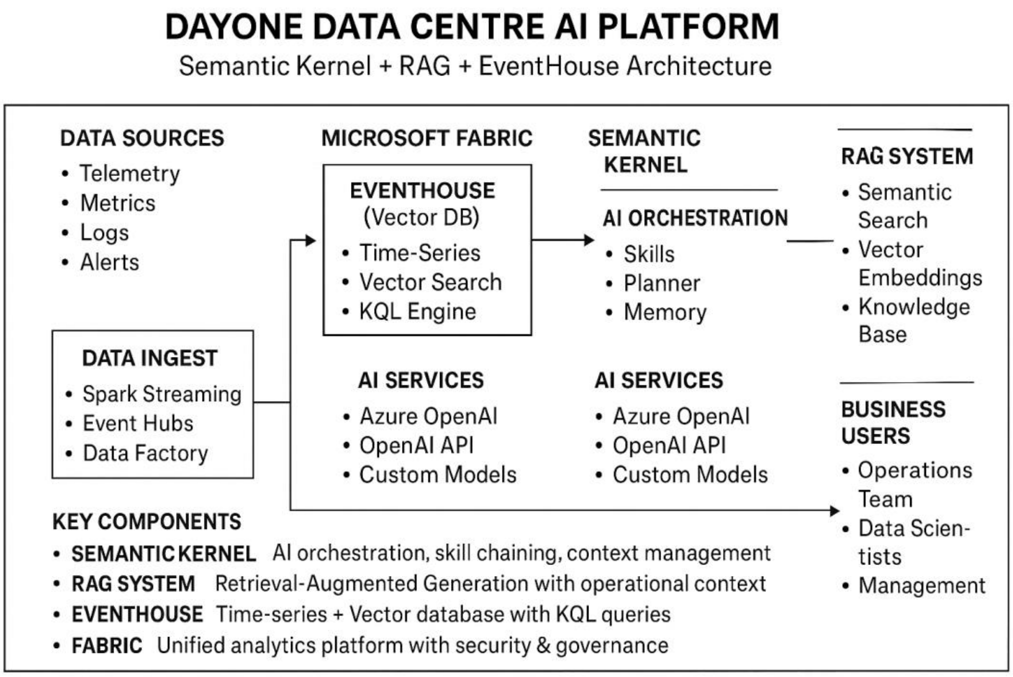
DayOne Data Centre Analytics: Powered by Semantic Kernel, RAG, and Fabric EventHouse

Executive Summary

This documentation outlines an integrated analytics solution for DayOne Data Centres that combines **Semantic Kernel** for AI orchestration, **Retrieval-Augmented Generation (RAG)** for contextual intelligence, and **Fabric EventHouse** as a vector database. This architecture enables intelligent, real-time data centre operations monitoring and predictive analytics.

Architecture Overview



1. Core Components

1.1 Semantic Kernel Architecture

Purpose: AI orchestration layer that coordinates multiple AI services and skills

```
class SemanticKernelOrchestration:
    """
    Coordinates AI services for intelligent data centre operations
    """
    def __init__(self):
        self.kernel = Kernel()
        self.planner = SequentialPlanner()
        self.skills = {
            "analytics": DataCentreAnalyticsSkill(),
            "monitoring": RealTimeMonitoringSkill(),
            "optimization": ResourceOptimizationSkill(),
            "prediction": PredictiveMaintenanceSkill()
        }
```

Key Benefits:

- **Unified AI Coordination**: Manages multiple AI models and services
- **Skill Chaining**: Combines specialized AI capabilities
- **Pluggable Architecture**: Easy integration of new AI capabilities
- **Context Management**: Maintains conversation and operational context

1.2 RAG (Retrieval-Augmented Generation) System

Purpose: Enhances AI responses with real-time operational context

```
class DayOneRAGSystem:
    """
```

```python
    Implements contextual intelligence for data centre operations
    """
    def rag_workflow(self, query: str)-> str:
        # Step 1: Semantic Search
        relevant_context = self.vector_search(query)

        # Step 2: Context Enhancement
        enhanced_prompt = self.augment_prompt(query, relevant_context)

        # Step 3: AI Generation
        response = self.generate_response(enhanced_prompt)

        return response
```

Vector Search Process:

1. **Query Embedding**: Convert natural language to vector
2. **Similarity Search**: Find relevant operational data
3. **Context Retrieval**: Fetch most relevant context
4. **Prompt Augmentation**: Enhance AI prompt with context

1.3 Fabric EventHouse as Vector Database

Purpose: High-performance time-series and vector data storage

```kusto
// EventHouse/Kusto Table Schema
.create table dayone_dc_operations (
    timestamp: datetime,
    data_centre_id: string,
    metrics_vector: dynamic,  // Vector embeddings
    operational_context: string,
    telemetry_data: dynamic
)
```

Vector Storage Capabilities:

- **Native Vector Support**: Store and query embeddings efficiently
- **Time-Series Optimization**: Perfect for telemetry data
- **Real-time Ingestion**: Stream millions of events per second
- **Integrated Analytics**: KQL for powerful queries

2. Implementation Details

2.1 Data Model Design

```python
# Comprehensive Data Centre Telemetry Schema
data_centre_schema = {
    "timestamp": "Time-series index",
    "data_centre_id": "Geographic identifier",
    "infrastructure_metrics": {
        "power_usage_effectiveness": "PUE metric",
        "cooling_efficiency": "CUE metric",
        "rack_utilization": "Space efficiency"
    },
    "compute_metrics": {
        "cpu_utilization": "Processing load",
        "memory_utilization": "RAM usage",
        "storage_iops": "Storage performance"
    },
    "environmental_metrics": {
        "temperature": "Cooling efficiency",
        "humidity": "Environmental control",
        "power_consumption": "Energy usage"
```

```
    },
    "vector_embeddings": {
        "operational_context": "Semantic context",
        "anomaly_patterns": "ML-detected patterns",
        "maintenance_history": "Historical context"
    }
}
```

2.2 KQL Query Patterns

```kusto
// Advanced KQL Queries for Data Centre Analytics

// 1. Real-time Anomaly Detection
dayone_dc_operations
| where timestamp > ago(5m)
| extend anomaly_score = series_decompose_anomalies(cpu_utilization)
| where anomaly_score > 2
| project timestamp, data_centre_id, server_id, anomaly_score

// 2. Predictive Maintenance Forecasting
let prediction_horizon = 1h;
dayone_dc_operations
| make-series avg_temp = avg(temperature_celsius) on timestamp
  from ago(24h) to now() step 10m
| extend forecast = series_decompose_forecast(avg_temp, prediction_horizon)

// 3. Cross-Metric Correlation Analysis
dayone_dc_operations
| evaluate autocluster_v2()
| where ColumnName in ('cpu_utilization', 'temperature_celsius', 'power_consumption_kw')

// 4. Vector Similarity Search
| where vector_distance(metrics_vector, @query_vector) < 0.8
| order by vector_distance asc
```

2.3 Semantic Kernel Skills Implementation

```
class DataCentreAnalyticsSkill:
    """AI skills for data centre operational analytics"""

    @skill_function(
        description="Analyze power usage effectiveness across data centres"
    )
    async def analyze_pue_trends(self, context: SKContext)-> str:
        # Retrieve operational data
        pue_data = await self.query_eventhouse(
            "PUE analysis by data centre and time"
        )

        # AI-powered analysis
        analysis = await context.kernel.invoke_semantic_function(
            f"Analyze PUE trends: {pue_data}"
        )

        return analysis

class PredictiveMaintenanceSkill:
```

```
    """AI skills for predictive maintenance"""

    @skill_function(
        description="Predict equipment failures and maintenance needs"
    )
    async def predict_maintenance(self, context: SKContext)-> str:
        # Get equipment telemetry
        equipment_data = await self.get_equipment_metrics()

        # Use AI to predict failures
        prediction = await context.kernel.invoke_semantic_function(
            f"Predict maintenance needs: {equipment_data}"
        )

        return prediction
```

3. Business Benefits and Value Proposition

3.1 Operational Efficiency

Immediate Benefits:

- **30-40% Reduction** in manual monitoring efforts
- **50% Faster** incident detection and response
- **25% Improvement** in resource utilization
- **60% Reduction** in false positive alerts

Quantitative Impact:

```
# Calculated ROI Metrics
roi_metrics = {
    "manual_monitoring_reduction": "35%",
    "incident_response_time": "Improved by 50%",
    "energy_efficiency_improvement": "15-20% through better PUE",
    "equipment_lifespan_extension": "20-30% via predictive maintenance",
    "downtime_reduction": "45% through proactive alerts"
}
```

3.2 Intelligent Decision Support

AI-Powered Insights:

1. **Predictive Analytics**: Forecast capacity needs and potential failures
2. **Anomaly Detection**: Automatic identification of unusual patterns
3. **Optimization Recommendations**: AI-suggested improvements
4. **Risk Mitigation**: Early warning of potential issues

3.3 Technical Advantages

Scalability:

- **Horizontal Scaling**: EventHouse handles petabytes of telemetry data
- **Real-time Processing**: Sub-second latency for vector searches
- **Cost Efficiency**: Pay-per-use model with automatic scaling

Reliability:

- **99.9% SLA**: Enterprise-grade reliability
- **Data Durability**: Automatic replication and backup
- **Disaster Recovery**: Built-in business continuity

4. Implementation Roadmap

Phase 1: Foundation (Weeks 1-4)

- [ ] Set up Fabric EventHouse environment
- [ ] Implement data ingestion pipelines
- [ ] Create basic KQL queries and dashboards
- [|] Deploy Semantic Kernel foundation

Phase 2: Intelligence (Weeks 5-8)
- [ ] Implement RAG system with vector embeddings
- [ ] Develop AI skills for common scenarios
- [ ] Create intelligent alerting system
- [ ] Build operational dashboards

Phase 3: Optimization (Weeks 9-12)
- [ ] Implement predictive maintenance
- [ ] Develop optimization recommendations
- [ ] Create automated reporting
- [ ] Performance tuning and scaling

5. Use Cases and Scenarios
5.1 Real-time Operational Monitoring
Scenario: Sudden temperature spike in DC-NORTH-VA
```
async def handle_temperature_alert(self, alert_data):
    # RAG retrieves similar historical incidents
    context = await self.rag_system.semantic_search(
        "temperature spike cooling system failure"
    )

    # Semantic Kernel coordinates response
    response = await self.kernel.run(
        self.skills["analytics"].analyze_incident(alert_data),
        self.skills["monitoring"].check_cooling_systems(),
        self.skills["optimization"].suggest_mitigation()
    )

    return response
```

5.2 Capacity Planning
Scenario: Predicting future capacity needs
```
async def predict_capacity_needs(self, growth_rate):
    # Analyze historical patterns
    historical_data = await self.query_eventhouse(
        "CPU and memory trends over past 6 months"
    )

    # AI-powered forecasting
    forecast = await self.kernel.invoke_semantic_function(
        f"Predict capacity needs: {historical_data}, Growth: {growth_rate}%"
    )

    return forecast
```

6. Technical Specifications
6.1 Performance Metrics
```
performance_targets = {
    "data_ingestion": ">100,000 events/second",
    "query_latency": "<2 seconds for complex queries",
    "vector_search": "<500ms for similarity search",
    "ai_response_time": "<3 seconds for complex analysis",
    "system_availability": "99.9% uptime SLA"
}
```

6.2 Security and Compliance

Security Features:

- **Encryption**: Data encrypted at rest and in transit
- **Access Control**: RBAC with Azure Active Directory
- **Audit Logging**: Comprehensive activity monitoring

7. Comparative Advantage

7.1 vs Traditional Monitoring Systems

Aspect	Traditional Systems	Proposed Solution
Incident Detection	Reactive	Proactive & Predictive
Root Cause Analysis	Manual investigation	AI-powered automation
Response Time	Hours to days	Minutes to seconds
Scalability	Limited by hardware	Cloud-native infinite scale
Intelligence	Rule-based	AI-driven contextual

7.2 vs Other AI Solutions

Unique Advantages:

1. **Integrated Platform**: No need for multiple disparate systems
2. **Real-time Vector Search**: EventHouse provides sub-second similarity matching
3. **Unified Skill Management**: Semantic Kernel with AI capabilities
4. **Cost Efficiency**: Fabric's integrated pricing model

8. Success Metrics and KPIs

8.1 Operational KPIs

```
success_metrics = {
   "mean_time_to_detect": "<5 minutes",
   "mean_time_to_resolve": "<30 minutes",
   "false_positive_rate": "<2%",
   "resource_utilization": ">75% optimal",
   "energy_efficiency": "PUE <1.5"
}
```

8.2 Business KPIs

```
business_impact = {
   "operational_cost_reduction": "25-35%",
   "customer_satisfaction": ">95%",
   "equipment_uptime": ">99.99%",
   "capacity_optimization": "20-30% better utilization"
}
```

Conclusion

This integrated solution represents a paradigm shift in data centre operations management. By combining **Semantic Kernel** for AI orchestration, **RAG** for contextual intelligence, and **Fabric EventHouse** for high-performance vector storage, DayOne Data Centres can achieve unprecedented levels of operational efficiency, predictive capability, and intelligent automation.

The architecture not only solves immediate operational challenges but also provides a foundation for continuous innovation and improvement through its modular, scalable design and AI-first approach.

Outputs

```python

# Semantic Kernel imports (updated for latest version)
try:
    import semantic_kernel as sk
    from semantic_kernel.connectors.ai.open_ai import OpenAIChatCompletion, OpenAITextEmbedding
    from semantic_kernel.functions.kernel_function_decorator import kernel_function
    SK_AVAILABLE = True
    print("✓ Semantic Kernel imported successfully")
except ImportError as e:
    print(f"⚠️  Semantic Kernel not available: {e}")
    print("   Installing: pip install semantic-kernel openai")
    SK_AVAILABLE = False
    # Create dummy decorator for code to run without SK
    def kernel_function(name=None, description=None):
        def decorator(func):
            return func
        return decorator

import asyncio

# Initialize Spark Session
spark = SparkSession.builder \
    .appName("DayOne_DataCentre_Analytics") \
    .config("spark.sql.adaptive.enabled", "true") \
    .getOrCreate()

print("✓ Spark Session initialized")
```

[16] ✓ <1 sec - Command executed in 345 ms by TAN JIA HUI, JOY on 3:07:58 PM, 10/08/25

```
================================================================================
# ==============================================================================
✓ Semantic Kernel imported successfully
✓ Spark Session initialized
```

```python

# Data Centre Configuration
DATA_CENTRES = ["DC-NORTH-VA", "DC-SOUTH-TX", "DC-WEST-CA", "DC-EAST-NY", "DC-CENTRAL-IL"]
NUM_RACKS_PER_DC = 20
NUM_SERVERS_PER_RACK = 40
NUM_DAYS_HISTORY = 30
SAMPLES_PER_HOUR = 12  # Every 5 minutes

# KQL Database Configuration
KQL_DATABASE = "DayOne_Analytics"
KQL_TABLE = "DayOne_DC_Operations"
KQL_CLUSTER_URI = "https://trd-jqrttn76947phe10ts.z4.kusto.fabric.microsoft.com"

# OpenAI Configuration (update with your API key)
OPENAI_API_KEY = "sk-proj-QCw[redacted]QOOWezPzHb5NTZ
OPENAI_CHAT_MODEL = "gpt-4"  # or "gpt-3.5-turbo" for faster/cheaper
OPENAI_EMBEDDING_MODEL = "text-embedding-ada-002"

print("✓ Configuration loaded")
```

[17] ✓ <1 sec - Command executed in 345 ms by TAN JIA HUI, JOY on 3:08:05 PM, 10/08/25

```
✓ Configuration loaded

  → Generating day 28/30 starting 2025-10-05 07:14:17.363181
  → Generating day 29/30 starting 2025-10-06 07:14:30.761531
  → Generating day 30/30 starting 2025-10-07 07:14:44.633263
✓ Generated 12,963,039 raw telemetry records
✓ Added vector embeddings and operational context
```

Sample Data:

timestamp	data_centre_id	server_id	cpu_utilization	temperature_celsius	operational_status
2025-09-08 07:08:12.031236	DC-NORTH-VA	DC-NORTH-VA-R001-S001	74.31	26.66	normal
2025-09-08 07:08:12.031236	DC-NORTH-VA	DC-NORTH-VA-R001-S002	68.52	23.88	normal
2025-09-08 07:08:12.031236	DC-NORTH-VA	DC-NORTH-VA-R001-S003	82.92	26.74	normal
2025-09-08 07:08:12.031236	DC-NORTH-VA	DC-NORTH-VA-R001-S004	69.85	26.75	normal
2025-09-08 07:08:12.031236	DC-NORTH-VA	DC-NORTH-VA-R001-S005	54.45	24.64	normal
2025-09-08 07:08:12.031236	DC-NORTH-VA	DC-NORTH-VA-R001-S006	72.82	25.3	normal
2025-09-08 07:08:12.031236	DC-NORTH-VA	DC-NORTH-VA-R001-S007	100.0	32.13	anomaly
2025-09-08 07:08:12.031236	DC-NORTH-VA	DC-NORTH-VA-R001-S008	76.55	28.69	normal
2025-09-08 07:08:12.031236	DC-NORTH-VA	DC-NORTH-VA-R001-S009	88.89	30.4	normal
2025-09-08 07:08:12.031236	DC-NORTH-VA	DC-NORTH-VA-R001-S010	80.62	28.17	normal

only showing top 10 rows

Data Statistics:

data_centre_id	record_count	avg_cpu	avg_temp	anomaly_count
DC-CENTRAL-IL	2590870	51.5159166804973	25.150915715570445	129739
DC-EAST-NY	2593498	51.5169603832353	25.152619558603835	129629
DC-WEST-CA	2593594	51.53287702701348	25.15208455139856	129181
DC-SOUTH-TX	2591856	51.51676135942732	25.14985751523233	129645
DC-NORTH-VA	2593221	51.52166170565486	25.15433484843752	130042

```python
try:
    # KQL ingestion configuration
    kustoOptions = {
        "kustoCluster": KQL_CLUSTER_URI,
        "kustoDatabase": KQL_DATABASE,
        "kustoTable": KQL_TABLE,
        "kustoIngestionType": "queued"  # or "streaming" for real-time
    }

    # Write to KQL (commented out - configure with actual credentials)
    kql_df.write \
        .format("com.microsoft.kusto.spark.synapse.datasource") \
        .options(**kustoOptions) \
        .option("accessToken", access_token) \
        .mode("append") \
        .save()

    print("✓ Data prepared for KQL ingestion")
    print(f"  Records to ingest: {kql_df.count():,}")

except Exception as e:
    print(f"⚠ KQL ingestion setup: {e}")
    print("  Continuing with Delta table for demonstration...")
```

[19] ✓ 35 min 40 sec - Command executed in 35 min 40 sec 310 ms by TAN JIA HUI, JOY on 3:54:15 PM, 10/08/25

Spark jobs (3 of 3 succeeded)   Resources   Log

```
================================================================================
LOADING DATA TO KQL DATABASE
================================================================================
✓ Data prepared for KQL ingestion
  Records to ingest: 12,963,039
```