# Space Debris Risk Prediction - PPT Outline

## Real-Time Big Data Pipeline Project

---

## Slide 1: Title Slide

**Space Debris Risk Prediction Using Big Data Analytics**

- Real-Time Streaming Pipeline for Orbital Position Tracking
- January 2026

---

## Slide 2: Problem Statement & Objectives

### The Space Debris Challenge

- **Over 36,000 tracked objects** orbiting Earth
- Growing collision risk threatens satellite infrastructure
- Traditional batch processing insufficient for real-time monitoring
- Space tracking APIs update infrequently (days/months between updates)

### Project Objectives

- **Build end-to-end real-time data pipeline** for space debris tracking
- **Process TLE data at scale** using Big Data technologies
- **Compute orbital positions** using SGP4 propagation algorithm
- **Store processed data** in distributed file system
- **Automate pipeline workflows** with orchestration tools

---

## Slide 3: System Architecture

### Data Flow Pipeline

```
TLE Data → Streaming API → Kafka → Spark Processing → HDFS Storage
```

### Pipeline Components

1. **Data Source**: Historical TLE (Two-Line Element) satellite data
2. **Streaming API**: Accelerated playback simulation (100x speed)
3. **Apache Kafka**: Distributed message streaming
4. **Apache Spark**: Real-time SGP4 computation
5. **HDFS**: Distributed storage for processed data
6. **Apache Airflow**: Workflow orchestration and automation

### Architecture Principles

- Distributed processing for scalability
- Fault-tolerant with data replication
- Real-time streaming with low latency
- Modular components for maintainability

---

## Slide 4: Technology Stack

### Core Big Data Technologies

**Apache Kafka**

- Distributed streaming platform
- High-throughput message broker
- Handles real-time data ingestion
- Fault-tolerant pub-sub architecture

**Apache Spark**

- Distributed data processing engine
- Structured streaming for real-time analytics
- In-memory computation for performance
- Supports complex transformations

**HDFS (Hadoop Distributed File System)**

- Scalable distributed storage
- Fault-tolerant with 3x replication
- Optimized for large files
- Cost-effective long-term storage

**Apache Airflow**

- Workflow orchestration platform
- Automated task scheduling
- Dependency management
- Monitoring and logging

---

## Slide 5: Data Source - TLE Format

### Two-Line Element (TLE) Sets

**NORAD Standard Format for Orbital Parameters**

### Key Data Elements

- **Satellite ID**: NORAD catalog number
- **Epoch**: Timestamp of measurement

- **Inclination**: Orbit angle relative to equator
- **RAAN**: Right Ascension of Ascending Node
- **Eccentricity**: Orbit shape (circular vs elliptical)
- **Mean Motion**: Number of orbits per day
- **Argument of Perigee**: Orientation parameter
- **Mean Anomaly**: Position in orbit

### Dataset Characteristics

- 1000+ satellites tracked
- Historical data spanning months
- Multiple TLE updates per satellite over time

---

## Slide 6: Component 1 - Streaming API

### Purpose & Implementation

- Simulates real-time TLE data stream from historical files
- Chronologically ordered playback by epoch timestamp
- Accelerated playback (100x-1000x real-time)
- REST endpoint for continuous data consumption

### Why Simulation?

- Real space APIs update slowly (days to months intervals)
- Enables continuous stream for testing and development
- Reproducible testing scenarios
- Demonstrates real-time processing capabilities

---

## Slide 7: Component 2 - Kafka Producer

### Data Ingestion & Enrichment

**Responsibilities**

- Consumes TLE stream from API endpoint
- Parses TLE format (3-line sets per satellite)
- Extracts and validates orbital elements
- Enriches with metadata and timestamps
- Publishes to Kafka topic

**Data Processing**

- Parse inclination, RAAN, eccentricity from TLE lines
- Extract mean motion and mean anomaly
- Add processing timestamp for tracking
- Validate data quality and completeness

**Message Delivery**

- Partitions by satellite_id for parallel processing
- JSON serialization for interoperability
- Guaranteed delivery with error handling

---

# Slide 8: Component 3 - Kafka Topic

## Topic Configuration: space_debris_tle

**Design Strategy**

- Multiple partitions for parallel processing
- Replication factor for fault tolerance
- Retention policy for historical availability
- Compression for storage efficiency

**Message Schema**

- Satellite identification (ID, name, NORAD number)
- TLE lines (line1, line2)
- Epoch and timestamp
- Parsed orbital elements
- Processing metadata

**Partitioning Benefits**

- Key by satellite_id ensures order per satellite
- Enables parallel consumer processing
- Load balancing across partitions

---

# Slide 9: Component 4 - Spark Streaming

## Real-Time Processing Engine

**Stream Consumption**

- Continuous reading from Kafka topics
- Micro-batch processing with low latency
- Automatic offset management
- Fault-tolerant checkpoint recovery

**Processing Steps**

1. Deserialize JSON messages from Kafka
2. Validate and clean TLE data
3. Apply SGP4 propagation algorithm
4. Compute 3D position vectors (x, y, z)
5. Calculate velocity vectors (vx, vy, vz)

6. Derive altitude and velocity magnitude
7. Prepare data for storage

**Distributed Computation**

- Parallel processing across cluster nodes
- In-memory operations for speed
- Automatic load balancing

---

# Slide 10: SGP4 Orbit Propagation

## Simplified General Perturbations Model

**What is SGP4?**

- Standard algorithm for satellite orbit propagation
- Developed by NORAD for space object tracking
- Industry standard for TLE-based orbit prediction
- Accounts for gravitational and atmospheric effects

**Computation Flow**

- **Input**: TLE parameters (orbital elements) + epoch time
- **Processing**: Mathematical propagation of orbital mechanics
- **Output**: 3D position and velocity vectors

**Computed Metrics**

- Position coordinates (x, y, z) in kilometers
- Velocity components (vx, vy, vz) in km/s
- Altitude above Earth's surface
- Orbital speed magnitude
- Distance from Earth's center

---

# Slide 11: Component 5 - HDFS Storage

## Distributed Data Persistence

**HDFS Architecture**

- NameNode manages metadata
- DataNodes store actual data blocks
- Default 3x replication for fault tolerance
- Block-based storage optimized for large files

**Storage Implementation**

- **Format**: Parquet (columnar, compressed)
- **Partitioning**: Date-based (year/month/day)
- **Schema**: Position, velocity, orbital parameters

- **Query optimization**: Columnar structure

**Data Organization**

```
/space_debris/sgp4_computed/
  year=2024/month=01/day=17/
```

**Benefits**

- Scalable to petabyte level
- Fault-tolerant with replication
- Cost-effective storage
- Efficient for analytical queries

---

# Slide 12: Component 6 - Airflow Orchestration

## Automated Workflow Management

**DAG (Directed Acyclic Graph) Structure**

- Programmatic workflow definition
- Task dependency management
- Scheduled execution (hourly)
- Manual trigger support

**Pipeline Tasks**

1. Verify API availability
2. Launch Kafka producer
3. Stream configured number of records
4. Monitor Spark processing status
5. Verify HDFS data writes
6. Generate processing logs

**Automation Benefits**

- End-to-end pipeline automation
- Error handling and retry logic
- Comprehensive monitoring
- Historical run tracking

---

# Slide 13: Current Implementation Status

## Completed Components ✓

**Data Pipeline (Fully Operational)**

- ✓ TLE streaming API with historical data

- ✓ Kafka producer with data enrichment
- ✓ Kafka topic configuration and messaging
- ✓ Spark streaming consumer
- ✓ SGP4 orbit propagation computation
- ✓ Position and velocity vector calculation
- ✓ HDFS storage with Parquet format
- ✓ Airflow DAG for automation

**System Capabilities**

- Process 100-1000 TLE records per second
- End-to-end latency under 5 seconds
- Handles 1000+ satellites
- Distributed processing and storage

### Future Work (Not Yet Implemented)

- Risk analysis and classification algorithm
- Collision detection system
- Real-time alerting mechanism
- Visualization dashboard

---

## Slide 14: Challenges & Solutions

### Technical Challenges Addressed

**Challenge 1: API Update Frequency**

- Problem: Real space APIs update infrequently
- Solution: Built streaming API with accelerated historical playback
- Result: Continuous data flow for testing and demonstration

**Challenge 2: Computational Complexity**

- Problem: SGP4 calculations are CPU-intensive
- Solution: Distributed processing with Spark cluster
- Result: Parallel computation across multiple nodes

**Challenge 3: Data Order Maintenance**

- Problem: Distributed systems can lose chronological order
- Solution: Kafka partitioning by satellite_id
- Result: Per-satellite ordering maintained

**Challenge 4: Storage Efficiency**

- Problem: Large volumes of time-series data
- Solution: Parquet columnar format with compression
- Result: 70% storage reduction

---

# Slide 15: Conclusion & Next Steps

## Project Achievements

- ✓ Operational end-to-end real-time big data pipeline
- ✓ Successfully integrated Kafka, Spark, HDFS, and Airflow
- ✓ Scalable architecture for space debris tracking
- ✓ SGP4 orbital position computation at scale
- ✓ Efficient distributed storage implementation

## Technical Impact

- Demonstrated real-time big data processing in space domain
- Applied industry-standard streaming architecture
- Built foundation for collision risk analysis
- Proved scalability with distributed technologies

## Future Development

- Implement risk classification algorithm
- Add collision detection capabilities
- Build monitoring dashboard
- Integration with live space tracking APIs

**Thank You**