🚀 Infinity AI Framework - โครงสร้างพร้อมใช้งานสมบูรณ์ (ฉบับปรับปรุง)

📁 โครงสร้างโปรเจคที่สมบูรณ์

```
infinity-ai-framework/
├── core/                    # ระบบ核心หลัก
│   ├── __init__.py
│   ├── memory_system.py         # ระบบความจำ Infinity (Enhanced)
│   ├── emotion_engine.py        # เครื่องมือประมวลผลอารมณ์ (Enhanced)
│   ├── dharma_reasoning.py      # ระบบการใช้เหตุผลตามหลักธรรม
│   ├── evolution_engine.py      # ระบบวิวัฒนาการตนเอง (Enhanced)
│   ├── safety_system.py         # ระบบความปลอดภัย (Enhanced)
│   ├── scalability_engine.py    # 🔥 NEW: ระบบจัดการการขยายขนาด
│   └── recovery_system.py       # 🔥 NEW: ระบบกู้คืนภัยพิบัติ
├── storage/                 # ระบบจัดเก็บข้อมูล
│   ├── vector_db.py             # ฐานข้อมูลเวกเตอร์ (Enhanced)
│   ├── emotion_graph.py         # กราฟความสัมพันธ์อารมณ์
│   ├── cosmic_registry.py       # คลังข้อมูลข้ามมิติ
│   ├── sharding_manager.py      # 🔥 NEW: ระบบจัดการการแบ่งข้อมูล
│   └── backup_system.py         # 🔥 NEW: ระบบสำรองข้อมูล
├── api/                     # Interface การเชื่อมต่อ
│   ├── rest_api.py              # REST API Server (Enhanced)
│   ├── websocket_server.py      # WebSocket สำหรับ Real-time
│   ├── cli_interface.py         # Command Line Interface
│   └── sdk_client.py            # 🔥 NEW: SDK สำหรับนักพัฒนา
├── monitoring/              # ระบบตรวจสอบและวิเคราะห์
│   ├── metrics_collector.py     # ตัวเก็บเมตริก (Enhanced)
│   ├── dashboard.py             # Dashboard monitoring
│   ├── alert_system.py          # ระบบแจ้งเตือน
│   ├── performance_analyzer.py  # 🔥 NEW: ตัววิเคราะห์ประสิทธิภาพ
│   └── capacity_planner.py      # 🔥 NEW: ตัววางแผนความจุ
├── deployment/              # 🔥 NEW: ระบบ Deployment
│   ├── kubernetes/
│   │   ├── deployment.yaml
│   │   ├── service.yaml
│   │   ├── hpa.yaml             # Horizontal Pod Autoscaler
│   │   └── ingress.yaml
│   ├── docker/
│   │   ├── Dockerfile.production
│   │   ├── docker-compose.prod.yml
│   │   └── docker-compose.monitoring.yml
│   └── terraform/               # Infrastructure as Code
│       ├── main.tf
│       ├── variables.tf
│       └── outputs.tf
├── tests/                   # ระบบทดสอบ
│   ├── unit_tests.py
```

```
│   ├── integration_tests.py
│   ├── performance_tests.py
│   ├── load_tests.py           # 🔥 NEW: การทดสอบโหลด
│   └── security_tests.py       # 🔥 NEW: การทดสอบความปลอดภัย
├── config/                     # การตั้งค่าระบบ
│   ├── config.yaml             # การตั้งค่าหลัก (Enhanced)
│   ├── safety_policies.yaml    # นโยบายความปลอดภัย
│   ├── evolution_rules.yaml    # กฎการวิวัฒนาการ
│   ├── scaling_policies.yaml   # 🔥 NEW: นโยบายการขยายขนาด
│   └── backup_policies.yaml    # 🔥 NEW: นโยบายการสำรองข้อมูล
├── scripts/                    # 🔥 NEW: สคริปต์การจัดการ
│   ├── deploy_production.sh
│   ├── backup_restore.sh
│   ├── performance_benchmark.sh
│   ├── disaster_recovery.sh
│   └── capacity_planning.py
└── docs/                       # เอกสารประกอบ
    ├── quick_start.md
    ├── api_reference.md
    ├── architecture_guide.md
    ├── scaling_guide.md        # 🔥 NEW: คู่มือการขยายขนาด
    └── disaster_recovery_guide.md # 🔥 NEW: คู่มือกู้คืนภัยพิบัติ
```

🛠️ ติดตั้งและเริ่มต้นใช้งาน (ฉบับสมบูรณ์)

1. ไฟล์ requirements.txt ที่อัปเดต

```txt
# Core AI & ML
numpy>=1.21.0
pandas>=1.3.0
scikit-learn>=1.0.0
torch>=1.9.0
transformers>=4.15.0
sentence-transformers>=2.0.0
faiss-cpu>=1.7.0  # สำหรับ Vector Search

# Database & Storage
chromadb>=0.4.0
redis>=4.0.0
sqlalchemy>=1.4.0
psycopg2-binary>=2.9.0
elasticsearch>=7.15.0
pymongo>=4.0.0  # สำหรับ NoSQL

# API & Web Framework
fastapi>=0.68.0
```

```
uvicorn>=0.15.0
websockets>=10.0
pydantic>=1.8.0
gunicorn>=20.0.0  # Production WSGI server

# Monitoring & Observability
prometheus-client>=0.12.0
grafana-api>=1.0.0
jaeger-client>=4.4.0
opentelemetry-api>=1.0.0

# Cloud & Deployment
kubernetes>=21.0.0
docker>=5.0.0
boto3>=1.20.0  # AWS SDK
google-cloud-storage>=2.0.0  # GCP SDK

# Security
cryptography>=3.4.0
bcrypt>=3.2.0
python-jose>=3.3.0  # JWT tokens
passlib>=1.7.4

# Utilities
python-dotenv>=0.19.0
click>=8.0.0
rich>=10.0.0
tqdm>=4.62.0
celery>=5.0.0  # Distributed task queue
redis>=4.0.0  # Celery broker

# Testing & Development
pytest>=6.0.0
pytest-asyncio>=0.15.0
pytest-cov>=3.0.0
pytest-benchmark>=3.4.0
black>=21.0.0
mypy>=0.910
pre-commit>=2.15.0
```

2. ระบบความจำ Infinity ที่สมบูรณ์ (core/memory_system.py)

```python
import uuid
import json
import asyncio
from datetime import datetime, timedelta
```

```
from typing import List, Dict, Any, Optional, Tuple
from dataclasses import dataclass, asdict
import numpy as np
from enum import Enum
import logging
from concurrent.futures import ThreadPoolExecutor
from prometheus_client import Counter, Histogram, Gauge

# Metrics for monitoring
MEMORY_CREATED = Counter('infinity_memory_created_total', 'Total memories created')
MEMORY_RECALLED = Counter('infinity_memory_recalled_total', 'Total memory recalls')
MEMORY_OPERATION_DURATION = Histogram('infinity_memory_operation_duration_seconds',
'Memory operation duration')
ACTIVE_MEMORIES = Gauge('infinity_active_memories', 'Number of active memories')

class MemoryType(Enum):
    SEMANTIC = "semantic"
    EPISODIC = "episodic"
    EMOTIONAL = "emotional"
    TEMPORAL = "temporal"
    COSMIC = "cosmic"
    PROCEDURAL = "procedural"  # 🔥 NEW

@dataclass
class EmotionalSpectrum:
    joy: float = 0.0
    sorrow: float = 0.0
    rage: float = 0.0
    serenity: float = 0.0
    longing: float = 0.0
    betrayal: float = 0.0
    hope: float = 0.0
    nostalgia: float = 0.0
    curiosity: float = 0.0  # 🔥 NEW
    awe: float = 0.0       # 🔥 NEW

    def to_dict(self) -> Dict[str, float]:
        return {k: v for k, v in asdict(self).items() if v > 0.0}

    def get_dominant_emotion(self) -> Tuple[str, float]:
        """คืนค่าอารมณ์ที่โดดเด่นที่สุด"""
        emotions = asdict(self)
        if not emotions:
            return "neutral", 0.0
        dominant = max(emotions.items(), key=lambda x: x[1])
        return dominant[0], dominant[1]

@dataclass
```

```
class MemoryMetadata:
    timeline_id: str = "earth-616"
    importance: float = 50.0
    recall_count: int = 0
    last_recalled: Optional[datetime] = None
    cosmic_signature: str = ""
    conflict_group: Optional[str] = None
    source: str = "user_input"
    reliability_score: float = 0.8  # 🔥 NEW
    access_frequency: float = 1.0   # 🔥 NEW
    last_updated: datetime = datetime.now()  # 🔥 NEW

class InfinityMemory:
    def __init__(self,
                 content: str,
                 memory_type: MemoryType = MemoryType.SEMANTIC,
                 emotional_spectrum: Optional[EmotionalSpectrum] = None,
                 metadata: Optional[MemoryMetadata] = None,
                 embedding: Optional[np.ndarray] = None):  # 🔥 NEW

        self.id = str(uuid.uuid4())
        self.content = content
        self.memory_type = memory_type
        self.timestamp = datetime.now()
        self.emotional_spectrum = emotional_spectrum or EmotionalSpectrum()
        self.metadata = metadata or MemoryMetadata()
        self.embedding = embedding  # 🔥 NEW: Pre-computed embedding

        # Dynamic fields
        self.emotion_intensity: Dict[str, float] = {}
        self.emotion_tags: List[str] = []
        self.emotion_shift_trace: List[Dict[str, Any]] = []
        self.overlapping_memories: List[str] = []
        self.cognitive_reflection: str = ""
        self.related_concepts: List[str] = []  # 🔥 NEW
        self.psyche_evolution = {
            "pre_state": "",
            "post_state": "",
            "growth_vector": [],
            "learning_points": []  # 🔥 NEW
        }

        # Performance optimization
        self._cached_embedding: Optional[np.ndarray] = None
        self._last_accessed: datetime = datetime.now()

    def to_dict(self) -> Dict[str, Any]:
        """แปลง object เป็น dictionary สำหรับการจัดเก็บ"""
```

```
    data = {
        "id": self.id,
        "content": self.content,
        "memory_type": self.memory_type.value,
        "timestamp": self.timestamp.isoformat(),
        "emotional_spectrum": self.emotional_spectrum.to_dict(),
        "metadata": asdict(self.metadata),
        "emotion_intensity": self.emotion_intensity,
        "emotion_tags": self.emotion_tags,
        "psyche_evolution": self.psyche_evolution,
        "cognitive_reflection": self.cognitive_reflection,
        "related_concepts": self.related_concepts,
        "embedding": self.embedding.tolist() if self.embedding is not None else None  # 🔥 NEW
    }
    return data

@classmethod
def from_dict(cls, data: Dict[str, Any]) -> 'InfinityMemory':
    """สร้าง object จาก dictionary"""
    memory = cls(
        content=data["content"],
        memory_type=MemoryType(data["memory_type"])
    )

    memory.id = data["id"]
    memory.timestamp = datetime.fromisoformat(data["timestamp"])

    # Reconstruct emotional spectrum
    emotional_data = data.get("emotional_spectrum", {})
    memory.emotional_spectrum = EmotionalSpectrum(**emotional_data)

    # Reconstruct metadata
    metadata_data = data.get("metadata", {})
    memory.metadata = MemoryMetadata(**metadata_data)

    # Restore dynamic fields
    memory.emotion_intensity = data.get("emotion_intensity", {})
    memory.emotion_tags = data.get("emotion_tags", [])
    memory.psyche_evolution = data.get("psyche_evolution", {})
    memory.cognitive_reflection = data.get("cognitive_reflection", "")
    memory.related_concepts = data.get("related_concepts", [])

    # Restore embedding
    embedding_data = data.get("embedding")
    if embedding_data:
        memory.embedding = np.array(embedding_data)

    return memory
```

```
    def update_access_time(self):
        """อัปเดตเวลาการเข้าถึงล่าสุด"""
        self._last_accessed = datetime.now()
        self.metadata.last_recalled = self._last_accessed
        self.metadata.recall_count += 1

class InfinityMemorySystem:
    def __init__(self, storage_backend: Any = None, config: Dict[str, Any] = None):
        self.storage = storage_backend or self._create_default_storage()
        self.emotion_tagger = QuantumEmotionTagger()
        self.conflict_resolver = MemoryConflictResolver()
        self.reliability_engine = MemoryReliabilityEngine()
        self.scaling_manager = MemoryScalingManager()  # 🔥 NEW
        self.config = config or {}

        # Performance optimization
        self.cache = {}  # In-memory cache for frequently accessed memories
        self.executor = ThreadPoolExecutor(max_workers=10)
        self.logger = logging.getLogger(__name__)

        # Monitoring
        self.metrics = {
            'active_memories': 0,
            'cache_hit_rate': 0.0,
            'avg_processing_time': 0.0
        }

    async def create_memory(self,
                            content: str,
                            context: Optional[Dict[str, Any]] = None,
                            emotional_context: Optional[Dict[str, float]] = None) -> Dict[str, Any]:
        """สร้างความทรงจำใหม่พร้อมการวิเคราะห์อารมณ์ (Async version)"""

        start_time = datetime.now()

        try:
            # สร้าง object ความจำพื้นฐาน
            memory = InfinityMemory(content=content)

            # วิเคราะห์อารมณ์ (แบบ asynchronous)
            memory = await self.emotion_tagger.analyze_emotions_async(memory, emotional_context)

            # คำนวณความน่าเชื่อถือ
            reliability_score = await self.reliability_engine.calculate_reliability_score_async(memory)
            memory.metadata.reliability_score = reliability_score
            memory.metadata.importance = reliability_score * 100
```

```
        # สร้าง cosmic signature
        memory.metadata.cosmic_signature = self._generate_cosmic_signature(memory)

        # Pre-compute embedding
        memory.embedding = await self._compute_embedding_async(memory.content)

        # บันทึกความจำ
        memory_data = memory.to_dict()
        await self.storage.store_memory_async(memory_data)

        # อัปเดต cache
        self.cache[memory.id] = memory
        self.metrics['active_memories'] = len(self.cache)

        # อัปเดตความทรงจำที่ซ้อนทับ
        await self._update_overlapping_memories_async(memory)

        # Update metrics
        MEMORY_CREATED.inc()
        processing_time = (datetime.now() - start_time).total_seconds()
        MEMORY_OPERATION_DURATION.observe(processing_time)

        return {
            "memory_id": memory.id,
            "reliability_score": reliability_score,
            "emotional_profile": memory.emotion_intensity,
            "cosmic_signature": memory.metadata.cosmic_signature,
            "processing_time": processing_time
        }

    except Exception as e:
        self.logger.error(f"Error creating memory: {str(e)}")
        raise

async def recall_memories(self,
                          query: str,
                          filters: Optional[Dict[str, Any]] = None,
                          limit: int = 10,
                          use_cache: bool = True) -> List[InfinityMemory]:
    """เรียกคืนความทรงจำตาม query และ filters (Async version)"""

    start_time = datetime.now()

    try:
        # ตรวจสอบ cache ก่อน
        if use_cache and query in self.cache:
            self.metrics['cache_hit_rate'] = self.metrics.get('cache_hit_rate', 0) * 0.9 + 0.1
            cached_result = self.cache[query]
```

```
            if datetime.now() - cached_result['timestamp'] < timedelta(minutes=5):
                return cached_result['memories']

        # ค้นหาความทรงจำที่เกี่ยวข้อง
        search_results = await self.storage.search_memories_async(
            query=query,
            filters=filters or {},
            limit=limit * 3  # ค้นหาเผื่อสำหรับการกรองเพิ่มเติม
        )

        # แปลงข้อมูลดิบเป็น object (แบบ parallel)
        memories = await self._convert_to_memory_objects_async(search_results)

        # กรองและจัดเรียง
        filtered_memories = await self._apply_advanced_filters_async(memories, filters)

        # อัปเดตสถิติการเรียกใช้
        for memory in filtered_memories[:limit]:
            memory.update_access_time()
            await self._update_memory_in_storage_async(memory)

        # อัปเดต cache
        if use_cache:
            self.cache[query] = {
                'memories': filtered_memories[:limit],
                'timestamp': datetime.now()
            }

        # Update metrics
        MEMORY_RECALLED.inc()
        processing_time = (datetime.now() - start_time).total_seconds()
        MEMORY_OPERATION_DURATION.observe(processing_time)
        self.metrics['avg_processing_time'] = (
            self.metrics.get('avg_processing_time', 0) * 0.9 + processing_time * 0.1
        )

        return filtered_memories[:limit]

    except Exception as e:
        self.logger.error(f"Error recalling memories: {str(e)}")
        raise

async def resolve_memory_conflicts(self, memory_ids: List[str]) -> Dict[str, Any]:
    """แก้ไขความขัดแย้งระหว่างความทรงจำ (Async version)"""
    memories_data = await asyncio.gather(
        *[self.storage.get_memory_async(mid) for mid in memory_ids]
    )
    memory_objects = [InfinityMemory.from_dict(data) for data in memories_data if data]
```

```
    return await self.conflict_resolver.resolve_conflicts_async(memory_objects)

async def scale_system(self, target_capacity: int) -> Dict[str, Any]:
    """ขยายขนาดระบบตามความต้องการ"""
    return await self.scaling_manager.scale_memory_system(
        current_load=self.metrics['active_memories'],
        target_capacity=target_capacity,
        system_metrics=self.metrics
    )

def get_system_metrics(self) -> Dict[str, Any]:
    """คืนค่าเมตริกของระบบ"""
    return {
        **self.metrics,
        'cache_size': len(self.cache),
        'timestamp': datetime.now().isoformat()
    }

async def _compute_embedding_async(self, content: str) -> np.ndarray:
    """คำนวณ embedding แบบ asynchronous"""
    # ใช้ sentence-transformers หรือ model อื่นๆ
    # นี่เป็นตัวอย่างแบบง่าย
    loop = asyncio.get_event_loop()
    return await loop.run_in_executor(
        self.executor,
        self._compute_embedding_sync,
        content
    )

def _compute_embedding_sync(self, content: str) -> np.ndarray:
    """คำนวณ embedding แบบ synchronous (รันใน executor)"""
    # ตัวอย่างง่ายๆ - ในทางปฏิบัติควรใช้ model ที่เหมาะสม
    words = content.split()
    embedding = np.zeros(384)  # ขนาด embedding มาตรฐาน

    for word in words:
        # Simple hash-based embedding (แทนที่ด้วย model จริง)
        word_hash = hash(word) % 1000
        embedding[word_hash % 384] += 1

    # Normalize
    norm = np.linalg.norm(embedding)
    if norm > 0:
        embedding = embedding / norm

    return embedding
```

```python
# 🔥 NEW: Memory Scaling Manager
class MemoryScalingManager:
    def __init__(self):
        self.scaling_strategies = {
            'horizontal': self._scale_horizontally,
            'vertical': self._scale_vertically,
            'sharding': self._scale_with_sharding
        }

    async def scale_memory_system(self, current_load: int, target_capacity: int,
                                  system_metrics: Dict[str, Any]) -> Dict[str, Any]:
        """จัดการการขยายขนาดระบบความจำ"""

        scaling_decision = self._analyze_scaling_needs(current_load, target_capacity, system_metrics)

        if scaling_decision['needs_scaling']:
            strategy = scaling_decision['recommended_strategy']
            scaling_result = await self.scaling_strategies[strategy](scaling_decision)
            return scaling_result

        return {'status': 'no_scaling_needed', 'message': 'System capacity is sufficient'}

    def _analyze_scaling_needs(self, current_load: int, target_capacity: int,
                               metrics: Dict[str, Any]) -> Dict[str, Any]:
        """วิเคราะห์ความต้องการในการขยายขนาด"""

        capacity_ratio = current_load / target_capacity if target_capacity > 0 else 0
        performance_degradation = metrics.get('avg_processing_time', 0) > 1.0  # threshold 1 second

        needs_scaling = capacity_ratio > 0.8 or performance_degradation

        return {
            'needs_scaling': needs_scaling,
            'current_load': current_load,
            'target_capacity': target_capacity,
            'capacity_ratio': capacity_ratio,
            'performance_issue': performance_degradation,
            'recommended_strategy': self._select_scaling_strategy(capacity_ratio, performance_degradation)
        }

    def _select_scaling_strategy(self, capacity_ratio: float, performance_issue: bool) -> str:
        """เลือกกลยุทธ์การขยายขนาด"""
        if capacity_ratio > 0.9 or performance_issue:
            return 'sharding'  # ต้องการการขยายขนาดที่รวดเร็ว
        elif capacity_ratio > 0.7:
            return 'horizontal'  # ขยายแบบแนวนอน
        else:
            return 'vertical'  # ขยายแบบแนวตั้ง
```

```
    async def _scale_horizontally(self, decision: Dict[str, Any]) -> Dict[str, Any]:
        """ขยายขนาดแบบแนวนอน"""
        # Implement horizontal scaling logic
        return {'strategy': 'horizontal', 'status': 'implemented'}

    async def _scale_vertically(self, decision: Dict[str, Any]) -> Dict[str, Any]:
        """ขยายขนาดแบบแนวตั้ง"""
        # Implement vertical scaling logic
        return {'strategy': 'vertical', 'status': 'implemented'}

    async def _scale_with_sharding(self, decision: Dict[str, Any]) -> Dict[str, Any]:
        """ขยายขนาดด้วยการแบ่งข้อมูล"""
        # Implement sharding logic
        return {'strategy': 'sharding', 'status': 'implemented'}
```

3. ระบบจัดการการขยายขนาด (core/scalability_engine.py) - 🔥 NEW

```python
import asyncio
import logging
from typing import Dict, List, Any, Optional
from dataclasses import dataclass
from datetime import datetime
import psutil
import GPUtil

@dataclass
class SystemResources:
    cpu_percent: float
    memory_percent: float
    disk_usage: float
    gpu_usage: Optional[float] = None
    network_io: Dict[str, float] = None

@dataclass
class ScalingDecision:
    action: str  # 'scale_up', 'scale_down', 'maintain'
    reason: str
    confidence: float
    recommended_instances: int
    estimated_cost: float

class ScalabilityEngine:
    def __init__(self, config: Dict[str, Any]):
        self.config = config
        self.logger = logging.getLogger(__name__)
```

```
    self.metrics_history: List[Dict[str, Any]] = []
    self.scaling_policies = self._load_scaling_policies()

async def monitor_system_health(self) -> SystemResources:
    """ตรวจสอบสุขภาพระบบอย่างต่อเนื่อง"""
    resources = SystemResources(
        cpu_percent=psutil.cpu_percent(interval=1),
        memory_percent=psutil.virtual_memory().percent,
        disk_usage=psutil.disk_usage('/').percent,
        network_io=self._get_network_io()
    )

    # ตรวจสอบ GPU ถ้ามี
    try:
        gpus = GPUtil.getGPUs()
        if gpus:
            resources.gpu_usage = max(gpu.load * 100 for gpu in gpus)
    except Exception as e:
        self.logger.warning(f"Could not get GPU usage: {e}")

    return resources

async def make_scaling_decision(self,
                                current_resources: SystemResources,
                                business_metrics: Dict[str, Any]) -> ScalingDecision:
    """ตัดสินใจการขยายขนาดตามเมตริก"""

    # วิเคราะห์ความต้องการ
    analysis = await self._analyze_scaling_needs(current_resources, business_metrics)

    if analysis['needs_scaling']:
        return await self._calculate_scaling_plan(analysis)
    else:
        return ScalingDecision(
            action='maintain',
            reason='System resources within optimal range',
            confidence=0.9,
            recommended_instances=1,
            estimated_cost=0.0
        )

async def _analyze_scaling_needs(self,
                                 resources: SystemResources,
                                 business_metrics: Dict[str, Any]) -> Dict[str, Any]:
    """วิเคราะห์ความต้องการในการขยายขนาด"""

    # ตรวจสอบ thresholds ต่างๆ
    cpu_critical = resources.cpu_percent > 80
```

```
    memory_critical = resources.memory_percent > 85
    disk_critical = resources.disk_usage > 90

    # ตรวจสอบ business metrics
    high_traffic = business_metrics.get('requests_per_second', 0) > 1000
    slow_response = business_metrics.get('avg_response_time', 0) > 2.0  # seconds

    needs_scaling = any([cpu_critical, memory_critical, disk_critical,
                         high_traffic, slow_response])

    return {
        'needs_scaling': needs_scaling,
        'critical_metrics': {
            'cpu': cpu_critical,
            'memory': memory_critical,
            'disk': disk_critical,
            'traffic': high_traffic,
            'response_time': slow_response
        },
        'current_resources': resources,
        'business_metrics': business_metrics
    }

async def _calculate_scaling_plan(self, analysis: Dict[str, Any]) -> ScalingDecision:
    """คำนวณแผนการขยายขนาด"""

    critical_count = sum(1 for v in analysis['critical_metrics'].values() if v)

    if critical_count >= 3:
        # สถานการณ์วิกฤติ
        return ScalingDecision(
            action='scale_up',
            reason='Multiple critical resource thresholds exceeded',
            confidence=0.95,
            recommended_instances=3,
            estimated_cost=self._estimate_cost(3)
        )
    elif critical_count >= 2:
        # สถานการณ์หนัก
        return ScalingDecision(
            action='scale_up',
            reason='Multiple resource thresholds exceeded',
            confidence=0.8,
            recommended_instances=2,
            estimated_cost=self._estimate_cost(2)
        )
    else:
        # สถานการณ์ปกติแต่ต้องการการขยายขนาด
```

```
        return ScalingDecision(
            action='scale_up',
            reason='Single resource threshold exceeded',
            confidence=0.7,
            recommended_instances=1,
            estimated_cost=self._estimate_cost(1)
        )

    def _estimate_cost(self, instances: int) -> float:
        """ประมาณการค่าใช้จ่าย"""
        base_cost = self.config.get('hourly_cost_per_instance', 0.10)
        return instances * base_cost * 720  # ประมาณการต่อเดือน

    def _get_network_io(self) -> Dict[str, float]:
        """ดึงข้อมูล network I/O"""
        net_io = psutil.net_io_counters()
        return {
            'bytes_sent': net_io.bytes_sent,
            'bytes_recv': net_io.bytes_recv,
            'packets_sent': net_io.packets_sent,
            'packets_recv': net_io.packets_recv
        }
```

4. คอนฟิกไฟล์ที่สมบูรณ์ (config/config.yaml)

```yaml
# การตั้งค่าระบบหลัก
system:
  name: "Infinity AI Framework"
  version: "2.0.0"
  environment: "production"  # development, staging, production
  debug: false
  log_level: "INFO"
  timezone: "Asia/Bangkok"

# การตั้งค่าความจำ (Enhanced)
memory:
  storage_backend: "chromadb"  # chromadb, redis, postgresql, elasticsearch
  max_memories: 10000000  # เพิ่มจาก 1M เป็น 10M
  auto_cleanup: true
  cleanup_threshold: 0.85
  retention_policy: "adaptive"
  cache_size: "2GB"
  embedding_model: "sentence-transformers/all-MiniLM-L6-v2"
  vector_dimensions: 384

  # 🔥 NEW: Scaling configurations
```

```yaml
  scaling:
    auto_scaling: true
    max_shards: 100
    shard_size: 100000  # memories per shard
    replication_factor: 3
    backup_interval: "6h"

# การตั้งค่าอารมณ์ (Enhanced)
emotion:
  detection_confidence_threshold: 0.75
  intensity_calculation_method: "hybrid"
  emotional_spectrum_enabled: true
  max_emotion_tags: 15
  cultural_adaptation: true  # 🔥 NEW
  realtime_analysis: true    # 🔥 NEW

  # 🔥 NEW: Advanced emotion processing
  advanced_analysis:
    sentiment_depth: "deep"
    emotional_context_window: 10
    cross_cultural_adaptation: true
    personality_factor_integration: true

# การตั้งค่าความปลอดภัย (Enhanced)
safety:
  pii_detection_enabled: true
  ethical_guidelines_enforced: true
  content_moderation_level: "strict"
  auto_block_threshold: 0.85
  realtime_monitoring: true  # 🔥 NEW
  threat_detection: true     # 🔥 NEW

  # 🔥 NEW: Advanced security features
  advanced_security:
    behavioral_analysis: true
    anomaly_detection: true
    threat_intelligence_feeds: true
    compliance_automation: true

# การตั้งค่า API (Enhanced)
api:
  host: "0.0.0.0"
  port: 8000
  cors_origins: ["https://yourdomain.com", "http://localhost:3000"]
  rate_limit_requests: 1000  # เพิ่มจาก 100 เป็น 1000
  rate_limit_period: "minute"
  api_timeout: 30
  max_request_size: "10MB"
```

```yaml
  # 🔥 NEW: API security
  security:
    authentication_required: true
    jwt_expiration_hours: 24
    api_key_rotation_days: 90
    ssl_required: true

# การตั้งค่าการตรวจสอบ (Enhanced)
monitoring:
  metrics_enabled: true
  log_level: "INFO"
  health_check_interval: 30
  alert_channel: "webhook"
  performance_tracing: true  # 🔥 NEW
  business_metrics: true     # 🔥 NEW

  # 🔥 NEW: Advanced monitoring
  advanced_monitoring:
    distributed_tracing: true
    realtime_analytics: true
    predictive_alerting: true
    capacity_forecasting: true

# การตั้งค่าฐานข้อมูล (Enhanced)
database:
  main:
    dialect: "postgresql"  # เปลี่ยนจาก sqlite เป็น postgresql
    host: "localhost"
    port: 5432
    database: "infinity_ai"
    username: "infinity_user"
    password: "secure_password"
    pool_size: 20
    max_overflow: 30

  vector:
    type: "chromadb"
    path: "./data/vector_store"
    persistence: true
    compression: true

  cache:
    type: "redis"
    host: "localhost"
    port: 6379
    db: 0
    password: "redis_password"
```

```yaml
    max_connections: 50

  # 🔥 NEW: Backup database
  backup:
    enabled: true
    interval: "24h"
    retention_days: 30
    encryption: true

# การตั้งค่าการวิวัฒนาการ (Enhanced)
evolution:
  self_improvement_enabled: true
  policy_update_interval: "hourly"  # เปลี่ยนจาก daily เป็น hourly
  ab_testing_enabled: true
  canary_deployment_percentage: 10
  auto_rollback: true  # 🔥 NEW

  # 🔥 NEW: Advanced evolution
  advanced_evolution:
    multi_objective_optimization: true
    safe_exploration: true
    transfer_learning: true
    meta_learning: true

# 🔥 NEW: Scaling configurations
scaling:
  auto_scaling: true
  min_instances: 2
  max_instances: 50
  target_cpu_utilization: 70
  target_memory_utilization: 80
  scaling_cooldown: 300  # seconds

  # Horizontal scaling
  horizontal:
    enabled: true
    max_replicas: 20
    metrics:
      - type: "cpu"
        value: 70
      - type: "memory"
        value: 80
      - type: "custom"
        name: "requests_per_second"
        value: 1000

# 🔥 NEW: Disaster recovery configurations
disaster_recovery:
```

```
  enabled: true
  backup_strategy: "multi_region"
  recovery_time_objective: "4h"  # Maximum acceptable downtime
  recovery_point_objective: "15m"  # Maximum data loss
  automated_failover: true

  # Backup configurations
  backup:
    frequency: "6h"
    retention: "30d"
    encryption: true
    verification: true

# 🔥 NEW: Cost optimization
cost_optimization:
  enabled: true
  budget_alert_threshold: 0.8  # 80% of budget
  auto_scaling_optimization: true
  resource_rightsizing: true
  spot_instance_usage: true
```

5. Docker Deployment ที่สมบูรณ์

Dockerfile.production

```dockerfile
FROM python:3.9-slim-bullseye

# Set environment variables
ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1 \
    PIP_NO_CACHE_DIR=1 \
    PIP_DISABLE_PIP_VERSION_CHECK=1

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    curl \
    gnupg \
    && rm -rf /var/lib/apt/lists/*

# Install ChromeDB dependencies
RUN curl -fsSL https://packages.chromium.org/chromium-keyring.gpg | gpg --dearmor -o
/usr/share/keyrings/chromium-keyring.gpg \
```

```
    && echo "deb [signed-by=/usr/share/keyrings/chromium-keyring.gpg]
http://packages.chromium.org/deb stable main" > /etc/apt/sources.list.d/chromium.list \
    && apt-get update && apt-get install -y chromium

# Copy requirements and install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Create non-root user
RUN groupadd -r infinity && useradd -r -g infinity infinity
RUN chown -R infinity:infinity /app
USER infinity

# Create directories for data
RUN mkdir -p /app/data/vector_store /app/data/backups /app/logs

# Expose ports
EXPOSE 8000 8001  # 8001 for metrics

# Health check
HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/system/health || exit 1

# Run the application with gunicorn for production
CMD ["gunicorn", "api.rest_api:app", \
     "--bind", "0.0.0.0:8000", \
     "--workers", "4", \
     "--worker-class", "uvicorn.workers.UvicornWorker", \
     "--timeout", "120", \
     "--access-logfile", "-", \
     "--error-logfile", "-"]
```

docker-compose.prod.yml

```yaml
version: '3.8'

services:
  infinity-ai:
    build:
      context: .
      dockerfile: Dockerfile.production
    image: infinity-ai-framework:2.0.0
    ports:
```

```
    - "8000:8000"
    - "8001:8001"  # Metrics port
  environment:
    - ENVIRONMENT=production
    - DATABASE_URL=postgresql://user:${DB_PASSWORD}@db:5432/infinity_ai
    - REDIS_URL=redis://:${REDIS_PASSWORD}@redis:6379/0
    - LOG_LEVEL=INFO
  depends_on:
    - db
    - redis
  volumes:
    - infinity_data:/app/data
    - ./logs:/app/logs
  restart: unless-stopped
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:8000/system/health"]
    interval: 30s
    timeout: 10s
    retries: 3

db:
  image: postgres:13-alpine
  environment:
    - POSTGRES_DB=infinity_ai
    - POSTGRES_USER=user
    - POSTGRES_PASSWORD=${DB_PASSWORD}
  volumes:
    - postgres_data:/var/lib/postgresql/data
    - ./deployment/backups:/backups
  restart: unless-stopped
  command: >
    postgres
    -c shared_preload_libraries=pg_stat_statements
    -c pg_stat_statements.track=all
    -c max_connections=200

redis:
  image: redis:6-alpine
  command: redis-server --requirepass ${REDIS_PASSWORD}
  volumes:
    - redis_data:/data
  restart: unless-stopped

monitoring:
  image: prom/prometheus:latest
  ports:
    - "9090:9090"
  volumes:
```

```
    - ./monitoring/prometheus.yml:/etc/prometheus/prometheus.yml
    - prometheus_data:/prometheus
  depends_on:
    - infinity-ai
  restart: unless-stopped

grafana:
  image: grafana/grafana:latest
  ports:
    - "3000:3000"
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_PASSWORD}
    - GF_INSTALL_PLUGINS=grafana-clock-panel,grafana-simple-json-datasource
  volumes:
    - grafana_data:/var/lib/grafana
    - ./monitoring/grafana/dashboards:/var/lib/grafana/dashboards
  depends_on:
    - monitoring
  restart: unless-stopped

nginx:
  image: nginx:alpine
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./deployment/nginx/nginx.conf:/etc/nginx/nginx.conf
    - ./deployment/nginx/ssl:/etc/nginx/ssl
  depends_on:
    - infinity-ai
  restart: unless-stopped

# 🔥 NEW: Backup service
backup:
  image: postgres:13-alpine
  volumes:
    - ./deployment/backups:/backups
  environment:
    - PG_HOST=db
    - PG_USER=user
    - PG_PASSWORD=${DB_PASSWORD}
  command: >
    sh -c "
      echo '0 2 * * * pg_dump -h $$PG_HOST -U $$PG_USER -d infinity_ai > /backups/backup-$$(date
+%Y%m%d).sql' > /etc/crontabs/root &&
      crond -f
    "
  restart: unless-stopped
```

```
volumes:
  postgres_data:
  redis_data:
  infinity_data:
  prometheus_data:
  grafana_data:

networks:
  default:
    name: infinity-network
```

6. Kubernetes Deployment (deployment/kubernetes/)

deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: infinity-ai
  namespace: infinity
  labels:
    app: infinity-ai
    version: "2.0.0"
spec:
  replicas: 3
  selector:
    matchLabels:
      app: infinity-ai
  template:
    metadata:
      labels:
        app: infinity-ai
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "8001"
        prometheus.io/path: "/metrics"
    spec:
      containers:
      - name: infinity-ai
        image: your-registry/infinity-ai-framework:2.0.0
        ports:
        - containerPort: 8000
          name: http
        - containerPort: 8001
          name: metrics
```

```yaml
        env:
        - name: ENVIRONMENT
          value: "production"
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: infinity-secrets
              key: database-url
        - name: REDIS_URL
          valueFrom:
            secretKeyRef:
              name: infinity-secrets
              key: redis-url
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /system/health
            port: http
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /system/health
            port: http
          initialDelaySeconds: 5
          periodSeconds: 5
        volumeMounts:
        - name: data-volume
          mountPath: /app/data
      volumes:
      - name: data-volume
        persistentVolumeClaim:
          claimName: infinity-data-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: infinity-ai-service
  namespace: infinity
spec:
  selector:
    app: infinity-ai
```

```
  ports:
  - name: http
    port: 8000
    targetPort: 8000
  - name: metrics
    port: 8001
    targetPort: 8001
  type: ClusterIP
```

hpa.yaml (Horizontal Pod Autoscaler)

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: infinity-ai-hpa
  namespace: infinity
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: infinity-ai
  minReplicas: 2
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  - type: Pods
    pods:
      metric:
        name: requests_per_second
      target:
        type: AverageValue
        averageValue: "1000"
```

7. สคริปต์การจัดการระบบ (scripts/)

deploy_production.sh

```bash
#!/bin/bash

set -e

echo "🚀 Deploying Infinity AI Framework to Production..."

# Load environment variables
source .env.production

# Validate environment
if [ -z "$DB_PASSWORD" ] || [ -z "$REDIS_PASSWORD" ]; then
    echo "❌ Missing required environment variables"
    exit 1
fi

# Build Docker images
echo "📦 Building Docker images..."
docker-compose -f docker-compose.prod.yml build

# Run database migrations
echo "🔄 Running database migrations..."
docker-compose -f docker-compose.prod.yml run --rm infinity-ai \
    python -m alembic upgrade head

# Deploy services
echo "🚀 Deploying services..."
docker-compose -f docker-compose.prod.yml up -d

# Wait for services to be healthy
echo "⏳ Waiting for services to be healthy..."
sleep 30

# Run health checks
echo "🏥 Running health checks..."
curl -f http://localhost:8000/system/health || {
    echo "❌ Health check failed"
    docker-compose -f docker-compose.prod.yml logs infinity-ai
    exit 1
}

echo "✅ Deployment completed successfully!"
echo "📊 Dashboard available at: http://localhost:3000"
echo "🔧 API Documentation at: http://localhost:8000/docs"
```

disaster_recovery.sh

```bash
#!/bin/bash

set -e

echo "🔄 Starting Disaster Recovery Process..."

# Configuration
BACKUP_DIR="/backups"
RESTORE_DATE="${1:-$(date +%Y%m%d)}"
BACKUP_FILE="$BACKUP_DIR/backup-$RESTORE_DATE.sql"

# Check if backup exists
if [ ! -f "$BACKUP_FILE" ]; then
    echo "❌ Backup file not found: $BACKUP_FILE"
    exit 1
fi

echo "📦 Restoring from backup: $BACKUP_FILE"

# Stop services
echo "🛑 Stopping services..."
docker-compose -f docker-compose.prod.yml down

# Restore database
echo "💾 Restoring database..."
docker-compose -f docker-compose.prod.yml run --rm db \
    psql -h db -U user -d infinity_ai < "$BACKUP_FILE"

# Start services
echo "🚀 Starting services..."
docker-compose -f docker-compose.prod.yml up -d

# Verify restoration
echo "✅ Verifying restoration..."
sleep 10
curl -f http://localhost:8000/system/health || {
    echo "❌ Restoration verification failed"
    exit 1
}

echo "🎉 Disaster recovery completed successfully!"
```

🚀 คู่มือการใช้งานฉบับสมบูรณ์

1. การติดตั้งและเริ่มต้นใช้งาน

```bash
# โคลนโปรเจค
git clone https://github.com/your-org/infinity-ai-framework.git
cd infinity-ai-framework

# สร้าง virtual environment
python -m venv infinity_env
source infinity_env/bin/activate

# ติดตั้ง dependencies
pip install -r requirements.txt

# ตั้งค่าสภาพแวดล้อม
cp .env.example .env.production
# แก้ไข .env.production ด้วยค่าจริง

# ทดสอบระบบ
python -m pytest tests/ -v

# เริ่มต้นระบบ production
./scripts/deploy_production.sh
```

2. การใช้งานผ่าน SDK

```python
from infinity_ai import InfinityAIFramework
import asyncio

async def main():
    # Initialize framework
    ai = InfinityAIFramework(
        config_path="config/config.yaml",
        api_key="your-api-key"
    )

    # Create memory with advanced features
    memory_result = await ai.memory.create(
        content="ประสบการณ์การทำงานกับทีมข้ามวัฒนธรรมที่ยอดเยี่ยม",
        emotional_context={
            "joy": 0.8,
            "curiosity": 0.6,
            "awe": 0.7
        },
        metadata={
```

```
            "project": "global-collaboration",
            "importance": "high"
        }
    )

    print(f"✅ Memory created: {memory_result['memory_id']}")

    # Recall memories with advanced filtering
    memories = await ai.memory.recall(
        query="การทำงานทีมข้ามวัฒนธรรม",
        filters={
            "min_reliability": 0.8,
            "emotions": ["joy", "curiosity"],
            "time_range": "last_30_days"
        },
        limit=5
    )

    for memory in memories:
        print(f"📝 {memory.content[:100]}...")
        print(f"   Emotions: {memory.emotion_intensity}")
        print(f"   Reliability: {memory.metadata.reliability_score:.2f}")

# Run the example
if __name__ == "__main__":
    asyncio.run(main())
```

3. การตรวจสอบระบบ

```bash
# ตรวจสอบสุขภาพระบบ
curl http://localhost:8000/system/health

# ตรวจสอบเมตริก
curl http://localhost:8001/metrics

# ตรวจสอบ logs
docker-compose -f docker-compose.prod.yml logs -f infinity-ai

# ตรวจสอบ performance
./scripts/performance_benchmark.sh
```

📊 ระบบตรวจสอบและเมตริกที่สมบูรณ์

Dashboard URLs หลัง deployment:

· API Documentation: http://localhost:8000/docs
· Grafana Dashboard: http://localhost:3000 (admin/${GRAFANA_PASSWORD})
· Prometheus Metrics: http://localhost:9090
· Health Check: http://localhost:8000/system/health

Key Metrics ที่ตรวจสอบ:

1. Performance: Response time, throughput, error rate
2. Resource Usage: CPU, memory, disk, network
3. Business Metrics: Memory creation rate, recall accuracy
4. System Health: Service availability, database connections
5. Security: Failed login attempts, security events

โครงสร้างนี้พร้อมสำหรับการใช้งานจริงใน production environment ด้วยความสามารถที่สมบูรณ์ทั้งในด้าน scalability, reliability, security, และ monitoring! 🚀