
class SADPAuth:
    roles = {
        "sarthi_admin": [
            "agents:read", "agents:write", "agents:deploy",
            "evaluation:read", "evaluation:write",
            "monitoring:read", "development:read", "development:write"
        ],
        "sarthi_developer": [
            "agents:read", "evaluation:read", "evaluation:write",
            "development:read", "monitoring:read"
        ],
        "sarthi_application": [
            "agents:execute", "workflows:execute",
            "monitoring:read"
        ],
        "sarthi_viewer": [
            "agents:read", "evaluation:read", "monitoring:read"
        ]
    }

    def verify_permission(self, user_role: str, action: str) -> bool:
        return action in self.roles.get(user_role, [])
```

HIPAA Compliance Features

```python
```

```
class HIPAACompliance:
    """HIPAA compliance enforcement for SADP"""

    def audit_log_interaction(self, request_data: dict) -> str:
        """Log all interactions for HIPAA audit trail"""
        audit_entry = {
            "timestamp": datetime.utcnow().isoformat(),
            "user_id": request_data.get("user_id"),
            "action": request_data.get("action"),
            "resource": request_data.get("resource"),
            "phi_accessed": self.detect_phi(request_data),
            "ip_address": request_data.get("ip_address"),
            "user_agent": request_data.get("user_agent")
        }
        return self.store_audit_log(audit_entry)

    def encrypt_sensitive_data(self, data: dict) -> dict:
        """Encrypt PHI and sensitive data"""
        # Implementation for AES-256 encryption
        pass

    def validate_data_access(self, user_id: str, patient_id: str) -> bool:
        """Validate user has permission to access patient data"""
        # Implementation for access control validation
        pass
```

Cost Optimization

Resource Management

```
python
```

```
class SADPResourceOptimizer:
    """Optimize resource usage and costs"""

    def auto_scale_workers(self, current_load: float) -> dict:
        """Auto-scale worker instances based on load"""
        if current_load > 0.8:
            return {"action": "scale_up", "instances": 2}
        elif current_load < 0.3:
            return {"action": "scale_down", "instances": 1}
        return {"action": "maintain", "instances": 0}

    def optimize_claude_api_usage(self, agent_metrics: dict) -> dict:
        """Optimize Claude API token usage"""
        recommendations = []

        if agent_metrics["avg_tokens"] > 3000:
            recommendations.append("Consider prompt length optimization")

        if agent_metrics["api_calls_per_hour"] > 1000:
            recommendations.append("Implement response caching")

        return {"recommendations": recommendations}
```

Benefits of Independent Service Architecture

1. Separation of Concerns

- **Main Sarthi App** focuses on healthcare workflows
- **SADP** focuses on AI agent management and optimization
- Clear boundaries and responsibilities

2. Independent Scaling

- Scale SADP based on AI workload demands
- Scale Sarthi app based on user traffic
- Optimize costs for each service independently

3. Technology Flexibility

- Use different tech stacks optimized for each purpose
- Upgrade AI capabilities without affecting main application

- Experiment with new technologies in isolation

4. **Development Velocity**

- Teams can develop and deploy independently
- Parallel development workflows
- Reduced merge conflicts and dependencies

5. **Reliability & Fault Isolation**

- SADP failures don't bring down main application
- Independent monitoring and alerting
- Easier debugging and troubleshooting

6. **Security & Compliance**

- Dedicated security controls for AI operations
- Isolated compliance auditing
- Reduced attack surface for main application

7. **Cost Optimization**

- Pay only for AI resources when needed
- Independent cost tracking and optimization
- Better resource utilization

This independent service architecture provides maximum flexibility, scalability, and maintainability while enabling your main Sarthi application to leverage advanced AI capabilities through a clean, well-defined API interface.