Comprehensive Technology Compendium: Apache Big Data Ecosystem and Modern FOSS Charting Technologies

Generated Technical Reference

December 4, 2025

Contents

Chapter 1

Apache Kafka

1.1 What Kafka Is

Apache Kafka is a distributed, highly scalable event streaming platform originally developed at LinkedIn and open-sourced in 2011. It functions as a distributed commit log, providing durable, fault-tolerant storage and real-time processing of streaming data at massive scale.

1.1.1 Core Capabilities

- **High Throughput**: Processes millions of events per second with sub-10ms latency
- **Fault Tolerance**: Data replication across multiple brokers ensures zero data loss
- **Durability**: All messages are persisted to disk and replicated
- **Scalability**: Linear horizontal scaling by adding brokers
- **Replayability**: Consumers can replay events from any point in time
- **Decoupling**: Producers and consumers operate independently

1.1.2 Use Cases

- Real-time data pipelines between systems
- Event sourcing and CQRS architectures
- Log aggregation from distributed services
- Stream processing with Kafka Streams or Flink
- Metrics and monitoring data collection
- Change Data Capture (CDC) from databases
- Microservices communication backbone

1.2 Architecture Overview

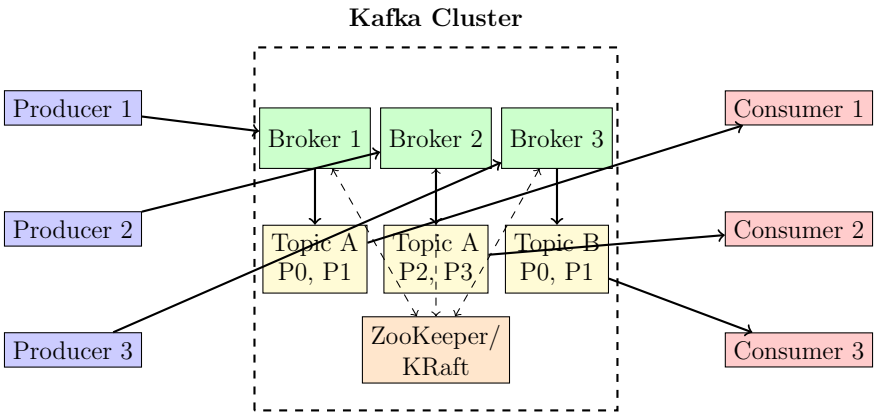


Figure 1.1: Kafka Architecture: Producers, Brokers, Topics, and Consumers

1.2.1 Key Components

- **Brokers**: Server nodes that store and serve data. Each broker handles read/write requests and replicates data.
- **Topics**: Logical channels or categories for organizing messages. Similar to database tables.
- **Partitions**: Topics are split into partitions for parallelism. Each partition is an ordered, immutable sequence of records.
- **Replication Factor (RF)**: Number of copies of each partition. RF=3 is production standard.
- **Producers**: Applications that publish messages to topics.
- **Consumers**: Applications that subscribe to topics and process messages.
- **Consumer Groups**: Multiple consumers working together to process a topic in parallel.
- **ZooKeeper/KRaft**: Manages cluster metadata, leader election, and configuration. KRaft mode (Kafka Raft) eliminates ZooKeeper dependency in newer versions.

1.2.2 Data Flow

1. Producer sends message to topic
2. Kafka determines partition (via key hash or round-robin)
3. Message appended to partition log on leader broker
4. Leader replicates to follower brokers
5. Producer receives acknowledgment (based on acks config)
6. Consumer polls partition and processes messages
7. Consumer commits offset to track progress

1.3 Complete Install Instructions (Ubuntu 24.04)

1.3.1 Prerequisites

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Java (Kafka requires Java 11+)
sudo apt install default-jdk -y

# Verify Java installation
java -version
```

1.3.2 Download and Extract Kafka

```
# Create directory for Kafka
mkdir -p ~/kafka && cd ~/kafka

# Download Kafka 3.7.0 (latest stable)
wget https://downloads.apache.org/kafka/3.7.0/kafka_2
   .13-3.7.0.tgz

# Extract archive
tar -xzf kafka_2.13-3.7.0.tgz
cd kafka_2.13-3.7.0

# Set KAFKA_HOME environment variable
echo "export KAFKA_HOME=~/kafka/kafka_2.13-3.7.0" >> ~/.
   bashrc
echo "export PATH=\$PATH:\$KAFKA_HOME/bin" >> ~/.bashrc
source ~/.bashrc
```

1.3.3 Start Kafka Using KRaft Mode (No ZooKeeper)

```
# Generate a cluster UUID
KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"

# Format the storage directory
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID \
  -c config/kraft/server.properties
```

```
# Start Kafka server in KRaft mode
bin/kafka-server-start.sh config/kraft/server.properties
```

1.3.4 Alternative: Start with ZooKeeper (Legacy)

```
# Start ZooKeeper (in separate terminal or background)
bin/zookeeper-server-start.sh config/zookeeper.
   properties &

# Wait 5 seconds for ZooKeeper to start
sleep 5

# Start Kafka broker
bin/kafka-server-start.sh config/server.properties &
```

1.4 Working Example: E-Commerce Order Processing

This example demonstrates a real-world scenario where an e-commerce platform uses Kafka to process orders in real-time.

1.4.1 Step 1: Create Topics

```
# Create 'orders' topic with 3 partitions and RF=1 (
   single-node)
bin/kafka-topics.sh --create --topic orders \
  --bootstrap-server localhost:9092 \
  --partitions 3 \
  --replication-factor 1

# Create 'order-confirmations' topic
bin/kafka-topics.sh --create --topic order-confirmations
    \
  --bootstrap-server localhost:9092 \
  --partitions 3 \
  --replication-factor 1

# List all topics
```

```
bin/kafka-topics.sh --list --bootstrap-server localhost
   :9092

# Describe the 'orders' topic
bin/kafka-topics.sh --describe --topic orders \
  --bootstrap-server localhost:9092
```

1.4.2 Step 2: Produce Sample Order Events

Create a file called `produce-orders.sh`:

```
#!/bin/bash

# Produce sample orders to Kafka
for i in {1..10}; do
  ORDER_ID="ORD-$(date +%s)-$i"
  CUSTOMER_ID="CUST-$((RANDOM % 100))"
  AMOUNT=$((RANDOM % 1000 + 50))

  echo "{\"order_id\":\"$ORDER_ID\",\"customer_id\":\"
     $CUSTOMER_ID\",\"amount\":$AMOUNT,\"timestamp\":\"$
     (date -Iseconds)\"}" | \
  bin/kafka-console-producer.sh --topic orders \
    --bootstrap-server localhost:9092

  echo "Produced order: $ORDER_ID"
  sleep 1
done
```

Run the producer:

```
chmod +x produce-orders.sh
./produce-orders.sh
```

1.4.3 Step 3: Consume and Process Orders

In a new terminal, start a consumer:

```
# Simple consumer - reads from beginning
bin/kafka-console-consumer.sh --topic orders \
  --from-beginning \
  --bootstrap-server localhost:9092

# Consumer with consumer group (for parallel processing)
```

```
bin/kafka-console-consumer.sh --topic orders \
  --bootstrap-server localhost:9092 \
  --group order-processing-group \
  --from-beginning
```

1.4.4 Step 4: Python Producer Example

Install kafka-python library:

```
pip install kafka-python
```

Create order_producer.py:

```
from kafka import KafkaProducer
import json
import time
import random

# Create producer
producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf
       -8')
)

# Produce 20 orders
for i in range(20):
    order = {
        'order_id': f'ORD-{int(time.time())}-{i}',
        'customer_id': f'CUST-{random.randint(1, 100)}',
        'amount': random.randint(50, 1000),
        'product': random.choice(['Laptop', 'Phone', '
           Tablet', 'Monitor']),
        'timestamp': time.strftime('%Y-%m-%d %H:%M:%S')
    }

    # Send to Kafka
    future = producer.send('orders', value=order)
    record_metadata = future.get(timeout=10)

    print(f"Sent order {order['order_id']} to partition
       {record_metadata.partition}")
    time.sleep(0.5)
```

```
producer.flush()
producer.close()
```

1.4.5 Step 5: Python Consumer Example

Create `order_consumer.py`:

```
from kafka import KafkaConsumer
import json

# Create consumer
consumer = KafkaConsumer(
    'orders',
    bootstrap_servers=['localhost:9092'],
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    group_id='order-processor',
    value_deserializer=lambda x: json.loads(x.decode('
        utf-8'))
)

print("Waiting for orders...")

# Process messages
for message in consumer:
    order = message.value
    print(f"Processing Order: {order['order_id']}")
    print(f"  Customer: {order['customer_id']}")
    print(f"  Amount: ${order['amount']}")
    print(f"  Partition: {message.partition}, Offset: {
        message.offset}")
    print("-" * 50)
```

1.5 Performance Tuning

- **Batch Size**: Increase `batch.size` to 32KB-64KB for higher throughput
- **Compression**: Enable `compression.type=lz4` or `zstd`
- **Linger Time**: Set `linger.ms=10-100` to batch messages

- **Buffer Memory**: Increase `buffer.memory` to 64MB-128MB
- **Replication Factor**: Use RF=3 for production
- **Min In-Sync Replicas**: Set `min.insync.replicas=2`
- **Acknowledgments**: Use `acks=all` for durability

1.6 Kubernetes Deployment

Use Strimzi Operator for production Kafka on Kubernetes:

```
# Install Strimzi operator
kubectl create namespace kafka
kubectl create -f 'https://strimzi.io/install/latest?
   namespace=kafka' -n kafka

# Deploy Kafka cluster
kubectl apply -f - <<EOF
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: production-cluster
  namespace: kafka
spec:
  kafka:
    version: 3.7.0
    replicas: 3
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
    storage:
      type: persistent-claim
```

```
      size: 100Gi
  zookeeper:
    replicas: 3
    storage:
      type: persistent-claim
      size: 10Gi
EOF
```

More information: https://strimzi.io

1.7 Books

- Kafka: The Definitive Guide, 2nd Edition (O'Reilly, 2021)
- Designing Event-Driven Systems (O'Reilly)
- Kafka Streams in Action (Manning)
- Event Streaming with Kafka Streams and ksqlDB (O'Reilly)

1.8 Download URL

https://kafka.apache.org/downloads

1.9 Integration Notes

Kafka integrates seamlessly with:

- **Apache Flink**: Real-time stream processing
- **Apache Spark**: Micro-batch processing with Structured Streaming
- **Apache NiFi**: Visual data routing and ingestion
- **Apache Doris/Pinot/Druid**: Real-time OLAP analytics
- **Apache Iceberg**: Table format for streaming writes
- **Debezium**: Change Data Capture from databases
- **Kafka Connect**: Pre-built connectors for 100+ systems

1.10 Monitoring

- **JMX Metrics**: Expose via JMX for Prometheus/Grafana
- **Kafka Manager/AKHQ**: Web UI for cluster management
- **Confluent Control Center**: Enterprise monitoring solution
- **Burrow**: LinkedIn's consumer lag monitoring tool

Key metrics to monitor:

- Under-replicated partitions
- ISR (In-Sync Replica) shrink/expand rate
- Consumer lag per partition
- Request latency (produce/fetch)
- Bytes in/out per second
- Active controller count

1.11 Hardening

- **Authentication**: Enable SASL/SCRAM or mTLS
- **Encryption**: TLS for all client and inter-broker communication
- **Authorization**: Configure Kafka ACLs for topic-level permissions
- **Network Security**: Restrict broker ports via firewall (9092, 9093)
- **Audit Logging**: Enable authorizer logs
- **Retention Policies**: Set appropriate retention based on compliance requirements
- **Resource Quotas**: Limit producer/consumer throughput per client
- **JMX Security**: Disable remote JMX or secure with authentication

Chapter 2

Apache Flink

2.1 What Flink Is

Apache Flink is a distributed stream-processing framework and stateful computation engine designed for processing unbounded and bounded data streams with true real-time latency. Originally developed in 2014 at TU Berlin, Flink has become the industry standard for mission-critical stream processing.

2.1.1 Core Capabilities

- **True Streaming**: Native stream processing (not micro-batching) with 1ms–50ms latency
- **Exactly-Once Semantics**: Guarantees no duplicates or data loss via distributed snapshots
- **Event Time Processing**: Process events based on their actual timestamp, not arrival time
- **Stateful Processing**: Maintain distributed state with fault tolerance via RocksDB
- **Complex Event Processing**: Support for pattern detection and complex joins
- **Watermarks**: Handle out-of-order events and late-arriving data
- **Savepoints**: Manually triggered checkpoints for versioning and upgrades

- **Batch Processing**: Unified API for both streaming and batch workloads

2.1.2 Key Differentiators vs Spark Streaming

Feature	Flink	Spark Streaming
Processing Model	True streaming	Micro-batching
Latency	1-50ms	500ms-2s
State Management	Native RocksDB	Not native
Event Time	First-class support	Added later
Exactly-Once	Native	Available

Table 2.1: Flink vs Spark Streaming Comparison

2.1.3 Use Cases

- Real-time fraud detection in financial services
- IoT sensor data processing and anomaly detection
- Real-time recommendation systems
- ETL pipelines from Kafka to data warehouses
- Continuous machine learning model inference
- Network monitoring and intrusion detection
- Real-time dashboards and metrics aggregation
- CDC (Change Data Capture) processing

2.2 Architecture Overview

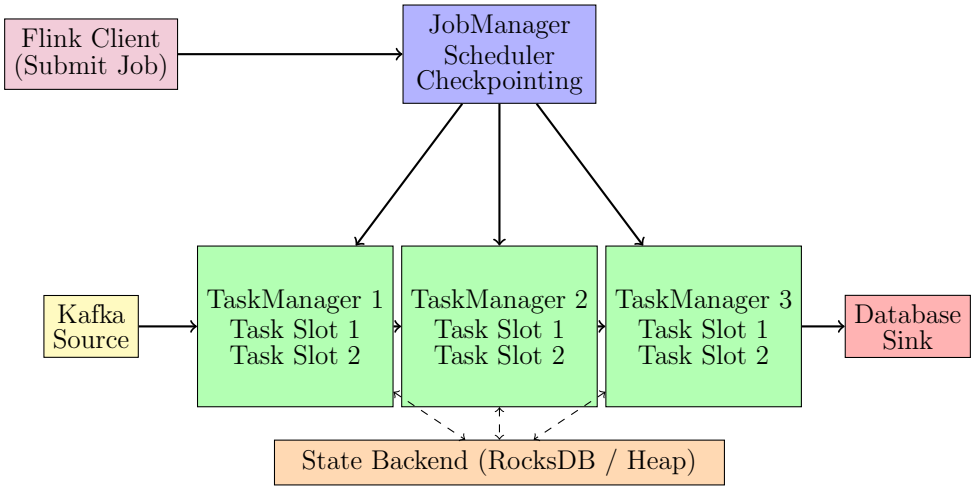


Figure 2.1: Flink Architecture: JobManager, TaskManagers, and State Backend

2.2.1 Key Components

- **JobManager**: Coordinates job execution, scheduling, checkpointing, and recovery
- **TaskManagers**: Worker nodes that execute tasks and maintain local state
- **Task Slots**: Resource units within TaskManagers (typically 1 slot per CPU core)
- **State Backend**: Persistent storage for stateful operations (RocksDB or heap-based)
- **Checkpoints**: Automatic periodic snapshots for fault tolerance
- **Savepoints**: Manual snapshots for upgrades and versioning
- **Watermarks**: Indicators of event time progress for windowing

2.3 Complete Install Instructions (Ubuntu 24.04)

2.3.1 Prerequisites

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Java 11 (Flink requires Java 11 or 17)
sudo apt install openjdk-11-jdk -y

# Verify Java installation
java -version
```

2.3.2 Download and Install Flink

```
# Create directory
mkdir -p ~/flink && cd ~/flink

# Download Flink 1.19.0 (latest stable)
wget https://downloads.apache.org/flink/flink-1.19.0/
   flink-1.19.0-bin-scala_2.12.tgz

# Extract archive
tar -xzf flink-1.19.0-bin-scala_2.12.tgz
cd flink-1.19.0

# Set FLINK_HOME
echo "export FLINK_HOME=~/flink/flink-1.19.0" >> ~/.
   bashrc
echo "export PATH=\$PATH:\$FLINK_HOME/bin" >> ~/.bashrc
source ~/.bashrc
```

2.3.3 Configure Flink

Edit `conf/flink-conf.yaml`:

```
# Increase TaskManager memory
taskmanager.memory.process.size: 4096m

# Set number of task slots per TaskManager
taskmanager.numberOfTaskSlots: 4
```

```
# Enable checkpointing directory (local filesystem)
state.checkpoints.dir: file:///tmp/flink-checkpoints

# Enable savepoints directory
state.savepoints.dir: file:///tmp/flink-savepoints

# Set parallelism default
parallelism.default: 2
```

2.3.4 Start Flink Cluster

```
# Start local cluster (1 JobManager + 1 TaskManager)
./bin/start-cluster.sh

# Verify cluster is running
jps

# Check Flink Web UI
# Open browser: http://localhost:8081

# View logs
tail -f log/flink-*-standalonesession-*.log
```

2.3.5 Stop Flink Cluster

```
./bin/stop-cluster.sh
```

2.4 Working Example: Real-Time Fraud Detection

This example processes credit card transactions from Kafka, detects suspicious patterns, and sends alerts in real-time.

2.4.1 Step 1: Set Up Kafka Topics (from Chapter 1)

```
# Create transactions topic
bin/kafka-topics.sh --create --topic transactions \
  --bootstrap-server localhost:9092 \
  --partitions 4 --replication-factor 1

# Create fraud-alerts topic
bin/kafka-topics.sh --create --topic fraud-alerts \
  --bootstrap-server localhost:9092 \
  --partitions 2 --replication-factor 1
```

2.4.2 Step 2: Create Maven Project

Create `pom.xml`:

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example.flink</groupId>
    <artifactId>fraud-detection</artifactId>
    <version>1.0</version>

    <properties>
        <flink.version>1.19.0</flink.version>
        <scala.binary.version>2.12</scala.binary.version
           >
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-java</artifactId
               >
            <version>${flink.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-kafka</
               artifactId>
            <version>3.0.2-1.18</version>
        </dependency>
        <dependency>
```

```
            <groupId>com.fasterxml.jackson.core</groupId
               >
            <artifactId>jackson-databind</artifactId>
            <version>2.15.2</version>
        </dependency>
    </dependencies>
</project>
```

2.4.3 Step 3: Flink Job - Fraud Detection

Create `FraudDetectionJob.java`:

```
import org.apache.flink.api.common.eventtime.
   WatermarkStrategy;
import org.apache.flink.api.common.functions.
   FilterFunction;
import org.apache.flink.connector.kafka.source.
   KafkaSource;
import org.apache.flink.connector.kafka.sink.KafkaSink;
import org.apache.flink.streaming.api.environment.
   StreamExecutionEnvironment;

public class FraudDetectionJob {
    public static void main(String[] args) throws
       Exception {
        // Set up execution environment
        StreamExecutionEnvironment env =
            StreamExecutionEnvironment.
               getExecutionEnvironment();

        // Enable checkpointing every 10 seconds
        env.enableCheckpointing(10000);

        // Kafka source
        KafkaSource<Transaction> source = KafkaSource
            .<Transaction>builder()
            .setBootstrapServers("localhost:9092")
            .setTopics("transactions")
            .setGroupId("fraud-detector")
            .setValueOnlyDeserializer(new
               TransactionDeserializer())
            .build();
```

```
        // Kafka sink for alerts
        KafkaSink<String> sink = KafkaSink.<String>
           builder()
            .setBootstrapServers("localhost:9092")
            .setRecordSerializer(...)
            .build();

        // Process stream
        env.fromSource(source, WatermarkStrategy.
           noWatermarks(), "Kafka")
           .filter(new FraudDetector())
           .map(tx -> "FRAUD ALERT: " + tx.toString())
           .sinkTo(sink);

        env.execute("Fraud Detection Job");
    }

    public static class FraudDetector implements
       FilterFunction<Transaction> {
        @Override
        public boolean filter(Transaction tx) {
            // Flag transactions > $5000 from new
               accounts
            return tx.amount > 5000 && tx.accountAge <
               30;
        }
    }
}
```

2.4.4 Step 4: Python Flink Example (PyFlink)

Install PyFlink:

```
pip install apache-flink==1.19.0
pip install apache-flink-libraries
```

Create `fraud_detection.py`:

```
from pyflink.datastream import
   StreamExecutionEnvironment
from pyflink.datastream.connectors.kafka import
   KafkaSource, KafkaSink
from pyflink.common.serialization import
   SimpleStringSchema
```

```
from pyflink.common import WatermarkStrategy
import json

def main():
    # Create execution environment
    env = StreamExecutionEnvironment.
       get_execution_environment()
    env.set_parallelism(2)

    # Enable checkpointing
    env.enable_checkpointing(10000)

    # Kafka source
    kafka_source = KafkaSource.builder() \
        .set_bootstrap_servers("localhost:9092") \
        .set_topics("transactions") \
        .set_group_id("fraud-detector") \
        .set_value_only_deserializer(SimpleStringSchema
           ()) \
        .build()

    # Read from Kafka
    stream = env.from_source(
        kafka_source,
        WatermarkStrategy.no_watermarks(),
        "Kafka Source"
    )

    # Fraud detection logic
    def detect_fraud(transaction_json):
        tx = json.loads(transaction_json)
        if tx['amount'] > 5000 and tx['account_age'] <
           30:
            return f"FRAUD: {tx['transaction_id']} -
               Amount: {tx['amount']}"
        return None

    # Process and filter
    alerts = stream.map(detect_fraud).filter(lambda x: x
        is not None)

    # Print alerts
    alerts.print()
```

```
    # Execute
    env.execute("Fraud Detection Job")

if __name__ == '__main__':
    main()
```

Run the job:

```
python fraud_detection.py
```

2.4.5 Step 5: Stateful Processing Example

Track spending per customer with state:

```
from pyflink.datastream import
   StreamExecutionEnvironment
from pyflink.datastream.functions import
   KeyedProcessFunction, RuntimeContext
from pyflink.datastream.state import
   ValueStateDescriptor

class SpendingTracker(KeyedProcessFunction):
    def __init__(self):
        self.total_spending_state = None

    def open(self, runtime_context: RuntimeContext):
        descriptor = ValueStateDescriptor("total-
           spending", float)
        self.total_spending_state = runtime_context.
           get_state(descriptor)

    def process_element(self, value, ctx):
        # Get current total
        current_total = self.total_spending_state.value
           ()
        if current_total is None:
            current_total = 0.0

        # Update total
        new_total = current_total + value['amount']
        self.total_spending_state.update(new_total)

        # Alert if spending > $10,000
```

```
        if new_total > 10000:
            yield f"High spending alert: Customer {value
               ['customer_id']} " \
                  f"has spent ${new_total}"

# Use in stream
env = StreamExecutionEnvironment.
   get_execution_environment()
stream.key_by(lambda tx: tx['customer_id']) \
      .process(SpendingTracker()) \
      .print()
```

2.5 Windowing Operations

Flink supports multiple window types:

```
from pyflink.datastream.window import
   TumblingEventTimeWindows, Time

# Tumbling window: non-overlapping, fixed size
stream.key_by(lambda x: x['customer_id']) \
      .window(TumblingEventTimeWindows.of(Time.minutes
         (5))) \
      .sum('amount')

# Sliding window: overlapping windows
from pyflink.datastream.window import
   SlidingEventTimeWindows
stream.key_by(lambda x: x['customer_id']) \
      .window(SlidingEventTimeWindows.of(Time.minutes
         (10), Time.minutes(5))) \
      .sum('amount')

# Session window: activity-based
from pyflink.datastream.window import
   EventTimeSessionWindows
stream.key_by(lambda x: x['customer_id']) \
      .window(EventTimeSessionWindows.with_gap(Time.
         minutes(30))) \
      .sum('amount')
```

2.6 Flink on Kubernetes

Use the Flink Kubernetes Operator:

```
# Install cert-manager (prerequisite)
kubectl create -f https://github.com/jetstack/cert-
   manager/releases/download/v1.8.2/cert-manager.yaml

# Install Flink Kubernetes Operator
kubectl create -f https://github.com/apache/flink-
   kubernetes-operator/releases/download/release-1.7.0/
   flink-kubernetes-operator-1.7.0.yaml

# Deploy Flink Application
kubectl apply -f - <<EOF
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: fraud-detection
spec:
  image: flink:1.19.0
  flinkVersion: v1_19
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "4"
  serviceAccount: flink
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "4096m"
      cpu: 2
  job:
    jarURI: local:///opt/flink/fraud-detection.jar
    parallelism: 4
    upgradeMode: savepoint
EOF
```

More info: https://github.com/apache/flink-kubernetes-operator

2.7 Performance Tuning

- **Parallelism**: Set based on available CPU cores and data volume
- **Task Slots**: Configure 1 slot per CPU core
- **Memory**: Allocate sufficient heap and off-heap memory
- **State Backend**: Use RocksDB for large state, heap for small state
- **Checkpointing**: Balance frequency (10-60s) vs overhead
- **Network Buffers**: Increase for high-throughput jobs
- **Watermark Interval**: Tune based on latency requirements

2.8 Books

- Stream Processing with Apache Flink (O'Reilly, 2019)
- Flink Forward Conference Talks (YouTube)
- Ververica Training Materials (official documentation)

2.9 Download URL

https://flink.apache.org/downloads.html

2.10 Monitoring and Debugging

- **Web UI**: http://localhost:8081 - Job overview, metrics, backpressure
- **Metrics Reporter**: Configure Prometheus metrics reporter
- **Logging**: Centralize logs with ELK or Grafana Loki
- **Flame Graphs**: Profile CPU usage with async-profiler
- **Checkpointing Stats**: Monitor checkpoint duration and size

2.11 Hardening

- **Security**: Enable Kerberos or SSL/TLS authentication
- **State Encryption**: Encrypt state backend (RocksDB) at rest
- **Network Encryption**: Enable SSL for inter-node communication
- **Resource Limits**: Configure memory and CPU limits
- **Checkpoint Storage**: Use secure distributed filesystem (HDFS/S3/MinIO)
- **Access Control**: Restrict Web UI and REST API access
- **Run as Non-Root**: Use dedicated service account in containers

Chapter 3

Apache Spark

3.1 What Spark Is

Apache Spark is a unified analytics engine for large-scale data processing, originally developed at UC Berkeley's AMPLab in 2009 and open-sourced in 2010. It provides high-level APIs in Java, Scala, Python (PySpark), and R, and an optimized execution engine that supports general computation graphs.

3.1.1 Core Capabilities

- **Batch Processing**: Distributed ETL and data transformations at petabyte scale
- **Streaming**: Micro-batch processing via Structured Streaming (500ms–2s latency)
- **SQL Analytics**: Spark SQL with DataFrame and Dataset APIs
- **Machine Learning**: MLlib library with distributed algorithms
- **Graph Processing**: GraphX for graph-parallel computation
- **In-Memory Computing**: Cache datasets in memory for 100x speedup
- **Fault Tolerance**: Lineage-based recovery via RDD DAG
- **Lazy Evaluation**: Optimizes execution plans before running

3.1.2 Spark vs MapReduce

Feature	Spark	MapReduce
Speed	100x faster (in-memory)	Disk-based (slower)
Ease of Use	High-level APIs	Complex, low-level
Real-time	Structured Streaming	Not supported
ML Support	Native MLlib	Mahout (external)
Iterations	Fast (in-memory)	Very slow (disk I/O)

Table 3.1: Spark vs MapReduce Comparison

3.1.3 Use Cases

- ETL pipelines processing terabytes of data daily
- Real-time analytics dashboards with Structured Streaming
- Machine learning model training at scale (recommendation engines, fraud detection)
- Log processing and aggregation from distributed systems
- Data lake analytics with Parquet, ORC, and Iceberg
- Graph analysis (social networks, fraud detection rings)
- Genomics and scientific data processing

3.2 Architecture Overview

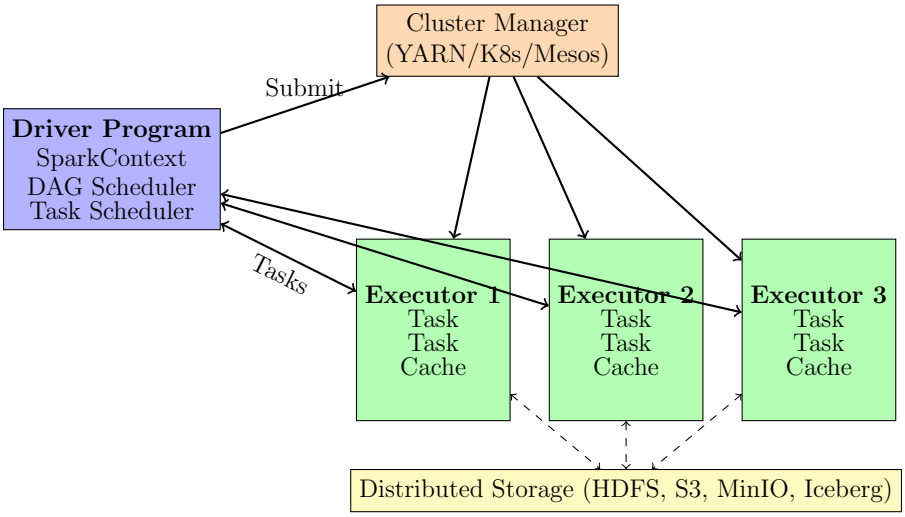


Figure 3.1: Spark Architecture: Driver, Cluster Manager, and Executors

3.2.1 Key Components

- **Driver Program**: Runs main() and creates SparkContext, coordinates job execution
- **SparkContext**: Entry point to Spark functionality, connects to cluster
- **Cluster Manager**: Allocates resources (YARN, Kubernetes, Mesos, Standalone)
- **Executors**: Worker processes that run tasks and cache data
- **Tasks**: Units of work sent to executors
- **RDD (Resilient Distributed Dataset)**: Immutable distributed collection
- **DataFrame**: Distributed table with named columns (like SQL table)
- **DAG (Directed Acyclic Graph)**: Execution plan optimized by Catalyst

3.3 Complete Install Instructions (Ubuntu 24.04)

3.3.1 Prerequisites

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Java 11 or 17
sudo apt install openjdk-11-jdk -y

# Install Python 3
sudo apt install python3 python3-pip -y

# Verify installations
java -version
python3 --version
```

3.3.2 Download and Install Spark

```
# Create directory
mkdir -p ~/spark && cd ~/spark

# Download Spark 3.5.0 with Hadoop 3.3
wget https://dlcdn.apache.org/spark/spark-3.5.0/spark
   -3.5.0-bin-hadoop3.tgz

# Extract
tar -xzf spark-3.5.0-bin-hadoop3.tgz
cd spark-3.5.0-bin-hadoop3

# Set environment variables
echo "export SPARK_HOME=~/spark/spark-3.5.0-bin-hadoop3"
    >> ~/.bashrc
echo "export PATH=\$PATH:\$SPARK_HOME/bin:\$SPARK_HOME/
   sbin" >> ~/.bashrc
echo "export PYSPARK_PYTHON=python3" >> ~/.bashrc
source ~/.bashrc
```

3.3.3 Install PySpark

```
# Install PySpark via pip
pip3 install pyspark==3.5.0

# Install additional libraries
pip3 install pandas numpy matplotlib jupyter
```

3.3.4 Configure Spark

Create/edit `conf/spark-defaults.conf`:

```
# Copy template
cp conf/spark-defaults.conf.template conf/spark-defaults
   .conf

# Edit configuration
cat >> conf/spark-defaults.conf <<EOF
spark.master                     local[*]
spark.driver.memory              4g
spark.executor.memory            4g
spark.executor.cores             2
spark.sql.shuffle.partitions     8
spark.default.parallelism        8
EOF
```

3.3.5 Start Spark

```
# Start Spark shell (Scala)
spark-shell

# Start PySpark shell
pyspark

# Start Spark SQL shell
spark-sql

# Start standalone cluster master
start-master.sh

# Start standalone worker
```

```
start-worker.sh spark://localhost:7077

# Access Spark UI
# http://localhost:4040 (running application)
# http://localhost:8080 (master UI)
```

3.4 Working Example: E-Commerce Sales Analytics

This example demonstrates batch processing and analytics on e-commerce sales data.

3.4.1 Step 1: Generate Sample Data

Create generate_sales_data.py:

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Generate 1 million sales records
np.random.seed(42)
num_records = 1000000

data = {
    'order_id': [f'ORD-{i:07d}' for i in range(
       num_records)],
    'customer_id': np.random.randint(1, 10000,
       num_records),
    'product_id': np.random.randint(1, 500, num_records)
       ,
    'category': np.random.choice(
        ['Electronics', 'Clothing', 'Books', 'Home', '
           Sports'],
        num_records
    ),
    'amount': np.random.uniform(10, 2000, num_records).
       round(2),
    'quantity': np.random.randint(1, 10, num_records),
    'timestamp': [
```

```
        datetime.now() - timedelta(days=np.random.
            randint(0, 365))
        for _ in range(num_records)
    ],
    'country': np.random.choice(
        ['US', 'UK', 'DE', 'FR', 'CA', 'AU'],
        num_records
    )
}

df = pd.DataFrame(data)
df.to_parquet('sales_data.parquet', index=False)
print(f"Generated {num_records} records")
print(df.head())
```

Run the generator:

```
python3 generate_sales_data.py
```

3.4.2 Step 2: PySpark Analytics

Create sales_analytics.py:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.window import Window

# Create Spark session
spark = SparkSession.builder \
    .appName("Sales Analytics") \
    .config("spark.driver.memory", "4g") \
    .config("spark.executor.memory", "4g") \
    .getOrCreate()

# Read Parquet file
df = spark.read.parquet("sales_data.parquet")

print("=== Dataset Info ===")
df.printSchema()
print(f"Total records: {df.count()}")

# 1. Total sales by category
print("\n=== Total Sales by Category ===")
sales_by_category = df.groupBy("category") \
```

```
    .agg(
        count("order_id").alias("total_orders"),
        sum("amount").alias("total_revenue"),
        avg("amount").alias("avg_order_value")
    ) \
    .orderBy(desc("total_revenue"))

sales_by_category.show()

# 2. Top 10 customers by spending
print("\n=== Top 10 Customers ===")
top_customers = df.groupBy("customer_id") \
    .agg(
        count("order_id").alias("num_orders"),
        sum("amount").alias("total_spent")
    ) \
    .orderBy(desc("total_spent")) \
    .limit(10)

top_customers.show()

# 3. Monthly revenue trend
print("\n=== Monthly Revenue Trend ===")
monthly_revenue = df.withColumn("month", date_format("
   timestamp", "yyyy-MM")) \
    .groupBy("month") \
    .agg(sum("amount").alias("revenue")) \
    .orderBy("month")

monthly_revenue.show()

# 4. Revenue by country
print("\n=== Revenue by Country ===")
country_revenue = df.groupBy("country") \
    .agg(
        count("*").alias("orders"),
        sum("amount").alias("revenue")
    ) \
    .orderBy(desc("revenue"))

country_revenue.show()

# 5. Product performance with window functions
```

```
print("\n=== Top 5 Products per Category ===")
window_spec = Window.partitionBy("category").orderBy(
    desc("product_revenue"))

product_performance = df.groupBy("category", "product_id
    ") \
    .agg(
        count("order_id").alias("sales_count"),
        sum("amount").alias("product_revenue")
    ) \
    .withColumn("rank", row_number().over(window_spec))
        \
    .filter(col("rank") <= 5) \
    .orderBy("category", "rank")

product_performance.show(25)

# 6. Cache for repeated analysis
df.cache()

# 7. Save results
sales_by_category.write \
    .mode("overwrite") \
    .parquet("output/sales_by_category")

print("\n=== Analysis Complete ===")

# Stop Spark
spark.stop()
```

Run the analysis:

```
python3 sales_analytics.py
```

3.4.3 Step 3: Spark SQL Example

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("SQL Example").
    getOrCreate()

# Read data
df = spark.read.parquet("sales_data.parquet")
```

```
# Register as temp view
df.createOrReplaceTempView("sales")

# SQL queries
result = spark.sql("""
    SELECT
        category,
        country,
        COUNT(*) as order_count,
        SUM(amount) as total_revenue,
        AVG(amount) as avg_order_value
    FROM sales
    WHERE amount > 100
    GROUP BY category, country
    HAVING total_revenue > 10000
    ORDER BY total_revenue DESC
""")

result.show(20)

# Complex query with CTEs
result2 = spark.sql("""
    WITH customer_stats AS (
        SELECT
            customer_id,
            COUNT(*) as num_orders,
            SUM(amount) as lifetime_value
        FROM sales
        GROUP BY customer_id
    )
    SELECT
        CASE
            WHEN lifetime_value > 10000 THEN 'VIP'
            WHEN lifetime_value > 5000 THEN 'Premium'
            WHEN lifetime_value > 1000 THEN 'Regular'
            ELSE 'New'
        END as customer_tier,
        COUNT(*) as customer_count,
        AVG(lifetime_value) as avg_ltv
    FROM customer_stats
    GROUP BY customer_tier
    ORDER BY avg_ltv DESC
```

```
""")

result2.show()
```

3.4.4 Step 4: Structured Streaming Example

Read from Kafka and process in real-time:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

spark = SparkSession.builder \
    .appName("Real-time Sales") \
    .getOrCreate()

# Define schema
schema = StructType([
    StructField("order_id", StringType()),
    StructField("customer_id", IntegerType()),
    StructField("amount", DoubleType()),
    StructField("category", StringType()),
    StructField("timestamp", TimestampType())
])

# Read from Kafka
sales_stream = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092")
        \
    .option("subscribe", "orders") \
    .load()

# Parse JSON
parsed_stream = sales_stream \
    .select(from_json(col("value").cast("string"),
       schema).alias("data")) \
    .select("data.*")

# Aggregate: revenue per category per 5-minute window
windowed_revenue = parsed_stream \
    .withWatermark("timestamp", "10 minutes") \
    .groupBy(
```

```
        window("timestamp", "5 minutes", "1 minute"),
        "category"
    ) \
    .agg(
        count("order_id").alias("num_orders"),
        sum("amount").alias("revenue")
    )

# Write to console
query = windowed_revenue.writeStream \
    .outputMode("update") \
    .format("console") \
    .option("truncate", False) \
    .start()

query.awaitTermination()
```

3.4.5 Step 5: Machine Learning Example

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

# Prepare features
df = spark.read.parquet("sales_data.parquet")

# Create features vector
assembler = VectorAssembler(
    inputCols=["customer_id", "product_id", "quantity"],
    outputCol="features"
)

df_ml = assembler.transform(df).select("features", "
   amount")

# Split data
train, test = df_ml.randomSplit([0.8, 0.2], seed=42)

# Train model
lr = LinearRegression(featuresCol="features", labelCol="
   amount")
model = lr.fit(train)
```

```
# Predictions
predictions = model.transform(test)

# Evaluate
evaluator = RegressionEvaluator(
    labelCol="amount",
    predictionCol="prediction",
    metricName="rmse"
)
rmse = evaluator.evaluate(predictions)
print(f"RMSE: {rmse}")

# Show coefficients
print(f"Coefficients: {model.coefficients}")
print(f"Intercept: {model.intercept}")
```

3.5 Spark on Kubernetes

```
# Submit Spark job to Kubernetes
spark-submit \
  --master k8s://https://kubernetes-api:6443 \
  --deploy-mode cluster \
  --name sales-analytics \
  --conf spark.executor.instances=5 \
  --conf spark.kubernetes.container.image=apache/spark
     :3.5.0 \
  --conf spark.kubernetes.namespace=spark \
  --conf spark.kubernetes.authenticate.driver.
     serviceAccountName=spark \
  sales_analytics.py

# Using Spark Operator
kubectl apply -f - <<EOF
apiVersion: sparkoperator.k8s.io/v1beta2
kind: SparkApplication
metadata:
  name: sales-analytics
  namespace: spark
spec:
  type: Python
```

```
  pythonVersion: "3"
  mode: cluster
  image: "apache/spark-py:3.5.0"
  mainApplicationFile: "local:///opt/spark/work-dir/
     sales_analytics.py"
  sparkVersion: "3.5.0"
  driver:
    cores: 1
    memory: "2g"
  executor:
    cores: 2
    instances: 3
    memory: "4g"
EOF
```

3.6 Performance Tuning

- **Partitioning**: Use 2-3 partitions per CPU core
- **Memory**: Allocate 60-75% of node memory to Spark
- **Serialization**: Use Kryo serializer for better performance
- **Caching**: Cache frequently-used DataFrames with `df.cache()`
- **Broadcasting**: Broadcast small lookup tables (¡ 10MB)
- **Shuffle Partitions**: Tune `spark.sql.shuffle.partitions` (default 200)
- **Dynamic Allocation**: Enable to scale executors automatically
- **File Format**: Use Parquet or ORC for columnar storage

3.7 Books

- Learning Spark, 2nd Edition (O'Reilly, 2020)
- Spark: The Definitive Guide (O'Reilly, 2018)
- High Performance Spark (O'Reilly, 2017)
- Advanced Analytics with Spark (O'Reilly, 2017)

3.8 Download URL

https://spark.apache.org/downloads.html

3.9 Monitoring

- **Spark UI**: http://localhost:4040 - Job stages, tasks, storage
- **History Server**: Persist UI data after job completion
- **Metrics**: Export to Prometheus/Grafana
- **Logs**: Centralize with ELK or Loki
- **Dr. Elephant**: LinkedIn's performance tuning tool

Key metrics to monitor:

- Task execution time and data skew
- Shuffle read/write bytes
- GC time percentage
- Executor memory usage
- Stage duration and failures

3.10 Integration

Spark integrates with:

- **Kafka**: Structured Streaming source/sink
- **Iceberg/Delta Lake/Hudi**: ACID table formats
- **Hive Metastore**: Schema registry
- **S3/MinIO/HDFS**: Distributed storage
- **Cassandra**: Spark-Cassandra connector
- **Elasticsearch**: Spark-ES connector
- **JDBC**: Connect to RDBMS (PostgreSQL, MySQL, etc.)

3.11 Hardening

- **Authentication**: Enable Kerberos or SASL
- **Encryption**: Enable SSL/TLS for RPC and HTTP
- **Authorization**: Use Apache Ranger or Sentry
- **Network Security**: Firewall executor ports
- **Secrets Management**: Use Kubernetes secrets or Vault
- **Resource Limits**: Configure quotas to prevent resource exhaustion
- **Audit Logging**: Enable event logging
- **Data Encryption**: Encrypt data at rest and in transit

Chapter 4

Apache Iceberg

4.1 What Iceberg Is

Apache Iceberg is an open table format for huge analytic datasets designed to bring ACID transactions, schema evolution, and time travel to data lakes. Developed at Netflix in 2017 and donated to Apache in 2018, Iceberg solves many of the limitations of the traditional Hive table format.

4.1.1 Core Capabilities

- **ACID Transactions**: Serializable isolation for concurrent writes
- **Schema Evolution**: Add, drop, rename columns without rewriting data
- **Partition Evolution**: Change partitioning scheme without rewriting
- **Time Travel**: Query historical table snapshots
- **Hidden Partitioning**: Users don't need to know partition structure
- **Snapshot Isolation**: Consistent reads while writes are happening
- **Scalable Metadata**: Metadata stored in Parquet/Avro, not filesystem listings
- **Object Storage Optimized**: Works efficiently on S3, Azure Blob, GCS, MinIO

4.1.2 Iceberg vs Hive vs Delta Lake

Feature	**Iceberg**	**Delta Lake**	**Hive**
ACID	Yes	Yes	No
Schema Evolution	Full support	Limited	Partial
Partition Evolution	Yes	No	No
Time Travel	Yes	Yes	No
Hidden Partitioning	Yes	No	No
Multi-Engine	Spark/Flink/Trino	Mostly Spark	Most engines
Metadata Scaling	Excellent	Good	Poor (filesystem)

Table 4.1: Iceberg vs Delta Lake vs Hive Comparison

4.1.3 Use Cases

- Building ACID-compliant data lakes on object storage
- Real-time streaming writes from Flink/Spark Streaming to S3
- Large-scale ETL pipelines with evolving schemas
- Multi-petabyte analytics with time travel and rollback
- Replacing Hive tables with better performance and features
- Slowly Changing Dimension (SCD) Type 2 tables
- Data quality workflows with atomic swaps

4.2 Architecture Overview

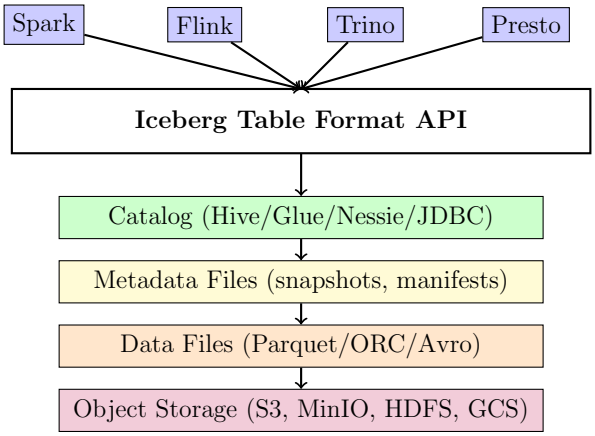


Figure 4.1: Iceberg Architecture: Multi-Engine Support with Metadata Layers

4.2.1 Key Components

- **Catalog**: Maps table names to current metadata file location
- **Metadata Files**: Store table schema, partitioning, and snapshot history
- **Manifest Lists**: Track which data files belong to which snapshot
- **Manifest Files**: List of data files with statistics (row counts, bounds)
- **Data Files**: Actual Parquet/ORC/Avro files containing table data
- **Snapshots**: Immutable table state at a point in time

4.3 Installation with Spark

4.3.1 Add Iceberg to Spark

```
# Download Iceberg JAR for Spark 3.5
wget https://repo1.maven.org/maven2/org/apache/iceberg/
   iceberg-spark-runtime-3.5_2.12/1.5.0/iceberg-spark-
   runtime-3.5_2.12-1.5.0.jar

# Copy to Spark jars directory
cp iceberg-spark-runtime-3.5_2.12-1.5.0.jar $SPARK_HOME/
   jars/
```

4.3.2 Configure Spark for Iceberg

Create `spark-iceberg.sh`:

```
#!/bin/bash

pyspark \
  --packages org.apache.iceberg:iceberg-spark-runtime
     -3.5_2.12:1.5.0 \
  --conf spark.sql.extensions=org.apache.iceberg.spark.
     extensions.IcebergSparkSessionExtensions \
  --conf spark.sql.catalog.spark_catalog=org.apache.
     iceberg.spark.SparkSessionCatalog \
  --conf spark.sql.catalog.spark_catalog.type=hive \
  --conf spark.sql.catalog.local=org.apache.iceberg.
     spark.SparkCatalog \
  --conf spark.sql.catalog.local.type=hadoop \
  --conf spark.sql.catalog.local.warehouse=/tmp/iceberg-
     warehouse
```

4.4 Working Example: Sales Data Lake with Time Travel

4.4.1 Step 1: Create Iceberg Table

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Create Spark session with Iceberg
spark = SparkSession.builder \
    .appName("Iceberg Example") \
    .config("spark.jars.packages", "org.apache.iceberg:
       iceberg-spark-runtime-3.5_2.12:1.5.0") \
    .config("spark.sql.extensions", "org.apache.iceberg.
       spark.extensions.IcebergSparkSessionExtensions")
       \
    .config("spark.sql.catalog.local", "org.apache.
       iceberg.spark.SparkCatalog") \
    .config("spark.sql.catalog.local.type", "hadoop") \
    .config("spark.sql.catalog.local.warehouse", "/tmp/
       iceberg-warehouse") \
    .getOrCreate()

# Create database
spark.sql("CREATE DATABASE IF NOT EXISTS local.sales_db"
   )

# Create Iceberg table
spark.sql("""
    CREATE TABLE IF NOT EXISTS local.sales_db.orders (
        order_id STRING,
        customer_id INT,
        product STRING,
        amount DOUBLE,
        order_date DATE,
        country STRING
    )
    USING iceberg
    PARTITIONED BY (order_date)
""")

print("Iceberg table created!")
```

4.4.2 Step 2: Insert Data

```
# Generate sample data
from datetime import date, timedelta
import random

orders_data = []
for i in range(1000):
    orders_data.append({
        'order_id': f'ORD-{i:05d}',
        'customer_id': random.randint(1, 100),
        'product': random.choice(['Laptop', 'Phone', '
          Tablet', 'Monitor']),
        'amount': round(random.uniform(100, 2000), 2),
        'order_date': date.today() - timedelta(days=
          random.randint(0, 365)),
        'country': random.choice(['US', 'UK', 'DE', 'FR'
          , 'CA'])
    })

df = spark.createDataFrame(orders_data)

# Insert into Iceberg table
df.writeTo("local.sales_db.orders").append()

print(f"Inserted {df.count()} orders")
```

4.4.3 Step 3: Query Iceberg Table

```
# Read from Iceberg table
orders = spark.table("local.sales_db.orders")

print("=== Total Orders ===")
print(f"Total: {orders.count()}")

print("\n=== Orders by Country ===")
orders.groupBy("country") \
    .agg(
        count("order_id").alias("num_orders"),
        sum("amount").alias("total_revenue")
    ) \
    .orderBy(desc("total_revenue")) \
```

```
    .show()

# Query with predicate pushdown (partition pruning)
recent_orders = spark.sql("""
    SELECT *
    FROM local.sales_db.orders
    WHERE order_date >= DATE_SUB(CURRENT_DATE, 30)
""")

print(f"\nOrders in last 30 days: {recent_orders.count()
   }")
```

4.4.4 Step 4: Schema Evolution

```
# Add new column without rewriting data
spark.sql("""
    ALTER TABLE local.sales_db.orders
    ADD COLUMN shipping_address STRING
""")

# Rename column
spark.sql("""
    ALTER TABLE local.sales_db.orders
    RENAME COLUMN product TO product_name
""")

# Drop column
spark.sql("""
    ALTER TABLE local.sales_db.orders
    DROP COLUMN shipping_address
""")

# Show updated schema
spark.table("local.sales_db.orders").printSchema()
```

4.4.5 Step 5: Time Travel

```
# Show table history
history = spark.sql("SELECT * FROM local.sales_db.orders
    .history")
history.show(truncate=False)

# Get snapshot IDs
snapshots = spark.sql("SELECT * FROM local.sales_db.
    orders.snapshots")
snapshots.select("snapshot_id", "committed_at").show()

# Query specific snapshot (time travel)
snapshot_id = snapshots.first()['snapshot_id']

df_snapshot = spark.read \
    .option("snapshot-id", snapshot_id) \
    .table("local.sales_db.orders")

print(f"\nRecords in snapshot {snapshot_id}: {
    df_snapshot.count()}")

# Query as of timestamp
df_yesterday = spark.read \
    .option("as-of-timestamp", "2024-01-15 00:00:00") \
    .table("local.sales_db.orders")
```

4.4.6 Step 6: Partition Evolution

```
# Change partitioning without rewriting data
spark.sql("""
    ALTER TABLE local.sales_db.orders
    ADD PARTITION FIELD country
""")

# Remove partition field
spark.sql("""
    ALTER TABLE local.sales_db.orders
    DROP PARTITION FIELD order_date
""")

# View partition spec evolution
```

```
spec_history = spark.sql("SELECT * FROM local.sales_db.
   orders.refs")
spec_history.show()
```

4.4.7 Step 7: Update and Delete (ACID Operations)

```
# Update records
spark.sql("""
    UPDATE local.sales_db.orders
    SET amount = amount * 1.1
    WHERE country = 'US'
""")

# Delete records
spark.sql("""
    DELETE FROM local.sales_db.orders
    WHERE order_date < DATE_SUB(CURRENT_DATE, 365)
""")

# Merge (UPSERT)
updates_df = spark.createDataFrame([
    ('ORD-00001', 150, 9999.99),
    ('ORD-99999', 200, 5000.00)
], ['order_id', 'customer_id', 'amount'])

updates_df.createOrReplaceTempView("updates")

spark.sql("""
    MERGE INTO local.sales_db.orders t
    USING updates s
    ON t.order_id = s.order_id
    WHEN MATCHED THEN
        UPDATE SET t.amount = s.amount
    WHEN NOT MATCHED THEN
        INSERT (order_id, customer_id, amount)
        VALUES (s.order_id, s.customer_id, s.amount)
""")
```

4.4.8 Step 8: Maintenance Operations

```
# Expire old snapshots (cleanup metadata)
spark.sql("""
    CALL local.system.expire_snapshots(
        table => 'sales_db.orders',
        older_than => TIMESTAMP '2024-01-01 00:00:00',
        retain_last => 10
    )
""")

# Remove orphan files
spark.sql("""
    CALL local.system.remove_orphan_files(
        table => 'sales_db.orders'
    )
""")

# Compact data files
spark.sql("""
    CALL local.system.rewrite_data_files(
        table => 'sales_db.orders',
        options => map('target-file-size-bytes',
           '134217728')
    )
""")

# Rewrite manifests
spark.sql("""
    CALL local.system.rewrite_manifests('sales_db.orders
       ')
""")
```

4.5 Flink Integration

```
import org.apache.flink.streaming.api.environment.
    StreamExecutionEnvironment;
import org.apache.flink.table.api.bridge.java.
    StreamTableEnvironment;

StreamExecutionEnvironment env =
    StreamExecutionEnvironment.getExecutionEnvironment();
StreamTableEnvironment tEnv = StreamTableEnvironment.
    create(env);

// Create Iceberg catalog
tEnv.executeSql("CREATE CATALOG iceberg_catalog WITH ("
    +
     "'type'='iceberg'," +
     "'catalog-type'='hadoop'," +
     "'warehouse'='file:///tmp/iceberg-warehouse'" +
     ")");

// Use catalog
tEnv.executeSql("USE CATALOG iceberg_catalog");

// Stream writes to Iceberg
tEnv.executeSql("INSERT INTO sales_db.orders " +
     "SELECT order_id, customer_id, amount FROM
        kafka_orders");
```

4.6 Catalog Types

Iceberg supports multiple catalog implementations:

- **Hive Metastore**: Most common for Spark/Trino integration
- **AWS Glue**: Native AWS catalog integration
- **Hadoop**: Simple filesystem-based catalog
- **JDBC**: Relational database as catalog
- **Nessie**: Git-like data lake with branches and tags

- **REST**: Custom REST catalog server

4.7 Performance Tuning

- **File Size**: Target 128MB–512MB data files
- **Partition Pruning**: Use partition filters in queries
- **Metadata Caching**: Enable catalog caching
- **Vectorized Reads**: Use Parquet with vectorization
- **Compaction**: Regularly compact small files
- **Snapshot Expiration**: Clean old snapshots to reduce metadata

4.8 Books and Resources

- Apache Iceberg: The Definitive Guide (O'Reilly, 2024)
- Iceberg documentation: https://iceberg.apache.org/docs/latest
- Netflix Tech Blog: Original Iceberg announcement

4.9 URL

https://iceberg.apache.org

4.10 Integration

Iceberg integrates with:

- **Spark**: Read/write, DDL, time travel
- **Flink**: Streaming writes, batch reads
- **Trino/Presto**: Fast OLAP queries
- **Hive**: Legacy Hive table migration

- **Dremio/Starburst**: Commercial query engines
- **dbt**: Data transformation workflows
- **AWS Athena**: Serverless queries on S3

4.11 Hardening

- **Catalog Security**: Secure Hive Metastore or Glue with IAM
- **Encryption**: Enable S3/MinIO server-side encryption
- **Access Control**: Use Ranger/Sentry for table-level permissions
- **Audit Logging**: Track table access and modifications
- **Snapshot Retention**: Define retention policies for compliance
- **Data Lineage**: Track snapshot provenance

Chapter 5

Apache Doris

5.1 What Doris Is

Apache Doris is a high-performance, real-time MPP (Massively Parallel Processing) analytical database designed for OLAP workloads. Originally developed at Baidu as Palo and donated to Apache in 2018, Doris provides sub-second query latency on massive datasets while supporting high-concurrency queries.

5.1.1 Core Capabilities

- **Sub-Second Queries**: Optimized columnar storage and vectorized execution
- **Real-Time Ingestion**: Stream data from Kafka, Flink with minimal latency
- **High Concurrency**: Handles thousands of concurrent queries
- **MySQL Protocol**: Compatible with MySQL clients and tools
- **Materialized Views**: Automatic query rewriting for acceleration
- **Distributed Joins**: MPP architecture for complex analytics
- **Rollup Tables**: Pre-aggregated data for faster queries
- **Storage Engine**: Columnar format with bitmap indexes

5.1.2 Use Cases

- Real-time dashboards and BI reporting
- Ad-hoc analytics on billions of rows
- User behavior analysis at web scale
- Real-time data warehousing
- Replacing traditional OLAP cubes
- Log analytics and monitoring

5.2 Architecture

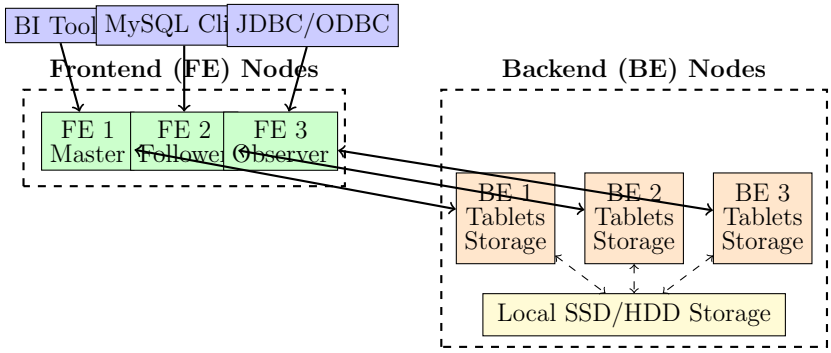


Figure 5.1: Doris Architecture: Frontend and Backend Separation

5.2.1 Key Components

- **Frontend (FE)**: Query parsing, planning, metadata management, and coordination
- **Backend (BE)**: Data storage and query execution
- **Tablets**: Horizontal shards of table data
- **Replicas**: Data copies for fault tolerance (default 3)
- **Broker**: Connector for external data sources (HDFS, S3)

5.3 Complete Install Instructions (Ubuntu 24.04)

5.3.1 Prerequisites

```
# Install Java 8 or 11
sudo apt update
sudo apt install openjdk-11-jdk -y

# Verify
java -version
```

5.3.2 Download and Install Doris

```
# Create directory
mkdir -p ~/doris && cd ~/doris

# Download Doris 2.0 (FE + BE combined package)
wget https://downloads.apache.org/doris/2.0.4/apache-
   doris-2.0.4-bin-x86_64.tar.gz

# Extract
tar -xzf apache-doris-2.0.4-bin-x86_64.tar.gz

# Set environment
echo "export DORIS_HOME=~/doris/apache-doris-2.0.4-bin-
   x86_64" >> ~/.bashrc
source ~/.bashrc
```

5.3.3 Start Frontend (FE)

```
cd $DORIS_HOME/fe

# Start FE
./bin/start_fe.sh --daemon

# Check logs
tail -f log/fe.log

# Verify FE is running (Web UI)
# http://localhost:8030
```

5.3.4 Start Backend (BE)

```
cd $DORIS_HOME/be

# Start BE
./bin/start_be.sh --daemon

# Check logs
tail -f log/be.log
```

5.3.5 Add BE to Cluster

```
# Connect to Doris using MySQL client
mysql -h 127.0.0.1 -P 9030 -u root

# Add BE node
ALTER SYSTEM ADD BACKEND "127.0.0.1:9050";

# Check backends
SHOW BACKENDS\G

# Exit
exit;
```

5.4 Working Example: E-Commerce Analytics

5.4.1 Step 1: Connect and Create Database

```
# Connect via MySQL client
mysql -h 127.0.0.1 -P 9030 -u root
```

```
-- Create database
CREATE DATABASE ecommerce;
USE ecommerce;

-- Create orders table (Duplicate Key model for
   analytics)
CREATE TABLE orders (
```

```
    order_id BIGINT,
    user_id INT,
    product_id INT,
    amount DECIMAL(10,2),
    order_date DATE,
    country VARCHAR(50)
)
DUPLICATE KEY(order_id)
PARTITION BY RANGE(order_date) (
    PARTITION p202401 VALUES LESS THAN ("2024-02-01"),
    PARTITION p202402 VALUES LESS THAN ("2024-03-01"),
    PARTITION p202403 VALUES LESS THAN ("2024-04-01")
)
DISTRIBUTED BY HASH(order_id) BUCKETS 10
PROPERTIES (
    "replication_num" = "1",
    "storage_format" = "V2"
);

-- Show table structure
DESC orders;
```

5.4.2 Step 2: Load Data via Stream Load

Create sample data file `orders.csv`:

```
cat > orders.csv <<EOF
1,101,501,299.99,2024-01-15,US
2,102,502,149.50,2024-01-16,UK
3,103,501,299.99,2024-01-17,DE
4,101,503,599.00,2024-01-18,US
5,104,504,89.99,2024-01-19,FR
EOF
```

Load via curl:

```
curl --location-trusted -u root: \
  -H "label:order_load_1" \
  -H "column_separator:," \
  -T orders.csv \
  http://127.0.0.1:8030/api/ecommerce/orders/
     _stream_load

# Check load status
```

```
mysql -h 127.0.0.1 -P 9030 -u root -e "SHOW LOAD FROM
    ecommerce;"
```

5.4.3 Step 3: Query Analytics

```
-- Total revenue
SELECT SUM(amount) as total_revenue FROM orders;

-- Revenue by country
SELECT
    country,
    COUNT(*) as order_count,
    SUM(amount) as revenue,
    AVG(amount) as avg_order_value
FROM orders
GROUP BY country
ORDER BY revenue DESC;

-- Daily revenue trend
SELECT
    order_date,
    COUNT(*) as orders,
    SUM(amount) as revenue
FROM orders
GROUP BY order_date
ORDER BY order_date;

-- Top users by spending
SELECT
    user_id,
    COUNT(*) as orders,
    SUM(amount) as total_spent
FROM orders
GROUP BY user_id
ORDER BY total_spent DESC
LIMIT 10;
```

5.4.4 Step 4: Create Rollup Table (Materialized View)

```
-- Create rollup for country aggregations
ALTER TABLE orders
ADD ROLLUP rollup_country (country, order_date, amount);

-- Check rollup status
SHOW ALTER TABLE ROLLUP FROM ecommerce;

-- Query will automatically use rollup
SELECT country, SUM(amount)
FROM orders
GROUP BY country;
```

5.4.5 Step 5: Integrate with Kafka

```
-- Create routine load job to continuously ingest from
   Kafka
CREATE ROUTINE LOAD ecommerce.kafka_orders ON orders
COLUMNS(order_id, user_id, product_id, amount,
   order_date, country)
PROPERTIES (
    "desired_concurrent_number" = "3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false"
)
FROM KAFKA (
    "kafka_broker_list" = "localhost:9092",
    "kafka_topic" = "orders",
    "property.group.id" = "doris_orders",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING
       "
);

-- Show routine load jobs
SHOW ROUTINE LOAD FOR ecommerce.kafka_orders\G

-- Pause load
PAUSE ROUTINE LOAD FOR ecommerce.kafka_orders;
```

```
-- Resume load
RESUME ROUTINE LOAD FOR ecommerce.kafka_orders;
```

5.4.6 Step 6: Python Integration

```
import pymysql

# Connect to Doris
conn = pymysql.connect(
    host='127.0.0.1',
    port=9030,
    user='root',
    password='',
    database='ecommerce'
)

cursor = conn.cursor()

# Query data
cursor.execute("""
    SELECT country, SUM(amount) as revenue
    FROM orders
    GROUP BY country
    ORDER BY revenue DESC
""")

results = cursor.fetchall()
for row in results:
    print(f"Country: {row[0]}, Revenue: ${row[1]}")

cursor.close()
conn.close()
```

5.5 Table Models

Doris supports multiple data models:

- **Duplicate**: Store raw data without aggregation (like orders table)
- **Aggregate**: Pre-aggregate metrics (SUM, MIN, MAX, REPLACE)

- **Unique**: Upsert model with primary key
- **Primary**: Row-level updates and deletes

5.6 Performance Tuning

- **Partitioning**: Use range partitioning on date columns
- **Bucketing**: Choose appropriate bucket count (10-50 per node)
- **Bloom Filters**: Enable for high-cardinality columns
- **Rollups**: Create for common aggregation patterns
- **Compaction**: Monitor and tune compaction policies
- **Query Cache**: Enable result cache for repeated queries

5.7 Monitoring

- **Web UI**: `http://localhost:8030` - Query stats, metrics
- **System Tables**: Query `information_schema`
- **Metrics**: Export Prometheus metrics from BE nodes
- **Logs**: Monitor FE and BE logs for errors

5.8 URL

`https://doris.apache.org`

5.9 Books

- Apache Doris documentation (comprehensive official docs)
- Doris community blog posts and case studies

5.10 Integration

Doris integrates with:

- **Kafka**: Routine load for streaming ingestion
- **Spark**: Spark-Doris connector for ETL
- **Flink**: Flink-Doris connector for real-time writes
- **HDFS/S3**: Broker load for batch imports
- **Superset/Grafana**: BI dashboarding
- **dbt**: Data transformation workflows

5.11 Hardening

- **Authentication**: Enable user authentication and SSL
- **RBAC**: Create users with specific database permissions
- **Network**: Firewall FE (9030) and BE (9050) ports
- **Encryption**: Enable data-at-rest encryption
- **Backups**: Regular snapshots of metadata and data
- **Resource Isolation**: Configure query queues and resource groups

Chapter 6

Apache AsterixDB

6.1 What AsterixDB Is

Apache AsterixDB is a scalable Big Data Management System (BDMS) designed for managing semi-structured data at scale. Developed at UC Irvine and open-sourced in 2015, AsterixDB provides a rich query language (SQL++) and high-performance ingestion for JSON and other semi-structured formats.

6.1.1 Core Capabilities

- **SQL++ Query Language**: Extended SQL for semi-structured data
- **Native JSON Support**: Store and query JSON documents efficiently
- **Data Feeds**: Continuous ingestion from external sources
- **LSM-Based Storage**: Log-Structured Merge tree for writes
- **Parallel Query Processing**: MPP architecture
- **Secondary Indexes**: B-tree, R-tree, inverted indexes
- **Full-Text Search**: Built-in text analysis

6.1.2 Use Cases

- IoT sensor data management
- Social media analytics
- Web click stream analysis
- Semi-structured log analytics
- Document-oriented applications

6.2 Complete Install Instructions (Ubuntu 24.04)

6.2.1 Prerequisites

```
# Install Java 11
sudo apt update
sudo apt install openjdk-11-jdk -y
java -version
```

6.2.2 Download and Install

```
# Download AsterixDB
mkdir -p ~/asterixdb && cd ~/asterixdb
wget https://dlcdn.apache.org/asterixdb/asterixdb-server
   -0.9.10/asterixdb-server-0.9.10-binary-assembly.zip

# Extract
unzip asterixdb-server-0.9.10-binary-assembly.zip
cd asterixdb

# Start AsterixDB
./bin/asterixdb start

# Verify running
# Web UI: http://localhost:19001
# Query: http://localhost:19002
```

6.3 Working Example: Social Media Analytics

6.3.1 Step 1: Create Dataverse and Dataset

Connect to query interface at `http://localhost:19006`:

```
-- Create dataverse (like database)
CREATE DATAVERSE social_media;
USE social_media;

-- Define type for tweets
CREATE TYPE TweetType AS {
    id: int64,
    user_id: int64,
    text: string,
    hashtags: [string],
    created_at: datetime,
    likes: int32,
    retweets: int32
};

-- Create dataset
CREATE DATASET tweets(TweetType)
PRIMARY KEY id;
```

6.3.2 Step 2: Insert Data

```
-- Insert sample tweets
INSERT INTO tweets ([
    {
        "id": 1,
        "user_id": 101,
        "text": "Loving Apache AsterixDB for big data!",
        "hashtags": ["BigData", "AsterixDB"],
        "created_at": datetime("2024-01-15T10:30:00"),
        "likes": 45,
        "retweets": 12
    },
    {
        "id": 2,
```

```
        "user_id": 102,
        "text": "Real-time analytics made easy",
        "hashtags": ["Analytics", "RealTime"],
        "created_at": datetime("2024-01-15T11:00:00"),
        "likes": 89,
        "retweets": 23
    }
]);
```

6.3.3 Step 3: Query with SQL++

```
-- Get all tweets
SELECT * FROM tweets;

-- Find popular tweets
SELECT t.text, t.likes, t.retweets
FROM tweets t
WHERE t.likes > 50
ORDER BY t.likes DESC;

-- Analyze hashtags
SELECT h AS hashtag, COUNT(*) AS tweet_count
FROM tweets t, t.hashtags h
GROUP BY h
ORDER BY tweet_count DESC;

-- Time-based aggregation
SELECT
    get-day(t.created_at) AS day,
    COUNT(*) AS tweets,
    SUM(t.likes) AS total_likes
FROM tweets t
GROUP BY get-day(t.created_at);
```

6.3.4 Step 4: Create Feed for Continuous Ingestion

```
-- Create feed from file
CREATE FEED tweet_feed USING file_feed
(("type"="localfs"),
 ("path"="localhost:///data/tweets.json"),
 ("format"="json"));

-- Connect feed to dataset
CONNECT FEED tweet_feed TO DATASET tweets;

-- Start feed
START FEED tweet_feed;

-- Stop feed
STOP FEED tweet_feed;
```

6.3.5 Step 5: Create Secondary Indexes

```
-- Create B-tree index on user_id
CREATE INDEX user_idx ON tweets(user_id);

-- Create keyword index for full-text search
CREATE INDEX text_idx ON tweets(text) TYPE KEYWORD;

-- Query using index
SELECT * FROM tweets
WHERE contains(text, "analytics");
```

6.4 Python Integration

```
import requests
import json

# AsterixDB Query API
ASTERIX_URL = "http://localhost:19002/query/service"

def run_query(query):
    response = requests.post(
        ASTERIX_URL,
```

```
        data={"statement": query},
        headers={"Content-Type": "application/x-www-form
            -urlencoded"}
    )
    return response.json()

# Execute query
query = """
USE social_media;
SELECT t.text, t.likes
FROM tweets t
WHERE t.likes > 50
ORDER BY t.likes DESC;
"""

results = run_query(query)
print(json.dumps(results, indent=2))
```

6.5 Performance Tuning

- **Partitioning**: Partition large datasets by key
- **Indexing**: Create secondary indexes on frequently queried fields
- **Memory**: Configure JVM heap size appropriately
- **Feeds**: Tune ingestion batch sizes
- **Compaction**: Schedule LSM compaction during off-peak hours

6.6 URL

https://asterixdb.apache.org

6.7 Books and Resources

- Official AsterixDB documentation
- Academic papers from UC Irvine research group
- SQL++ language specification

6.8 Integration

- **Spark**: Read/write via connectors
- **Kafka**: Ingest via feeds
- **HDFS**: External dataset support

6.9 Hardening

- **Network**: Run behind reverse proxy/ingress
- **Ports**: Restrict node-to-node communication ports
- **Maintenance**: Schedule periodic LSM compaction
- **Backups**: Regular snapshots of metadata and data
- **Authentication**: Configure access control

Chapter 7

Apache Cassandra

7.1 What Cassandra Is

Apache Cassandra is a distributed, masterless NoSQL database designed for handling massive amounts of data across multiple data centers with no single point of failure. Originally developed at Facebook and open-sourced in 2008, Cassandra provides linear scalability and high availability for mission-critical applications.

7.1.1 Core Capabilities

- **Masterless Architecture**: No single point of failure, all nodes equal
- **Linear Scalability**: Add nodes to increase capacity
- **High Write Throughput**: Hundreds of thousands of writes/sec per cluster
- **Tunable Consistency**: Choose between consistency and availability (CAP theorem)
- **Multi-DC Replication**: Built-in geo-distribution
- **LSM Storage**: SSTables + memtables for high write performance
- **CQL**: Cassandra Query Language (SQL-like)

7.1.2 Use Cases

- Time-series data (IoT sensor data, metrics)
- User activity tracking and personalization
- Messaging and notification systems
- Product catalogs and inventory management
- Fraud detection systems requiring high availability
- Global applications needing multi-region deployment

7.2 Architecture

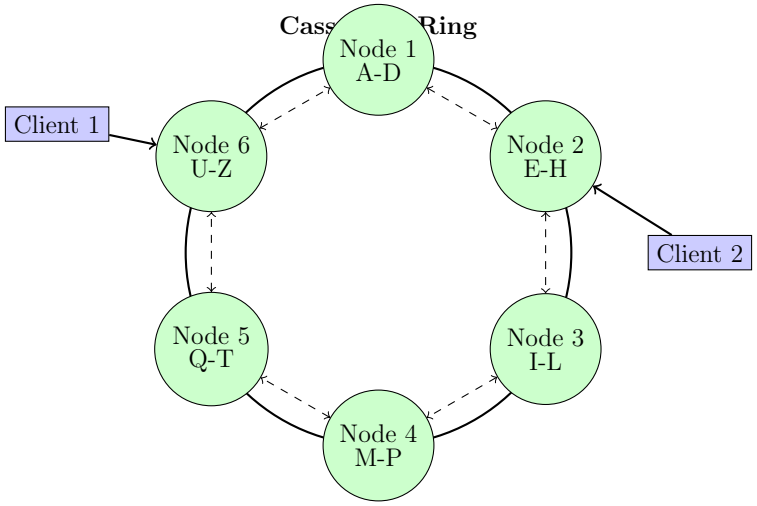


Figure 7.1: Cassandra Ring: Peer-to-Peer Architecture with Gossip Protocol

7.2.1 Key Components

- **Gossip Protocol**: Nodes exchange state information every second
- **Partitioner**: Distributes data across nodes (Murmur3Hash default)
- **Replication Strategy**: SimpleStrategy or NetworkTopologyStrategy
- **Consistency Level**: Tunable per-query (ONE, QUORUM, ALL)
- **Commit Log**: Write-ahead log for durability
- **Memtable**: In-memory write buffer
- **SSTable**: Immutable on-disk sorted string tables
- **Compaction**: Background process merging SSTables

7.3 Complete Install Instructions (Ubuntu 24.04)

7.3.1 Prerequisites

```
# Install Java 11
sudo apt update
sudo apt install openjdk-11-jdk -y
java -version
```

7.3.2 Install Cassandra

```
# Add Cassandra repository
echo "deb https://debian.cassandra.apache.org 50x main"
   | \
  sudo tee /etc/apt/sources.list.d/cassandra.list

# Add repository keys
wget -q -O - https://www.apache.org/dist/cassandra/KEYS
   | sudo apt-key add -

# Install Cassandra
sudo apt update
sudo apt install cassandra -y
```

```
# Start Cassandra
sudo systemctl start cassandra
sudo systemctl enable cassandra

# Check status
sudo systemctl status cassandra
nodetool status
```

7.3.3 Configure Cassandra

Edit `/etc/cassandra/cassandra.yaml`:

```
# Key configurations
cluster_name: 'ProductionCluster'
seeds: "127.0.0.1"
listen_address: localhost
rpc_address: localhost
```

7.4 Working Example: User Activity Tracking

7.4.1 Step 1: Connect with CQL Shell

```
# Connect to Cassandra
cqlsh
```

7.4.2 Step 2: Create Keyspace and Table

```
-- Create keyspace with replication
CREATE KEYSPACE user_activity
WITH replication = {
  'class': 'SimpleStrategy',
  'replication_factor': 3
};

USE user_activity;

-- Create table for user events
```

```
CREATE TABLE events (
    user_id UUID,
    event_time TIMESTAMP,
    event_type TEXT,
    page_url TEXT,
    session_id UUID,
    device_type TEXT,
    PRIMARY KEY ((user_id), event_time)
) WITH CLUSTERING ORDER BY (event_time DESC);

-- Describe table
DESCRIBE TABLE events;
```

7.4.3 Step 3: Insert Data

```
-- Insert sample events
INSERT INTO events (user_id, event_time, event_type,
   page_url, session_id, device_type)
VALUES (uuid(), toTimestamp(now()), 'page_view', '/home'
   , uuid(), 'mobile');

INSERT INTO events (user_id, event_time, event_type,
   page_url, session_id, device_type)
VALUES (uuid(), toTimestamp(now()), 'click', '/products
   /123', uuid(), 'desktop');

-- Insert with TTL (time-to-live)
INSERT INTO events (user_id, event_time, event_type,
   page_url, session_id, device_type)
VALUES (uuid(), toTimestamp(now()), 'purchase', '/
   checkout', uuid(), 'mobile')
USING TTL 86400;  -- Expire after 24 hours
```

7.4.4 Step 4: Query Data

```
-- Query by partition key (user_id)
SELECT * FROM events
WHERE user_id = <some-uuid>
LIMIT 10;

-- Query with time range (requires partition key)
```

```
SELECT * FROM events
WHERE user_id = <some-uuid>
  AND event_time > '2024-01-01'
  AND event_time < '2024-02-01';

-- Count events (avoid on large datasets)
SELECT COUNT(*) FROM events WHERE user_id = <some-uuid>;
```

7.4.5 Step 5: Create Secondary Index

```
-- Create index on event_type (use sparingly)
CREATE INDEX ON events (event_type);

-- Query using index
SELECT * FROM events
WHERE event_type = 'purchase'
ALLOW FILTERING;
```

7.4.6 Step 6: Python Integration

```
# Install driver
pip install cassandra-driver
```

```
from cassandra.cluster import Cluster
from cassandra.query import SimpleStatement
import uuid
from datetime import datetime

# Connect to Cassandra
cluster = Cluster(['127.0.0.1'])
session = cluster.connect('user_activity')

# Insert data
user_id = uuid.uuid4()
session_id = uuid.uuid4()

insert_query = """
    INSERT INTO events (user_id, event_time, event_type,
        page_url, session_id, device_type)
    VALUES (%s, %s, %s, %s, %s, %s)
"""
```

```
session.execute(insert_query, (
    user_id,
    datetime.now(),
    'page_view',
    '/dashboard',
    session_id,
    'desktop'
))

# Query data
query = "SELECT * FROM events WHERE user_id = %s LIMIT
   10"
rows = session.execute(query, (user_id,))

for row in rows:
    print(f"Event: {row.event_type} at {row.event_time}
       - {row.page_url}")

cluster.shutdown()
```

7.4.7 Step 7: Batch Writes

```
-- Batch insert (use for same partition key only)
BEGIN BATCH
    INSERT INTO events (user_id, event_time, event_type,
        page_url, session_id, device_type)
    VALUES (uuid(), toTimestamp(now()), 'login', '/login
       ', uuid(), 'mobile');

    INSERT INTO events (user_id, event_time, event_type,
        page_url, session_id, device_type)
    VALUES (uuid(), toTimestamp(now()), 'view', '/
       profile', uuid(), 'mobile');
APPLY BATCH;
```

7.5 Data Modeling Best Practices

- **Query-First Design**: Model tables around queries, not entities
- **Denormalization**: Duplicate data to avoid joins

- **Partition Key**: Choose keys for even distribution
- **Clustering Columns**: Order data within partitions
- **Avoid ALLOW FILTERING**: Indicates poor data model
- **Time-Series Pattern**: Use time-based clustering

7.6 Performance Tuning

- **Compaction Strategy**: Choose SizeTiered, Leveled, or TimeWindow
- **Read/Write Consistency**: Balance latency vs consistency
- **Replication Factor**: RF=3 for production
- **Partition Size**: Keep under 100MB
- **GC Tuning**: Configure JVM garbage collection
- **Nodetool**: Use for repairs and compaction

7.7 Monitoring

```
# Node status
nodetool status

# Ring topology
nodetool ring

# Check repairs
nodetool repair

# View table stats
nodetool tablestats user_activity.events

# Check compaction
nodetool compactionstats
```

7.8 URL

https://cassandra.apache.org

7.9 Books

- Cassandra: The Definitive Guide, 3rd Edition (O'Reilly, 2020)
- Mastering Apache Cassandra 3.x (Packt)
- The Cassandra Query Language (DataStax documentation)

7.10 Integration

- **Spark**: Spark-Cassandra connector for analytics
- **Kafka**: Stream writes via Kafka Connect
- **Presto/Trino**: SQL queries across Cassandra tables
- **Elasticsearch**: Cassandra + Elasticsearch for search

7.11 Hardening

- **Authentication**: Enable internal authentication
- **Encryption**: Client-to-node and node-to-node TLS
- **Authorization**: Role-based access control
- **JMX**: Secure or disable remote JMX
- **Network**: Restrict ports (9042 CQL, 7000 inter-node)
- **Backups**: Regular snapshots with nodetool snapshot
- **Audit Logging**: Enable audit logs for compliance

Chapter 8

Apache NiFi

8.1 What NiFi Is

Apache NiFi is a powerful, scalable data ingestion and distribution system designed to automate the flow of data between systems. Originally developed by the NSA and open-sourced in 2014, NiFi provides a browser-based UI for designing, controlling, and monitoring dataflows with guaranteed delivery and complete data provenance.

8.1.1 Core Capabilities

- **Web-Based UI**: Drag-and-drop interface for building dataflows
- **Data Provenance**: Track data lineage from source to destination
- **Guaranteed Delivery**: Persistent write-ahead log ensures no data loss
- **Back Pressure**: Automatic throttling when downstream systems slow
- **350+ Processors**: Pre-built connectors for files, databases, APIs, Kafka, S3
- **Site-to-Site Protocol**: Secure node-to-node communication
- **Priority Queues**: Route urgent data first
- **Extensibility**: Custom processors in Java

8.1.2 Use Cases

- Real-time data ingestion from IoT devices
- ETL from databases to data lakes
- API data collection and enrichment
- Log aggregation and routing
- File-based data movement and transformation
- Multi-cloud data synchronization

8.2 Complete Install Instructions (Ubuntu 24.04)

8.2.1 Prerequisites

```
# Install Java 11 or 17
sudo apt update
sudo apt install openjdk-17-jdk -y
java -version
```

8.2.2 Download and Install

```
# Create directory
mkdir -p ~/nifi && cd ~/nifi

# Download NiFi 2.0
wget https://downloads.apache.org/nifi/2.0.0/nifi-2.0.0-
   bin.zip
unzip nifi-2.0.0-bin.zip
cd nifi-2.0.0

# Start NiFi
./bin/nifi.sh start

# Access Web UI (wait 1-2 minutes)
# https://localhost:8443/nifi

# Get auto-generated credentials
grep "Generated Username" logs/nifi-app.log
```

8.3 Working Example: CSV to Kafka Pipeline

Build a flow that monitors a directory for CSV files and publishes to Kafka.

8.3.1 Flow Design

1. **GetFile**: Monitor /tmp/nifi-input for CSV files
2. **ConvertRecord**: Convert CSV to JSON
3. **PublishKafka**: Send to Kafka topic
4. **LogAttribute**: Log failures

8.3.2 Test Data

```
# Create input directory
mkdir -p /tmp/nifi-input

# Create sample CSV
cat > /tmp/nifi-input/sample.csv <<EOF
id,name,amount,timestamp
1,Product A,299.99,2024-01-15T10:00:00
2,Product B,149.50,2024-01-15T11:00:00
EOF
```

8.4 NiFi Expression Language

```
# File attributes
${filename}
${file.size}

# String manipulation
${filename:toUpper()}
${timestamp:format('yyyy-MM-dd')}

# Conditional
${fileSize:gt(1000):ifElse('large','small')}
```

8.5 Kubernetes Deployment

```
helm repo add cetic https://cetic.github.io/helm-charts
helm install nifi cetic/nifi --set replicaCount=3
```

More info: https://artifacthub.io/packages/helm/cetic/nifi

8.6 Books

- Apache NiFi: Data Flow Automation (Packt)
- Learning Apache NiFi (Packt)

8.7 URL

https://nifi.apache.org

8.8 Hardening

- **HTTPS**: Enabled by default in 2.0
- **Authentication**: LDAP, Kerberos, or OIDC
- **Authorization**: Fine-grained access policies
- **Site-to-Site mTLS**: Secure cluster communication
- **Network**: Firewall ports 8443, 10443

Chapter 9

Apache Airflow

9.1 What Airflow Is

Apache Airflow is a platform to programmatically author, schedule, and monitor workflows. Created by Airbnb in 2014 and donated to Apache in 2016, Airflow uses Python to define Directed Acyclic Graphs (DAGs) of tasks, making it the industry standard for orchestrating complex data pipelines.

9.1.1 Core Capabilities

- **DAG-Based Workflows**: Define dependencies between tasks as Python code
- **Rich Scheduling**: Cron expressions, intervals, dataset-driven triggers
- **100+ Operators**: Pre-built tasks for Spark, Hive, Kafka, S3, Kubernetes
- **Web UI**: Monitor DAG runs, task states, logs, and metrics
- **Backfilling**: Re-run historical data processing
- **Dynamic Pipelines**: Generate DAGs programmatically
- **Extensibility**: Custom operators, hooks, sensors
- **RBAC**: Role-based access control for teams

9.1.2 Use Cases

- Daily ETL pipelines orchestration
- Machine learning model training workflows
- Data quality validation and testing
- Multi-step data transformations
- Cloud resource provisioning and teardown
- Report generation and distribution

9.2 Complete Install Instructions (Ubuntu 24.04)

9.2.1 Prerequisites

```
# Install Python 3.8+
sudo apt update
sudo apt install python3 python3-pip python3-venv -y
python3 --version
```

9.2.2 Install Airflow

```
# Create virtual environment
python3 -m venv ~/airflow-venv
source ~/airflow-venv/bin/activate

# Set Airflow home
export AIRFLOW_HOME=~/airflow
mkdir -p $AIRFLOW_HOME

# Install Airflow 2.8
AIRFLOW_VERSION=2.8.0
PYTHON_VERSION="$(python3 --version | cut -d " " -f 2 |
   cut -d "." -f 1-2)"
CONSTRAINT_URL="https://raw.githubusercontent.com/apache
   /airflow/constraints-${AIRFLOW_VERSION}/constraints-$
   {PYTHON_VERSION}.txt"
```

```
pip install "apache-airflow==${AIRFLOW_VERSION}" --
   constraint "${CONSTRAINT_URL}"

# Initialize database
airflow db init

# Create admin user
airflow users create \
  --username admin \
  --firstname Admin \
  --lastname User \
  --role Admin \
  --email admin@example.com \
  --password admin123
```

9.2.3 Start Airflow

```
# Start webserver (terminal 1)
airflow webserver --port 8080

# Start scheduler (terminal 2 - new terminal)
source ~/airflow-venv/bin/activate
export AIRFLOW_HOME=~/airflow
airflow scheduler

# Access Web UI
# http://localhost:8080
# Login: admin / admin123
```

9.3 Working Example: Daily Sales ETL Pipeline

Create `$AIRFLOW_HOME/dags/sales_etl.py`:

```
from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator
from airflow.providers.postgres.operators.postgres
   import PostgresOperator
import pandas as pd
```

```
# Default arguments for all tasks
default_args = {
    'owner': 'data-team',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'email_on_failure': True,
    'email': ['data@example.com'],
    'retries': 2,
    'retry_delay': timedelta(minutes=5),
}

# Define DAG
dag = DAG(
    'sales_etl_pipeline',
    default_args=default_args,
    description='Daily sales data ETL',
    schedule='0 2 * * *',  # Run daily at 2 AM
    catchup=False,
    tags=['sales', 'etl'],
)

# Task 1: Extract data from API
def extract_sales_data(**context):
    # Simulate API call
    data = {
        'sale_id': [1, 2, 3],
        'amount': [100, 200, 150],
        'date': ['2024-01-15'] * 3
    }
    df = pd.DataFrame(data)
    df.to_csv('/tmp/sales_raw.csv', index=False)
    print(f"Extracted {len(df)} sales records")

extract_task = PythonOperator(
    task_id='extract_sales',
    python_callable=extract_sales_data,
    dag=dag,
)

# Task 2: Transform data
def transform_sales_data(**context):
    df = pd.read_csv('/tmp/sales_raw.csv')
    # Add 10% tax
```

```
    df['amount_with_tax'] = df['amount'] * 1.1
    df.to_csv('/tmp/sales_transformed.csv', index=False)
    print(f"Transformed {len(df)} records")

transform_task = PythonOperator(
    task_id='transform_sales',
    python_callable=transform_sales_data,
    dag=dag,
)

# Task 3: Load to warehouse (simulated)
load_task = BashOperator(
    task_id='load_to_warehouse',
    bash_command='echo "Loading data to warehouse..." &&
        sleep 2',
    dag=dag,
)

# Task 4: Send notification
notify_task = BashOperator(
    task_id='send_notification',
    bash_command='echo "ETL pipeline completed
       successfully"',
    dag=dag,
)

# Define task dependencies
extract_task >> transform_task >> load_task >>
   notify_task
```

9.3.1 Trigger DAG

```
# Manually trigger DAG
airflow dags trigger sales_etl_pipeline

# List all DAGs
airflow dags list

# Test specific task
airflow tasks test sales_etl_pipeline extract_sales
   2024-01-15
```

9.4 Advanced DAG Example: Spark Job Orchestration

```
from airflow.providers.apache.spark.operators.
    spark_submit import SparkSubmitOperator
from airflow.providers.amazon.aws.sensors.s3 import
    S3KeySensor

# Wait for S3 file
wait_for_file = S3KeySensor(
    task_id='wait_for_data',
    bucket_name='my-bucket',
    bucket_key='data/input/*.parquet',
    aws_conn_id='aws_default',
    timeout=3600,
    poke_interval=60,
    dag=dag,
)

# Run Spark job
spark_job = SparkSubmitOperator(
    task_id='process_with_spark',
    application='/path/to/spark_job.py',
    conn_id='spark_default',
    conf={
        'spark.executor.memory': '4g',
        'spark.executor.cores': '2'
    },
    dag=dag,
)

wait_for_file >> spark_job
```

9.5 Kubernetes Deployment

```
# Add Helm repository
helm repo add apache-airflow https://airflow.apache.org
helm repo update

# Install Airflow on Kubernetes
```

```
helm install airflow apache-airflow/airflow \
  --namespace airflow \
  --create-namespace \
  --set executor=KubernetesExecutor \
  --set webserver.replicas=2 \
  --set scheduler.replicas=2 \
  --set postgresql.enabled=true

# Access Web UI
kubectl port-forward svc/airflow-webserver 8080:8080 -n
   airflow
```

Official Helm chart: https://airflow.apache.org/docs/helm-chart/

9.6 Airflow Executors

- **SequentialExecutor**: Single task at a time (development only)
- **LocalExecutor**: Multiple tasks on single machine
- **CeleryExecutor**: Distributed task execution via Celery
- **KubernetesExecutor**: Each task in separate Kubernetes pod
- **CeleryKubernetesExecutor**: Hybrid approach

9.7 Best Practices

- **Idempotent Tasks**: Tasks should produce same result when re-run
- **XCom Sparingly**: Don't pass large data between tasks
- **Task Atomicity**: Each task should do one thing well
- **Connection Pooling**: Reuse database connections
- **DAG Versioning**: Store DAGs in Git
- **Test DAGs**: Use pytest for unit testing

9.8 Monitoring

- **Web UI**: Real-time DAG and task monitoring
- **Metrics**: Export to Prometheus/StatsD
- **Logs**: Centralize with ELK or Cloud Logging
- **Alerts**: Email, Slack, PagerDuty integrations
- **SLAs**: Set task-level SLA monitoring

9.9 Books

- Data Pipelines with Apache Airflow (Manning, 2021)
- Apache Airflow: The Hands-On Guide (Packt, 2023)

9.10 URL

https://airflow.apache.org

9.11 Integration

Airflow integrates with:

- **Spark**: SparkSubmitOperator for Spark jobs
- **Kubernetes**: KubernetesPodOperator
- **AWS/GCP/Azure**: Cloud-specific operators
- **Databases**: Postgres, MySQL, Snowflake, BigQuery
- **Kafka**: Trigger DAGs from Kafka topics
- **dbt**: Run dbt models as Airflow tasks

9.12 Hardening

- **Authentication**: LDAP, OAuth, or OIDC
- **RBAC**: Fine-grained permissions per DAG
- **Database**: Secure PostgreSQL metadata database
- **Secrets Backend**: Use Vault or cloud secret managers
- **TLS**: Enable HTTPS for webserver
- **Network**: Firewall webserver port (8080)
- **Fernet Key**: Rotate encryption keys regularly

Chapter 10

Apache Hop

10.1 What Hop Is

Apache Hop (Hop Orchestration Platform) is a metadata-driven, visual data orchestration and engineering platform designed for building, running, and managing data pipelines and workflows. Hop originated from the Pentaho Data Integration (Kettle/PDI) project and became an Apache top-level project in December 2021.

10.1.1 Key Characteristics

- **Visual Development**: Drag-and-drop interface for building data pipelines without extensive coding
- **Metadata-Driven**: Separates pipeline logic from environment-specific configurations
- **Dual Abstraction**: Pipelines (data flow) and Workflows (orchestration)
- **Multi-Engine Support**: Execute on Spark, Flink, Google Dataflow, or native Hop engine
- **Version Control Friendly**: Pipelines stored as XML files, ideal for Git integration
- **Extensible**: 350+ transforms and actions covering data integration, quality, and orchestration

10.1.2 Core Concepts

Pipelines are directed graphs of transforms (operations) connected by hops (data flows). They execute in parallel where possible and handle streaming data transformations.

Workflows orchestrate pipelines and other workflows using actions. They run sequentially and provide branching, looping, and error handling.

Environments define configurations for different deployment targets (dev, test, prod) without changing pipeline code.

Projects organize related pipelines, workflows, metadata, and configurations into logical units.

10.1.3 Use Cases

- ETL/ELT processing from diverse sources (databases, files, APIs, streams)
- Data quality validation and cleansing
- Real-time stream processing integration with Kafka and Flink
- Batch data migration and synchronization
- Orchestration of complex multi-stage data workflows
- Integration testing and data pipeline CI/CD

10.2 Install Instructions

10.2.1 Prerequisites

```
# Java 11 or later required
sudo apt update
sudo apt install default-jdk unzip -y

java -version  # Should show version 11+
```

10.2.2 Install Apache Hop

```
# Download Hop client (version 2.9.0)
cd /opt
sudo wget https://dlcdn.apache.org/hop/2.9.0/apache-hop-
   client-2.9.0.zip

# Extract
sudo unzip apache-hop-client-2.9.0.zip
sudo mv hop /opt/hop
sudo chown -R $USER:$USER /opt/hop

# Add to PATH
echo 'export HOP_HOME=/opt/hop' >> ~/.bashrc
echo 'export PATH=$PATH:$HOP_HOME' >> ~/.bashrc
source ~/.bashrc

# Make scripts executable
chmod +x /opt/hop/*.sh

# Launch Hop GUI
cd /opt/hop
./hop-gui.sh
```

10.2.3 Verify Installation

The Hop GUI should launch. You'll see:

- Menu bar with File, Edit, Pipeline, Workflow options
- Left panel showing Projects and Environments
- Main canvas for designing pipelines
- Transform/Action palette on the right

10.3 Working Example: CSV ETL Pipeline

We'll build a pipeline that reads CSV sales data, transforms it, and writes to PostgreSQL.

10.3.1 Setup Test Database

```
# Install PostgreSQL
sudo apt install postgresql postgresql-contrib -y
sudo systemctl start postgresql

# Create database and table
sudo -u postgres psql << EOF
CREATE DATABASE hopdb;
\c hopdb
CREATE TABLE sales_clean (
    sale_id INTEGER PRIMARY KEY,
    product_name VARCHAR(100),
    amount DECIMAL(10,2),
    sale_date DATE,
    region VARCHAR(50)
);
EOF
```

10.3.2 Create Sample CSV Data

```
cat > /tmp/sales_raw.csv << 'EOF'
sale_id,product,amount,date,region
1,Laptop,1299.99,2024-01-15,North
2,Mouse,29.99,2024-01-16,South
3,Keyboard,79.99,2024-01-16,East
4,Monitor,349.99,2024-01-17,West
5,Laptop,1299.99,2024-01-18,North
EOF
```

10.3.3 Build Pipeline in Hop GUI

Step 1: Create New Pipeline

1. Launch Hop GUI: `./hop-gui.sh`
2. File → New → Pipeline
3. Save as `sales_etl_pipeline.hpl`

Step 2: Add CSV Input Transform

1. From transform palette, search "CSV file input"

2. Drag onto canvas

3. Double-click to configure:

    - Filename: `/tmp/sales_raw.csv`
    - Delimiter: comma
    - Header present: Yes
    - Get Fields button to auto-detect columns

Step 3: Add Filter Rows Transform

1. Search "Filter rows", drag onto canvas

2. Connect CSV input to Filter (hold Shift + drag)

3. Configure:

    - Condition: `amount > 50`
    - Send 'true' data to: next transform
    - Send 'false' data to: (null - discard)

Step 4: Add Calculator Transform

1. Add "Calculator" transform

2. Configure:

    - New field: `product_name`
    - Calculation: Upper case of `product`

Step 5: Add Table Output Transform

1. Add "Table output" transform

2. First create database connection:

    - Metadata → Relational Database Connection → New
    - Name: `PostgreSQL-Local`
    - Connection Type: PostgreSQL
    - Host: localhost, Port: 5432
    - Database: hopdb
    - Username: postgres

- Test connection

3. Configure table output:
   - Connection: PostgreSQL-Local
   - Target table: `sales_clean`
   - Map fields: sale_id → sale_id, product_name → product_name, etc.

10.3.4 Pipeline Execution

In Hop GUI:

1. Click the "Run" button (play icon) in toolbar
2. Select "Local" run configuration
3. Click "Launch"
4. Monitor execution in bottom panel - should show green checks
5. Check metrics: rows read, filtered, written

10.3.5 Verify Results

```
sudo -u postgres psql -d hopdb -c "SELECT * FROM
   sales_clean;"
```

Expected output (rows with amount ¿ 50):

```
 sale_id | product_name | amount   | sale_date   | region
---------+--------------+----------+-------------+--------
       3 | KEYBOARD     |    79.99 | 2024-01-16 | East
       4 | MONITOR      |   349.99 | 2024-01-17 | West
       1 | LAPTOP       | 1299.99 | 2024-01-15 | North
       5 | LAPTOP       | 1299.99 | 2024-01-18 | North
```

10.4 Command-Line Execution

Hop pipelines can run without GUI for production:

```
# Run pipeline from command line
/opt/hop/hop-run.sh \
  --file=/path/to/sales_etl_pipeline.hpl \
  --runconfig=Local \
  --level=Basic \
  --parameters=INPUT_FILE=/tmp/sales_raw.csv

# Run with environment override
/opt/hop/hop-run.sh \
  --file=/path/to/sales_etl_pipeline.hpl \
  --environment=Production \
  --runconfig=Spark
```

10.5 Workflow Example

Workflows orchestrate multiple pipelines:

```
# Create workflow in GUI: File     New     Workflow
# Add actions:
# 1. Start action
# 2. Pipeline action     Run sales_etl_pipeline.hpl
# 3. SQL action     Run aggregation query
# 4. Mail action     Send completion notification
# 5. Success action (end)

# Connect actions with hops
# Configure conditions (success/failure paths)
# Save as daily_sales_workflow.hwf
```

10.6 Integration with Execution Engines

10.6.1 Apache Spark Execution

```
# Add Spark run configuration in Hop GUI
# Metadata     Pipeline Run Configuration     New
# Name: Spark-Cluster
# Engine: Spark
# Master: spark://master:7077

# Pipeline runs on Spark cluster
/opt/hop/hop-run.sh \
  --file=sales_etl_pipeline.hpl \
  --runconfig=Spark-Cluster
```

10.6.2 Apache Flink Execution

```
# Configure Flink run configuration
# Engine: Flink
# Flink Master: localhost:8081

# Suitable for streaming pipelines
# with Kafka input and continuous processing
```

10.7 Metadata and Environments

10.7.1 Environment Configuration

Create environment file `config/dev.json`:

```
{
  "name": "Development",
  "description": "Dev environment",
  "variables": {
    "DB_HOST": "localhost",
    "DB_PORT": "5432",
    "DB_NAME": "hopdb_dev",
    "INPUT_PATH": "/tmp/data"
  }
}
```

Production environment `config/prod.json`:

```
{
```

```
  "name": "Production",
  "description": "Prod environment",
  "variables": {
    "DB_HOST": "prod-db.company.com",
    "DB_PORT": "5432",
    "DB_NAME": "hopdb_prod",
    "INPUT_PATH": "/data/prod/input"
  }
}
```

Reference in pipelines using: `${DB_HOST}`

10.8 CI/CD Integration

10.8.1 Version Control

```
# Initialize Git repository
cd /opt/hop/my-hop-project
git init
git add *.hpl *.hwf config/
git commit -m "Initial pipeline commit"
git push origin main
```

10.8.2 Automated Testing

```
# Unit test pipeline with hop-run
/opt/hop/hop-run.sh \
  --file=tests/test_sales_pipeline.hpl \
  --runconfig=Test \
  --parameters=MODE=test

# Check exit code
if [ $? -eq 0 ]; then
    echo "Pipeline test passed"
else
    echo "Pipeline test failed"
    exit 1
fi
```

10.8.3 Jenkins Integration

```
pipeline {
    agent any
    stages {
        stage('Run Hop Pipeline') {
            steps {
                sh '''
                    /opt/hop/hop-run.sh \
                      --file=${WORKSPACE}/pipelines/
                         sales_etl.hpl \
                      --environment=Production \
                      --runconfig=Spark
                '''
            }
        }
    }
}
```

10.9 Monitoring and Logging

10.9.1 Enable Detailed Logging

Edit `config/hop-log.xml`:

```
<configuration>
  <appender name="FILE" class="ch.qos.logback.core.
     FileAppender">
    <file>/var/log/hop/hop.log</file>
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %
         logger{36} - %msg%n</pattern>
    </encoder>
  </appender>
  <root level="INFO">
    <appender-ref ref="FILE" />
  </root>
</configuration>
```

10.9.2 Performance Metrics

```
# Run with detailed metrics
/opt/hop/hop-run.sh \
  --file=sales_pipeline.hpl \
  --level=Detailed \
  --metrics=true
```

Metrics include:

- Rows read/written per transform
- Execution time per transform
- Memory usage
- Error counts

10.10 Best Practices

1. **Use Projects**: Organize related pipelines, workflows, and metadata into projects for better management
2. **Parameterize Everything**: Use environment variables for paths, connections, and configuration
3. **Modular Design**: Break complex ETL into smaller, reusable pipelines
4. **Error Handling**: Add error handling transforms and configure failure paths in workflows
5. **Version Control**: Store all .hpl and .hwf files in Git with meaningful commit messages
6. **Testing**: Create test workflows with sample data before production deployment
7. **Documentation**: Use pipeline/workflow descriptions and transform notes
8. **Performance**: Use streaming transforms for large datasets, minimize sorts and aggregations

10.11 Common Transforms

Transform	Purpose
CSV file input	Read CSV files with configurable delimiter, encoding
Table input	Execute SQL SELECT and read results
Table output	Write to database tables with insert/update options
Filter rows	Conditional row filtering based on expressions
Calculator	Mathematical and string calculations
Sort rows	Sort data by one or more fields
Group by	Aggregate operations (SUM, AVG, COUNT)
Join rows	Merge streams using join conditions
JSON input/output	Parse and generate JSON data
Kafka Consumer/Producer	Real-time stream processing

10.12 URL

Official Apache Hop website and documentation:
https://hop.apache.org
Documentation: https://hop.apache.org/manual/latest/

10.13 Books and Learning Resources

- **Apache Hop User Guide** (official documentation) - comprehensive coverage
- **Hop Technical Documentation** - available at https://hop.apache.org/tech-manual/latest/
- **Video Tutorials** - Apache Hop YouTube channel
- **Community Blog Posts** - https://hop.apache.org/community/

10.14 Hardening and Security

1. **Encrypt Database Passwords**: Use Hop's password encryption utility

```
/opt/hop/hop-encrypt.sh -hop mypassword
# Use encrypted value in metadata
```

2. **Secure Hop Server**: When running Hop Server (web-based execution):
   - Enable HTTPS with TLS certificates
   - Use authentication (username/password or LDAP)
   - Run in isolated network subnet
   - Configure firewall to restrict access
3. **File System Permissions**: Restrict access to pipeline files

```
chmod 640 /opt/hop/config/*.json
chown hop:hop /opt/hop/config/
```

4. **Environment Variables**: Never hardcode credentials
   - Use environment-specific configurations
   - Store secrets in external vault (HashiCorp Vault, AWS Secrets Manager)
   - Reference via variables: `${DB_PASSWORD}`
5. **Audit Logging**: Enable comprehensive logging
   - Log all pipeline executions
   - Track who executed what and when
   - Monitor for failed authentications
   - Retain logs for compliance requirements
6. **Network Security**:
   - Use VPN for remote Hop Server access
   - Enable SSL/TLS for database connections
   - Implement network segmentation
7. **Regular Updates**: Keep Hop updated to latest stable version for security patches

Chapter 11

Apache Hadoop

11.1 What Hadoop Is

Apache Hadoop is the foundational distributed storage and processing framework that launched the big data era. Created in 2006 by Doug Cutting and Mike Cafarella (inspired by Google's GFS and MapReduce papers), Hadoop enables processing of massive datasets across clusters of commodity hardware.

While newer technologies have surpassed Hadoop for many use cases, it remains widely deployed for cost-effective large-scale storage and serves as the foundation for many modern data platforms.

11.1.1 Core Components

HDFS (Hadoop Distributed File System) is a distributed, fault-tolerant filesystem that:

- Stores data across multiple nodes with replication (default 3x)
- Optimized for large files (GB to TB range)
- Write-once, read-many access pattern
- Namenode (metadata) + Datanodes (storage) architecture
- Supports files up to petabyte scale

YARN (Yet Another Resource Negotiator) is the cluster resource manager that:

- Schedules jobs across cluster nodes

- Manages memory and CPU allocation
- Supports multiple processing frameworks (MapReduce, Spark, Flink)
- ResourceManager (global) + NodeManagers (per-node) architecture
- Enables multi-tenancy with resource queues

MapReduce is the original batch processing framework that:

- Divides work into Map (transform) and Reduce (aggregate) phases
- Processes data where it's stored (data locality)
- Automatic fault tolerance and retry
- Java-based with streaming support for other languages
- Largely superseded by Spark for performance reasons

11.1.2 Modern Relevance

Despite newer alternatives, Hadoop remains valuable for:

- Cost-effective archival storage (HDFS on commodity hardware)
- Foundation for Hive, HBase, and other ecosystem tools
- Object storage compatibility (S3, Azure Blob, GCS)
- Enterprise data lakes with existing Hadoop investments
- Batch processing workloads not requiring real-time speed

11.2 Install Instructions (Pseudo-Distributed)

Pseudo-distributed mode runs Hadoop on a single machine with all daemons running as separate processes, simulating a cluster.

11.2.1 Prerequisites

```
# Install Java 8 or 11
sudo apt update
sudo apt install default-jdk ssh pdsh -y

# Verify Java installation
java -version

# Configure passwordless SSH (required for Hadoop)
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys

# Test SSH
ssh localhost
exit
```

11.2.2 Download and Install Hadoop

```
# Download Hadoop 3.4.0
cd /opt
sudo wget https://dlcdn.apache.org/hadoop/common/hadoop
   -3.4.0/hadoop-3.4.0.tar.gz
sudo tar -xzf hadoop-3.4.0.tar.gz
sudo mv hadoop-3.4.0 hadoop
sudo chown -R $USER:$USER /opt/hadoop

# Set environment variables
echo 'export HADOOP_HOME=/opt/hadoop' >> ~/.bashrc
echo 'export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/
   sbin' >> ~/.bashrc
echo 'export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop' >>
    ~/.bashrc
echo 'export JAVA_HOME=/usr/lib/jvm/default-java' >> ~/.
   bashrc
source ~/.bashrc

# Configure JAVA_HOME in Hadoop
echo "export JAVA_HOME=/usr/lib/jvm/default-java" >>
   $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

11.2.3 Configure Hadoop

Edit `/opt/hadoop/etc/hadoop/core-site.xml`:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://localhost:9000</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/opt/hadoop/tmp</value>
    </property>
</configuration>
```

Edit `/opt/hadoop/etc/hadoop/hdfs-site.xml`:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:///opt/hadoop/hdfs/namenode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:///opt/hadoop/hdfs/datanode</value>
    </property>
</configuration>
```

Edit `/opt/hadoop/etc/hadoop/mapred-site.xml`:

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>mapreduce.application.classpath</name>
        <value>$HADOOP_HOME/share/hadoop/mapreduce/*:$
           HADOOP_HOME/share/hadoop/mapreduce/lib/*</
           value>
    </property>
</configuration>
```

Edit `/opt/hadoop/etc/hadoop/yarn-site.xml`:

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.env-whitelist</name>
        <value>JAVA_HOME,HADOOP_COMMON_HOME,
           HADOOP_HDFS_HOME,HADOOP_CONF_DIR,
           CLASSPATH_PREPEND_DISTCACHE,HADOOP_YARN_HOME,
           HADOOP_HOME,PATH,LANG,TZ,HADOOP_MAPRED_HOME</
           value>
    </property>
</configuration>
```

11.2.4 Start Hadoop

```
# Create directories
mkdir -p /opt/hadoop/hdfs/namenode
mkdir -p /opt/hadoop/hdfs/datanode
mkdir -p /opt/hadoop/tmp

# Format namenode (only first time)
hdfs namenode -format

# Start HDFS
start-dfs.sh

# Start YARN
start-yarn.sh

# Verify processes are running
jps
# Should show: NameNode, DataNode, SecondaryNameNode,
   ResourceManager, NodeManager
```

11.2.5 Access Web UIs

- NameNode UI: http://localhost:9870
- ResourceManager UI: http://localhost:8088
- DataNode UI: http://localhost:9864

11.3 Working Example: Word Count MapReduce

11.3.1 HDFS Operations

```
# Create input directory in HDFS
hdfs dfs -mkdir -p /user/$USER/input

# Create sample text file
cat > sample.txt << 'EOF'
Apache Hadoop is a framework for distributed storage and
    processing
Hadoop enables big data processing across clusters of
   computers
MapReduce is the original Hadoop processing paradigm
HDFS provides distributed fault-tolerant storage for
   Hadoop
EOF

# Upload to HDFS
hdfs dfs -put sample.txt /user/$USER/input/

# List files in HDFS
hdfs dfs -ls /user/$USER/input/

# View file contents from HDFS
hdfs dfs -cat /user/$USER/input/sample.txt

# Check HDFS storage usage
hdfs dfs -du -h /user/$USER/
```

11.3.2 MapReduce Word Count Example

Hadoop includes sample MapReduce programs. Let's run the word count example:

```
# Run word count MapReduce job
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-
   mapreduce-examples-3.4.0.jar \
  wordcount \
  /user/$USER/input \
  /user/$USER/output

# View the results
hdfs dfs -cat /user/$USER/output/part-r-00000
```

Expected output:

```
Apache    1
HDFS      1
Hadoop    4
MapReduce 1
across    1
big       1
...
```

11.3.3 Custom MapReduce Job (Python Streaming)

Create a Python mapper (`mapper.py`):

```
#!/usr/bin/env python3
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print(f"{word.lower()}\t1")
```

Create a Python reducer (`reducer.py`):

```
#!/usr/bin/env python3
import sys

current_word = None
current_count = 0
```

```
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    count = int(count)

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

if current_word:
    print(f"{current_word}\t{current_count}")
```

Make them executable and run:

```
chmod +x mapper.py reducer.py

# Upload to HDFS
hdfs dfs -put mapper.py reducer.py /user/$USER/

# Run streaming job
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-
   streaming-3.4.0.jar \
  -input /user/$USER/input \
  -output /user/$USER/output_python \
  -mapper /user/$USER/mapper.py \
  -reducer /user/$USER/reducer.py

# View results
hdfs dfs -cat /user/$USER/output_python/part-00000
```

11.4 HDFS Commands Reference

Command	Description
hdfs dfs -ls ¡path¿	List directory contents
hdfs dfs -mkdir ¡path¿	Create directory
hdfs dfs -put ¡local¿ ¡hdfs¿	Upload file to HDFS
hdfs dfs -get ¡hdfs¿ ¡local¿	Download file from HDFS
hdfs dfs -cat ¡path¿	Display file contents
hdfs dfs -rm ¡path¿	Delete file
hdfs dfs -rm -r ¡path¿	Delete directory recursively
hdfs dfs -du -h ¡path¿	Show disk usage
hdfs dfs -chmod ¡perm¿ ¡path¿	Change permissions
hdfs dfs -chown ¡owner¿ ¡path¿	Change ownership

11.5 Integration with Modern Tools

11.5.1 Running Spark on YARN

```
# Spark can use YARN as resource manager
spark-submit --master yarn --deploy-mode cluster my_job.
   py
```

11.5.2 Hive with HDFS Storage

```
-- Hive stores data in HDFS
CREATE TABLE sales (
    id INT,
    product STRING,
    amount DECIMAL(10,2)
)
STORED AS PARQUET
LOCATION '/user/hive/warehouse/sales';
```

11.5.3 HBase on HDFS

HBase uses HDFS as its underlying storage layer for fault-tolerant, distributed data storage.

11.6 Monitoring and Management

11.6.1 NameNode Health

```
# Check HDFS health
hdfs dfsadmin -report

# Safe mode operations (prevents writes)
hdfs dfsadmin -safemode get
hdfs dfsadmin -safemode enter
hdfs dfsadmin -safemode leave

# Check filesystem
hdfs fsck /
```

11.6.2 YARN Application Management

```
# List running applications
yarn application -list

# Kill an application
yarn application -kill <application_id>

# View application logs
yarn logs -applicationId <application_id>

# Check node status
yarn node -list
```

11.7 Performance Tuning

1. **HDFS Block Size**: Default is 128MB. Larger blocks (256MB) reduce NameNode memory
2. **Replication Factor**: Default is 3. Reduce for non-critical data to save storage
3. **YARN Memory**: Configure `yarn.nodemanager.resource.memory-mb` based on available RAM
4. **MapReduce Tuning**: Adjust mapper/reducer memory (`mapreduce.map.memory.mb`)
5. **Compression**: Enable intermediate compression to reduce network I/O

11.8 URL

Official Apache Hadoop website and documentation:
https://hadoop.apache.org
Documentation: https://hadoop.apache.org/docs/current/

11.9 Books and Learning Resources

- **Hadoop: The Definitive Guide** (O'Reilly) - comprehensive coverage of HDFS, MapReduce, and YARN
- **Hadoop Application Architectures** (O'Reilly) - real-world patterns and best practices
- **Hadoop Security** (O'Reilly) - security hardening and Kerberos integration
- **Hadoop Operations** (O'Reilly) - cluster management and monitoring

11.10 Hardening and Security

1. **Kerberos Authentication**: Enable strong authentication across all Hadoop services

```
# Configure Kerberos in core-site.xml
<property>
    <name>hadoop.security.authentication</name>
    <value>kerberos</value>
</property>
<property>
    <name>hadoop.security.authorization</name>
    <value>true</value>
</property>
```

2. **TLS/SSL Encryption**: Enable encrypted communication for all daemons
   - Configure SSL for HDFS, YARN, and MapReduce web UIs
   - Use `hadoop.ssl.enabled=true`
   - Distribute truststore and keystore to all nodes

3. **HDFS Permissions**: Implement proper file and directory permissions

```
# Set permissions on HDFS directories
hdfs dfs -chmod 750 /user/data
hdfs dfs -chown hadoop:hadoop /user/data

# Enable HDFS ACLs for fine-grained access
hdfs dfs -setfacl -m user:alice:rwx /user/data
```

4. **YARN Resource Queues**: Implement multi-tenancy with resource isolation
   - Configure capacity scheduler or fair scheduler
   - Set queue ACLs to restrict job submission
   - Enforce resource limits per user/queue

5. **Audit Logging**: Enable comprehensive audit logs
   - HDFS audit logs: track all file access
   - YARN audit logs: track job submissions

- Regular log rotation and archival

6. **Network Security**:
   - Firewall rules to restrict access to Hadoop ports
   - VPN for remote administrative access
   - Network segmentation for data nodes
7. **Data Encryption at Rest**:
   - HDFS Transparent Data Encryption (TDE) for sensitive data zones
   - Key management with Hadoop KMS or external systems (HashiCorp Vault)
8. **Regular Security Updates**:
   - Apply security patches promptly
   - Monitor CVE announcements for Hadoop
   - Test updates in staging before production deployment

11.10.1 Security Best Practices

- Use dedicated service accounts (not root) for all Hadoop daemons
- Implement least-privilege access principle
- Regular security audits and penetration testing
- Monitor for suspicious activity (unusual job submissions, data access patterns)
- Implement data classification and apply appropriate controls
- Backup NameNode metadata regularly
- Test disaster recovery procedures

Chapter 12

Apache Hive

12.1 What Hive Is

Apache Hive is a data warehouse system built on top of Hadoop that enables SQL-based querying and analysis of large datasets stored in distributed storage. Created at Facebook in 2008 and open-sourced in 2010, Hive translates SQL queries (HiveQL) into MapReduce, Tez, or Spark jobs.

12.1.1 Key Capabilities

- **SQL Interface**: Write standard SQL queries instead of MapReduce code
- **Schema-on-Read**: Apply schema when reading data, not when writing
- **Multiple Storage Formats**: ORC, Parquet, Avro, JSON, CSV
- **Table Formats**: Traditional Hive tables, Apache Iceberg, Delta Lake
- **ACID Transactions**: Full ACID support for update/delete operations
- **Partitioning and Bucketing**: Optimize query performance through data organization

12.1.2 Hive Metastore

The Hive Metastore is a critical component that stores metadata about tables, partitions, columns, and storage locations. It serves as the central schema repository for the entire Hadoop ecosystem:

- **Used by Spark**: SparkSQL uses Hive Metastore for table definitions
- **Used by Presto/Trino**: Query engines connect to Hive Metastore
- **Used by Impala**: Cloudera's SQL engine reads from Hive Metastore
- **Used by Doris**: Can integrate with Hive for external tables

The metastore typically uses a relational database backend (MySQL, PostgreSQL) to store metadata.

12.1.3 Execution Engines

Hive supports multiple execution engines:

- **MapReduce**: Original engine (slow, legacy)
- **Tez**: Directed Acyclic Graph (DAG) engine for better performance
- **Spark**: Use Spark as execution engine for fastest queries

12.1.4 Use Cases

- Batch ETL processing on large datasets
- Data warehouse queries on historical data
- Log analysis and aggregation
- Ad-hoc analytics for data scientists
- Reporting and business intelligence

12.2 Install Instructions

12.2.1 Prerequisites

Hive requires Hadoop to be installed and running (see Chapter 11).

```
# Verify Hadoop is running
jps | grep -E "NameNode|DataNode"

# Install MySQL for metastore (recommended over Derby)
sudo apt install mysql-server -y
sudo systemctl start mysql
```

12.2.2 Create Metastore Database

```
# Create Hive metastore database
sudo mysql << EOF
CREATE DATABASE metastore;
CREATE USER 'hive'@'localhost' IDENTIFIED BY '
   hivepassword';
GRANT ALL PRIVILEGES ON metastore.* TO 'hive'@'localhost
   ';
FLUSH PRIVILEGES;
EOF
```

12.2.3 Download and Install Hive

```
# Download Hive 4.0.0
cd /opt
sudo wget https://dlcdn.apache.org/hive/hive-4.0.0/
   apache-hive-4.0.0-bin.tar.gz
sudo tar -xzf apache-hive-4.0.0-bin.tar.gz
sudo mv apache-hive-4.0.0-bin hive
sudo chown -R $USER:$USER /opt/hive

# Set environment variables
echo 'export HIVE_HOME=/opt/hive' >> ~/.bashrc
echo 'export PATH=$PATH:$HIVE_HOME/bin' >> ~/.bashrc
source ~/.bashrc

# Download MySQL connector
cd $HIVE_HOME/lib
```

```
wget https://repo1.maven.org/maven2/mysql/mysql-
   connector-java/8.0.33/mysql-connector-java-8.0.33.jar
```

12.2.4 Configure Hive

Create `/opt/hive/conf/hive-site.xml`:

```
<configuration>
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://localhost:3306/metastore?
           useSSL=false</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionDriverName</
           name>
        <value>com.mysql.cj.jdbc.Driver</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>hive</value>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>hivepassword</value>
    </property>
    <property>
        <name>hive.metastore.warehouse.dir</name>
        <value>/user/hive/warehouse</value>
    </property>
    <property>
        <name>hive.exec.scratchdir</name>
        <value>/tmp/hive</value>
    </property>
    <property>
        <name>hive.execution.engine</name>
        <value>tez</value>
    </property>
</configuration>
```

12.2.5 Initialize Metastore

```
# Create Hive directories in HDFS
hdfs dfs -mkdir -p /user/hive/warehouse
hdfs dfs -mkdir -p /tmp/hive
hdfs dfs -chmod 777 /tmp/hive
hdfs dfs -chmod 777 /user/hive/warehouse

# Initialize the metastore schema
schematool -dbType mysql -initSchema

# Start HiveServer2
hiveserver2 &

# Or start Hive CLI
hive
```

12.3 Working Example: Sales Data Warehouse

12.3.1 Create Database and Tables

```
-- Create database
CREATE DATABASE sales_db;
USE sales_db;

-- Create external table (data stored outside Hive)
CREATE EXTERNAL TABLE sales_raw (
    sale_id INT,
    product_name STRING,
    category STRING,
    amount DECIMAL(10,2),
    sale_date DATE,
    region STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/data/sales_raw';

-- Create managed ORC table (optimized storage)
CREATE TABLE sales_optimized (
```

```
    sale_id INT,
    product_name STRING,
    category STRING,
    amount DECIMAL(10,2),
    sale_date DATE,
    region STRING
)
PARTITIONED BY (year INT, month INT)
CLUSTERED BY (sale_id) INTO 4 BUCKETS
STORED AS ORC
TBLPROPERTIES ('transactional'='true');
```

12.3.2 Load Sample Data

```
# Create sample CSV data
cat > sales_data.csv << 'EOF'
1,Laptop,Electronics,1299.99,2024-01-15,North
2,Chair,Furniture,299.99,2024-01-16,South
3,Keyboard,Electronics,79.99,2024-02-10,East
4,Desk,Furniture,599.99,2024-02-12,West
5,Monitor,Electronics,349.99,2024-03-05,North
6,Laptop,Electronics,1299.99,2024-03-08,South
EOF

# Upload to HDFS
hdfs dfs -mkdir -p /user/data/sales_raw
hdfs dfs -put sales_data.csv /user/data/sales_raw/
```

12.3.3 Query and Transform Data

```
-- View raw data
SELECT * FROM sales_raw LIMIT 10;

-- Insert into optimized partitioned table
INSERT INTO TABLE sales_optimized PARTITION (year, month
    )
SELECT
    sale_id,
    product_name,
    category,
    amount,
```

```
    sale_date,
    region,
    YEAR(sale_date) as year,
    MONTH(sale_date) as month
FROM sales_raw;

-- Aggregation query
SELECT
    category,
    region,
    COUNT(*) as total_sales,
    SUM(amount) as revenue,
    AVG(amount) as avg_sale
FROM sales_optimized
WHERE year = 2024 AND month = 2
GROUP BY category, region
ORDER BY revenue DESC;

-- Window function example
SELECT
    product_name,
    amount,
    sale_date,
    AVG(amount) OVER (PARTITION BY category) as
       category_avg,
    RANK() OVER (PARTITION BY category ORDER BY amount
       DESC) as rank_in_category
FROM sales_optimized
WHERE year = 2024;
```

12.3.4 Update and Delete Operations (ACID)

```
-- Update records
UPDATE sales_optimized
SET amount = amount * 1.1
WHERE category = 'Electronics' AND year = 2024;

-- Delete records
DELETE FROM sales_optimized
WHERE amount < 100 AND year = 2024 AND month = 1;

-- Merge (UPSERT) operation
```

```
MERGE INTO sales_optimized AS target
USING sales_updates AS source
ON target.sale_id = source.sale_id
WHEN MATCHED THEN UPDATE SET amount = source.amount
WHEN NOT MATCHED THEN INSERT VALUES (
    source.sale_id,
    source.product_name,
    source.category,
    source.amount,
    source.sale_date,
    source.region,
    YEAR(source.sale_date),
    MONTH(source.sale_date)
);
```

12.4 Advanced Features

12.4.1 Partitioning

Partitions improve query performance by organizing data into subdirectories:

```
-- Show partitions
SHOW PARTITIONS sales_optimized;

-- Add partition manually
ALTER TABLE sales_optimized
ADD PARTITION (year=2024, month=4);

-- Drop partition
ALTER TABLE sales_optimized
DROP PARTITION (year=2023, month=1);

-- Dynamic partitioning (automatically creates
   partitions)
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;
```

12.4.2 Views and Materialized Views

```
-- Create view
CREATE VIEW high_value_sales AS
SELECT * FROM sales_optimized
WHERE amount > 500;

-- Create materialized view (cached results)
CREATE MATERIALIZED VIEW monthly_revenue AS
SELECT
    year,
    month,
    category,
    SUM(amount) as total_revenue
FROM sales_optimized
GROUP BY year, month, category;

-- Refresh materialized view
ALTER MATERIALIZED VIEW monthly_revenue REBUILD;
```

12.4.3 User-Defined Functions (UDF)

```
-- Add JAR containing custom UDF
ADD JAR /path/to/custom-udf.jar;

-- Create temporary function
CREATE TEMPORARY FUNCTION my_upper AS 'com.example.
   MyUpperUDF';

-- Use in query
SELECT my_upper(product_name) FROM sales_raw;
```

12.5 Integration with Spark

```
from pyspark.sql import SparkSession

# Create Spark session with Hive support
spark = SparkSession.builder \
    .appName("Hive Integration") \
    .enableHiveSupport() \
```

```
    .getOrCreate()

# Query Hive tables using SparkSQL
df = spark.sql("SELECT * FROM sales_db.sales_optimized
   WHERE year = 2024")
df.show()

# Write DataFrame to Hive table
df.write.mode("overwrite") \
    .partitionBy("year", "month") \
    .saveAsTable("sales_db.spark_sales")

# Read using Hive table format
spark.table("sales_db.sales_optimized").count()
```

12.6 Performance Optimization

1. **Use ORC or Parquet**: Columnar formats provide 10-100x compression
2. **Partition Data**: Partition by date or frequently filtered columns
3. **Bucketing**: Distribute data for join optimization
4. **Vectorization**: Enable for ORC/Parquet

   ```
   SET hive.vectorized.execution.enabled=true;
   ```

5. **Cost-Based Optimizer (CBO)**: Collect statistics

   ```
   ANALYZE TABLE sales_optimized COMPUTE STATISTICS;
   ANALYZE TABLE sales_optimized COMPUTE STATISTICS FOR
       COLUMNS;
   ```

6. **Tez Execution**: Use Tez instead of MapReduce

   ```
   SET hive.execution.engine=tez;
   ```

12.7 URL

Official Apache Hive website and documentation:
https://hive.apache.org
Documentation: https://cwiki.apache.org/confluence/display/Hive/

12.8 Books and Learning Resources

- **Programming Hive** (O'Reilly) - comprehensive HiveQL guide
- **Hive Essentials** (Packt) - practical examples and best practices
- **Apache Hive Cookbook** (Packt) - recipes for common tasks
- **Hive Language Manual** - official wiki documentation

12.9 Hardening and Security

1. **Metastore Security**: Enable TLS for metastore connections

```
<property>
    <name>hive.metastore.use.SSL</name>
    <value>true</value>
</property>
```

2. **Authentication**: Configure Kerberos or LDAP

```
<property>
    <name>hive.server2.authentication</name>
    <value>KERBEROS</value>
</property>
```

3. **Authorization with Apache Ranger/Sentry**:
    - Fine-grained access control on databases, tables, columns
    - Row-level filtering
    - Data masking for sensitive columns
4. **Table-Level ACLs**:

```
-- Grant privileges
GRANT SELECT ON TABLE sales_optimized TO USER alice;
GRANT ALL ON DATABASE sales_db TO ROLE analyst;

-- Revoke privileges
REVOKE SELECT ON TABLE sales_optimized FROM USER bob
   ;
```

5. **SQL Injection Prevention**:
   - Use prepared statements in application code
   - Validate and sanitize user inputs
   - Limit HiveServer2 user permissions
6. **Audit Logging**: Enable comprehensive audit logs

```
<property>
    <name>hive.server2.logging.operation.enabled</
       name>
    <value>true</value>
</property>
```

7. **Data Encryption**:
   - Use HDFS encryption zones for sensitive data
   - Encrypt metastore database connections
   - TLS for HiveServer2 client connections
8. **Network Security**:
   - Firewall rules to restrict HiveServer2 access (port 10000)
   - VPN or SSH tunnels for remote access
   - Separate metastore network segment

Chapter 13

Apache HBase

13.1 What HBase Is

Apache HBase is a distributed, scalable, column-family NoSQL database modeled after Google's Bigtable. Built on top of HDFS, HBase provides random, real-time read/write access to big data, making it ideal for sparse datasets where you need millisecond access to billions of rows.

13.1.1 Key Characteristics

- **Column-Oriented Storage**: Data organized by column families, not rows
- **Horizontal Scalability**: Linearly scalable to thousands of nodes
- **Strong Consistency**: Guarantees consistency for reads and writes
- **Automatic Sharding**: Tables automatically split into regions
- **Built on HDFS**: Leverages Hadoop's fault tolerance
- **No SQL**: Schema-free, sparse data model

13.1.2 Data Model

HBase uses a four-dimensional data model:

- **Row Key**: Primary key for indexing (lexicographically sorted)
- **Column Family**: Group of columns stored together
- **Column Qualifier**: Individual columns within a family
- **Timestamp**: Version identifier for each cell

Example structure:

```
Row: user123
  cf:name {timestamp=T1, value="Alice"}
  cf:email {timestamp=T1, value="alice@example.com"}
  activity:login {timestamp=T2, value="2024-01-15"}
  activity:purchase {timestamp=T3, value="$99.99"}
```

13.1.3 Architecture

- **HMaster**: Coordinates region assignment and DDL operations
- **RegionServer**: Serves reads/writes for assigned regions
- **ZooKeeper**: Coordinates distributed operations and leader election
- **HDFS**: Underlying storage layer

13.1.4 Use Cases

- Time-series data (IoT sensor data, logs, metrics)
- Real-time analytics on sparse datasets
- Message/event storage with high write throughput
- User profile and session data
- Content management systems
- Recommendation engines

13.2 Install Instructions

13.2.1 Prerequisites

HBase requires Hadoop and ZooKeeper:

```
# Verify Hadoop is running (Chapter 11)
jps | grep -E "NameNode|DataNode"

# Install ZooKeeper (Chapter 19) or use HBase embedded
   ZK for testing
```

13.2.2 Download and Install HBase

```
# Download HBase 2.5.8
cd /opt
sudo wget https://dlcdn.apache.org/hbase/2.5.8/hbase
   -2.5.8-bin.tar.gz
sudo tar -xzf hbase-2.5.8-bin.tar.gz
sudo mv hbase-2.5.8 hbase
sudo chown -R $USER:$USER /opt/hbase

# Set environment variables
echo 'export HBASE_HOME=/opt/hbase' >> ~/.bashrc
echo 'export PATH=$PATH:$HBASE_HOME/bin' >> ~/.bashrc
source ~/.bashrc
```

13.2.3 Configure HBase (Standalone Mode)

Edit `/opt/hbase/conf/hbase-site.xml`:

```
<configuration>
    <property>
        <name>hbase.rootdir</name>
        <value>file:///opt/hbase/data</value>
    </property>
    <property>
        <name>hbase.zookeeper.property.dataDir</name>
        <value>/opt/hbase/zookeeper</value>
    </property>
    <property>
```

```
        <name>hbase.unsafe.stream.capability.enforce</
           name>
        <value>false</value>
    </property>
</configuration>
```

Edit /opt/hbase/conf/hbase-env.sh:

```
export JAVA_HOME=/usr/lib/jvm/default-java
export HBASE_MANAGES_ZK=true
```

13.2.4 Start HBase

```
# Start HBase
start-hbase.sh

# Verify processes
jps
# Should show: HMaster, HRegionServer, HQuorumPeer

# Access HBase shell
hbase shell
```

Access web UI: http://localhost:16010

13.3 Working Example: User Activity Tracking

13.3.1 Create Table and Column Families

```
# In HBase shell
# Create table with two column families
create 'user_activity', 'profile', 'events'

# List tables
list

# Describe table
describe 'user_activity'
```

13.3.2 Insert Data (Put Operations)

```
# Insert user profile data
put 'user_activity', 'user001', 'profile:name', 'Alice
    Johnson'
put 'user_activity', 'user001', 'profile:email', '
    alice@example.com'
put 'user_activity', 'user001', 'profile:joined', '
    2024-01-01'

# Insert event data
put 'user_activity', 'user001', 'events:login_2024-01-15
    ', '09:30:00'
put 'user_activity', 'user001', 'events:purchase_2024
    -01-15', 'product_123'
put 'user_activity', 'user001', 'events:logout_2024
    -01-15', '17:45:00'

# Insert data for another user
put 'user_activity', 'user002', 'profile:name', 'Bob
    Smith'
put 'user_activity', 'user002', 'profile:email', '
    bob@example.com'
put 'user_activity', 'user002', 'events:login_2024-01-16
    ', '10:00:00'
```

13.3.3 Read Data (Get and Scan)

```
# Get specific row
get 'user_activity', 'user001'

# Get specific column family
get 'user_activity', 'user001', {COLUMN => 'profile'}

# Get specific column
get 'user_activity', 'user001', {COLUMN => 'profile:name
    '}

# Scan entire table
scan 'user_activity'

# Scan with limits
```

```
scan 'user_activity', {LIMIT => 5}

# Scan specific column family
scan 'user_activity', {COLUMNS => 'events'}

# Scan with row key prefix
scan 'user_activity', {STARTROW => 'user001', STOPROW =>
    'user002'}
```

13.3.4 Update and Delete Operations

```
# Update (just put with new value)
put 'user_activity', 'user001', 'profile:email', 'alice.
   j@example.com'

# Delete specific column
delete 'user_activity', 'user001', 'events:logout_2024
   -01-15'

# Delete entire row
deleteall 'user_activity', 'user002'

# View multiple versions
get 'user_activity', 'user001', {COLUMN => 'profile:
   email', VERSIONS => 5}
```

13.4 Java API Example

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

public class HBaseExample {
    public static void main(String[] args) throws
       Exception {
        // Configuration
        Configuration config = HBaseConfiguration.create
           ();
```

```
        config.set("hbase.zookeeper.quorum", "localhost"
            );
        config.set("hbase.zookeeper.property.clientPort"
            , "2181");

        // Connection
        try (Connection connection = ConnectionFactory.
            createConnection(config)) {

            // Get table
            Table table = connection.getTable(TableName.
                valueOf("user_activity"));

            // Put operation
            Put put = new Put(Bytes.toBytes("user003"));
            put.addColumn(
                Bytes.toBytes("profile"),
                Bytes.toBytes("name"),
                Bytes.toBytes("Charlie Brown")
            );
            table.put(put);

            // Get operation
            Get get = new Get(Bytes.toBytes("user003"));
            Result result = table.get(get);
            byte[] value = result.getValue(
                Bytes.toBytes("profile"),
                Bytes.toBytes("name")
            );
            System.out.println("Name: " + Bytes.toString
                (value));

            // Scan operation
            Scan scan = new Scan();
            scan.addFamily(Bytes.toBytes("profile"));
            ResultScanner scanner = table.getScanner(
                scan);

            for (Result res : scanner) {
                System.out.println("Row: " + Bytes.
                    toString(res.getRow()));
            }
            scanner.close();
```

```
            table.close();
        }
    }
}
```

13.5 Python API Example (HappyBase)

```
import happybase

# Connect to HBase
connection = happybase.Connection('localhost')

# Open table
table = connection.table('user_activity')

# Put data
table.put(b'user004', {
    b'profile:name': b'Diana Prince',
    b'profile:email': b'diana@example.com',
    b'events:login_2024-01-17': b'08:00:00'
})

# Get data
row = table.row(b'user004')
print(f"Name: {row[b'profile:name'].decode()}")

# Scan table
for key, data in table.scan():
    print(f"Row: {key.decode()}")
    for col, val in data.items():
        print(f"  {col.decode()} => {val.decode()}")

# Delete
table.delete(b'user004')

connection.close()
```

Install HappyBase:

```
pip install happybase
```

13.6 Performance Optimization

1. **Row Key Design**: Most critical for performance
   - Use composite keys: `userid_timestamp`
   - Avoid monotonically increasing keys (causes hotspots)
   - Use salting or hashing for even distribution
2. **Column Family Design**:
   - Keep number of column families small (1-3)
   - Group frequently accessed columns together
   - Use short names to reduce storage
3. **Compression**: Enable Snappy or LZ4

```
alter 'user_activity', {NAME => 'profile',
   COMPRESSION => 'SNAPPY'}
```

4. **Block Cache**: Configure for read-heavy workloads

```
<property>
    <name>hfile.block.cache.size</name>
    <value>0.4</value> <!-- 40% of heap -->
</property>
```

5. **Pre-splitting Tables**: Avoid initial hotspotting

```
create 'user_activity', 'cf', SPLITS => ['row100', '
   row200', 'row300']
```

6. **Bulk Loading**: Use ImportTSV or BulkLoad for large datasets

13.7 Monitoring and Management

```
# Check cluster status
status

# Check table status
status 'user_activity'

# Compact table (merge HFiles)
major_compact 'user_activity'

# Balance regions across servers
balance_switch true

# Flush memstore to disk
flush 'user_activity'
```

13.8 URL

Official Apache HBase website and documentation:
https://hbase.apache.org
Documentation: https://hbase.apache.org/book.html

13.9 Books and Learning Resources

- **HBase: The Definitive Guide** (O'Reilly) - comprehensive coverage
- **HBase in Action** (Manning) - practical patterns and use cases
- **HBase Administration Cookbook** (Packt) - operational best practices
- **Apache HBase Reference Guide** - official documentation

13.10 Hardening and Security

1. **Kerberos Authentication**: Enable strong authentication

```
<property>
    <name>hbase.security.authentication</name>
    <value>kerberos</value>
</property>
<property>
    <name>hbase.security.authorization</name>
    <value>true</value>
</property>
```

2. **Cell-Level ACLs**: Fine-grained access control

```
# Grant permissions at table level
grant 'alice', 'RW', 'user_activity'

# Grant permissions at column family level
grant 'bob', 'R', 'user_activity', 'profile'

# Revoke permissions
revoke 'bob', 'user_activity'
```

3. **Encryption at Rest**: Use HDFS encryption zones

4. **TLS/SSL**: Enable encrypted client-server communication

```
<property>
    <name>hbase.ssl.enabled</name>
    <value>true</value>
</property>
```

5. **Audit Logging**: Track all data access
   - Enable HBase audit logs
   - Integrate with centralized log management (ELK, Splunk)
   - Monitor for suspicious access patterns

6. **Network Segmentation**:
   - Isolate HBase cluster in dedicated network
   - Firewall rules for RegionServer ports (16020, 16030)

- VPN for remote administrative access

7. **Data Backup**:

```
# Snapshot for backup
hbase shell
snapshot 'user_activity', '
   user_activity_backup_20240115'

# Export snapshot
hbase org.apache.hadoop.hbase.snapshot.
   ExportSnapshot \
  -snapshot user_activity_backup_20240115 \
  -copy-to hdfs://backup-cluster:9000/hbase
```

Chapter 14

Apache Arrow

14.1 What Arrow Is

Apache Arrow is a cross-language development platform for in-memory columnar data. It specifies a standardized language-independent columnar memory format for flat and hierarchical data, enabling zero-copy data sharing between different systems without serialization overhead.

14.1.1 Key Features

- **Zero-Copy Data Sharing**: Share data between processes without serialization
- **Language Agnostic**: Implementations in C++, Python, Java, Rust, Go, JavaScript, C#, Ruby, Julia
- **Columnar Format**: Optimized for analytical workloads and SIMD operations
- **IPC/RPC**: Arrow Flight for high-performance data transfer
- **Ecosystem Integration**: Works with Pandas, Spark, Parquet, DuckDB
- **Memory Efficiency**: Designed for modern CPU cache hierarchies

14.1.2 Use Cases

- High-performance data interchange between systems
- Accelerating Pandas/DataFrame operations
- Reading/writing Parquet files efficiently
- Building data processing pipelines
- Streaming analytics with Arrow Flight
- GPU-accelerated analytics (RAPIDS)

14.2 Install Instructions

14.2.1 Python (PyArrow)

```
pip install pyarrow

# With Pandas integration
pip install pyarrow pandas

# Verify installation
python -c "import pyarrow; print(pyarrow.__version__)"
```

14.2.2 Other Languages

```
# Java (Maven)
# Add to pom.xml:
# <dependency>
#   <groupId>org.apache.arrow</groupId>
#   <artifactId>arrow-vector</artifactId>
#   <version>15.0.0</version>
# </dependency>

# Rust
cargo add arrow

# JavaScript/Node.js
npm install apache-arrow
```

14.3 Working Example: Data Processing with PyArrow

14.3.1 Creating Arrow Tables

```
import pyarrow as pa
import pyarrow.compute as pc

# Create arrays
names = pa.array(["Alice", "Bob", "Charlie", "Diana"])
ages = pa.array([25, 30, 35, 28])
salaries = pa.array([50000.0, 75000.0, 90000.0,
   65000.0])

# Create table
table = pa.table({
    "name": names,
    "age": ages,
    "salary": salaries
})

print(table)
print(f"Schema: {table.schema}")
print(f"Num rows: {table.num_rows}")
```

14.3.2 Arrow with Pandas

```
import pandas as pd
import pyarrow as pa

# Pandas DataFrame to Arrow Table
df = pd.DataFrame({
    "id": range(1000000),
    "value": range(1000000)
})

# Convert to Arrow (zero-copy when possible)
table = pa.Table.from_pandas(df)

# Convert back to Pandas
```

```
df_back = table.to_pandas()

# Memory-mapped reading (very fast for large files)
table = pa.ipc.open_file("data.arrow").read_all()
```

14.3.3 Parquet Integration

```
import pyarrow.parquet as pq

# Write Parquet file
pq.write_table(table, "sales_data.parquet", compression=
    "snappy")

# Read Parquet file
table = pq.read_table("sales_data.parquet")

# Read specific columns only
table = pq.read_table("sales_data.parquet", columns=["
    name", "salary"])

# Read with filter (predicate pushdown)
table = pq.read_table(
    "sales_data.parquet",
    filters=[("age", ">", 30)]
)

# Read partitioned dataset
dataset = pq.ParquetDataset("data/year=2024/")
table = dataset.read()
```

14.3.4 Compute Functions

```
import pyarrow.compute as pc

# Filter
filtered = table.filter(pc.field("age") > 30)

# Aggregations
mean_salary = pc.mean(table["salary"])
max_age = pc.max(table["age"])
```

```
# String operations
upper_names = pc.utf8_upper(table["name"])

# Sorting
sorted_table = table.sort_by([("salary", "descending")])

# Group by aggregation
from pyarrow import acero
# Use acero for complex aggregations
```

14.4 Arrow Flight (High-Performance RPC)

```
import pyarrow.flight as flight

# Flight Server
class MyFlightServer(flight.FlightServerBase):
    def __init__(self, location):
        super().__init__(location)
        self.tables = {}

    def do_put(self, context, descriptor, reader, writer
       ):
        key = descriptor.path[0].decode()
        self.tables[key] = reader.read_all()

    def do_get(self, context, ticket):
        key = ticket.ticket.decode()
        return flight.RecordBatchStream(self.tables[key
           ])

# Start server
server = MyFlightServer("grpc://0.0.0.0:8815")
server.serve()

# Flight Client
client = flight.connect("grpc://localhost:8815")

# Upload data
upload_descriptor = flight.FlightDescriptor.for_path("
   my_table")
writer, _ = client.do_put(upload_descriptor, table.
```

```
    schema)
writer.write_table(table)
writer.close()

# Download data
ticket = flight.Ticket(b"my_table")
reader = client.do_get(ticket)
result = reader.read_all()
```

14.5 Performance Tips

1. **Use Memory Mapping**: For large files, use memory-mapped I/O
2. **Column Pruning**: Read only needed columns from Parquet
3. **Predicate Pushdown**: Filter at read time when possible
4. **Chunked Processing**: Process large datasets in chunks
5. **Zero-Copy Slicing**: Use `slice()` instead of copying data
6. **Dictionary Encoding**: Use for low-cardinality string columns

14.6 URL

https://arrow.apache.org
Documentation: https://arrow.apache.org/docs/

14.7 Books and Resources

- **Apache Arrow Documentation** - comprehensive official guide
- **PyArrow API Reference** - Python-specific documentation
- **Arrow Cookbook** - practical examples and recipes

14.8 Hardening and Security

1. **Schema Validation**: Always validate schemas on data ingest

```
expected_schema = pa.schema([
    ("id", pa.int64()),
    ("name", pa.string())
])
if table.schema != expected_schema:
    raise ValueError("Schema mismatch")
```

2. **Memory Limits**: Set memory limits for large operations

3. **Flight Authentication**: Enable authentication for Arrow Flight

```
# Use TLS and authentication tokens
client = flight.connect(
    "grpc+tls://server:8815",
    tls_root_certs=cert_data
)
```

4. **Input Sanitization**: Validate file sources before reading

5. **Resource Cleanup**: Properly close readers and writers

Chapter 15

Apache Drill

15.1 What Drill Is

Apache Drill is a schema-free SQL query engine for big data exploration. It enables direct queries on semi-structured and nested data (JSON, Parquet, CSV) without requiring upfront schema definitions, making it ideal for data exploration and ad-hoc analytics.

15.1.1 Key Features

- **Schema-Free**: Discover schema at query time, no ETL required
- **ANSI SQL**: Standard SQL syntax with extensions for nested data
- **Multiple Data Sources**: Files (JSON, Parquet, CSV), databases (MongoDB, HBase, JDBC)
- **Nested Data Support**: Query complex JSON structures directly
- **Distributed**: Scale across cluster for large datasets
- **Self-Service**: Analysts can explore data without DBA involvement

15.1.2 Supported Data Sources

- **File Systems**: Local, HDFS, S3, Azure Blob, Google Cloud Storage
- **File Formats**: JSON, Parquet, CSV, TSV, Avro, log files
- **Databases**: MongoDB, HBase, Hive, Kudu, Elasticsearch
- **JDBC**: Any JDBC-compliant database (MySQL, PostgreSQL, etc.)

15.2 Install Instructions

15.2.1 Download and Install

```
# Download Drill 1.21.1
cd /opt
sudo wget https://dlcdn.apache.org/drill/drill-1.21.1/
   apache-drill-1.21.1.tar.gz
sudo tar -xzf apache-drill-1.21.1.tar.gz
sudo mv apache-drill-1.21.1 drill
sudo chown -R $USER:$USER /opt/drill

# Set environment variables
echo 'export DRILL_HOME=/opt/drill' >> ~/.bashrc
echo 'export PATH=$PATH:$DRILL_HOME/bin' >> ~/.bashrc
source ~/.bashrc
```

15.2.2 Start Drill (Embedded Mode)

```
# Start Drill shell (embedded mode for development)
drill-embedded

# Or start Drillbit daemon
drillbit.sh start

# Access web UI
# http://localhost:8047
```

15.3 Working Example: Querying JSON Data

15.3.1 Sample JSON Data

```
# Create sample JSON file
mkdir -p /tmp/drill_data
cat > /tmp/drill_data/employees.json << 'EOF'
{"id": 1, "name": "Alice", "department": "Engineering",
   "salary": 85000, "skills": ["Python", "Java"]}
{"id": 2, "name": "Bob", "department": "Sales", "salary"
   : 72000, "skills": ["Communication"]}
{"id": 3, "name": "Charlie", "department": "Engineering"
   , "salary": 92000, "skills": ["Go", "Rust", "Python"
   ]}
{"id": 4, "name": "Diana", "department": "Marketing", "
   salary": 68000, "skills": ["SEO", "Analytics"]}
EOF
```

15.3.2 Query JSON with SQL

```
-- In Drill shell or web UI

-- Basic query
SELECT * FROM dfs.`/tmp/drill_data/employees.json`;

-- Filter and project
SELECT name, department, salary
FROM dfs.`/tmp/drill_data/employees.json`
WHERE salary > 75000;

-- Query nested arrays
SELECT name, FLATTEN(skills) AS skill
FROM dfs.`/tmp/drill_data/employees.json`;

-- Aggregations
SELECT department, COUNT(*) as count, AVG(salary) as
   avg_salary
FROM dfs.`/tmp/drill_data/employees.json`
GROUP BY department
ORDER BY avg_salary DESC;
```

```
-- Array functions
SELECT name, REPEATED_COUNT(skills) as num_skills
FROM dfs.`/tmp/drill_data/employees.json`
WHERE REPEATED_CONTAINS(skills, 'Python');
```

15.3.3 Query Parquet Files

```
-- Query Parquet directly (auto-discovers schema)
SELECT * FROM dfs.`/data/sales.parquet` LIMIT 100;

-- Join Parquet and JSON
SELECT p.product_name, j.category
FROM dfs.`/data/products.parquet` p
JOIN dfs.`/data/categories.json` j
ON p.category_id = j.id;
```

15.4 Storage Plugin Configuration

15.4.1 Configure S3 Access

In Drill web UI (http://localhost:8047), go to Storage → Update for `dfs`:

```
{
  "type": "file",
  "connection": "s3a://my-bucket",
  "config": {
    "fs.s3a.access.key": "YOUR_ACCESS_KEY",
    "fs.s3a.secret.key": "YOUR_SECRET_KEY"
  },
  "workspaces": {
    "data": {
      "location": "/data",
      "writable": false
    }
  },
  "formats": {
    "parquet": {"type": "parquet"},
    "json": {"type": "json"}
  }
}
```

15.4.2 Connect to MongoDB

```
{
  "type": "mongo",
  "connection": "mongodb://localhost:27017",
  "enabled": true
}
```

Query:

```
SELECT * FROM mongo.mydb.users WHERE age > 25;
```

15.5 Performance Optimization

1. **Use Parquet**: Much faster than JSON for large datasets
2. **Partition Data**: Organize files in directories by date/category
3. **Column Pruning**: Select only needed columns
4. **Filter Pushdown**: Use WHERE clauses to reduce data scanned
5. **Increase Parallelism**: Configure `planner.width.max_per_node`

15.6 URL

https://drill.apache.org
Documentation: https://drill.apache.org/docs/

15.7 Books and Resources

- **Learning Apache Drill** (O'Reilly) - comprehensive guide
- **Apache Drill Documentation** - official reference

15.8 Hardening and Security

1. **User Authentication**: Enable PAM or LDAP authentication

```
# In drill-override.conf
drill.exec.security.user.auth.enabled: true
drill.exec.security.user.auth.impl: "pam"
```

2. **Impersonation**: Enable user impersonation for access control

3. **Query Limits**: Set result set and memory limits

```
drill.exec.query.max_rows: 1000000
drill.exec.memory.operator.max: 8589934592
```

4. **TLS/SSL**: Enable encrypted connections

5. **Network Security**: Firewall Drill ports (8047, 31010)

Chapter 16

Apache Pinot

16.1 What Pinot Is

Apache Pinot is a real-time distributed OLAP datastore designed for ultra-low-latency, high-throughput analytics. Originally developed at LinkedIn for user-facing analytics, Pinot excels at providing sub-second query responses on massive datasets with high query concurrency.

16.1.1 Key Features

- **Ultra-Low Latency**: Sub-second P99 query latency at high throughput
- **Real-Time Ingestion**: Stream data from Kafka with minimal delay
- **Hybrid Tables**: Combine real-time and batch data seamlessly
- **Pluggable Indexing**: Star-tree, sorted, inverted, range, text, JSON indexes
- **SQL Interface**: Standard SQL with analytics extensions
- **High Concurrency**: Handle thousands of queries per second

16.1.2 Use Cases

- User-facing analytics dashboards
- Real-time metrics and monitoring
- Ad-hoc OLAP queries on event data
- Anomaly detection on streaming data
- Site analytics and A/B testing

16.2 Install Instructions

16.2.1 Download and Install

```
# Download Pinot 1.0.0
cd /opt
sudo wget https://dlcdn.apache.org/pinot/apache-pinot
   -1.0.0/apache-pinot-1.0.0-bin.tar.gz
sudo tar -xzf apache-pinot-1.0.0-bin.tar.gz
sudo mv apache-pinot-1.0.0-bin pinot
sudo chown -R $USER:$USER /opt/pinot

# Set environment
echo 'export PINOT_HOME=/opt/pinot' >> ~/.bashrc
echo 'export PATH=$PATH:$PINOT_HOME/bin' >> ~/.bashrc
source ~/.bashrc
```

16.2.2 Start Pinot Quickstart

```
# Start all components (ZK, Controller, Broker, Server)
cd /opt/pinot
bin/pinot-admin.sh QuickStart -type batch

# Access Pinot UI
# http://localhost:9000
```

16.3 Working Example: Real-Time Analytics

16.3.1 Create Schema

```
{
  "schemaName": "events",
  "dimensionFieldSpecs": [
    {"name": "user_id", "dataType": "STRING"},
    {"name": "event_type", "dataType": "STRING"},
    {"name": "country", "dataType": "STRING"}
  ],
  "metricFieldSpecs": [
    {"name": "count", "dataType": "LONG"}
  ],
  "dateTimeFieldSpecs": [{
    "name": "timestamp",
    "dataType": "TIMESTAMP",
    "format": "1:MILLISECONDS:EPOCH",
    "granularity": "1:MILLISECONDS"
  }]
}
```

16.3.2 Create Table

```
{
  "tableName": "events",
  "tableType": "REALTIME",
  "segmentsConfig": {
    "timeColumnName": "timestamp",
    "replication": "1"
  },
  "tenants": {},
  "tableIndexConfig": {
    "loadMode": "MMAP",
    "streamConfigs": {
      "streamType": "kafka",
      "stream.kafka.consumer.type": "lowlevel",
      "stream.kafka.topic.name": "events",
      "stream.kafka.broker.list": "localhost:9092",
```

```
      "stream.kafka.consumer.factory.class.name": "org.
         apache.pinot.plugin.stream.kafka20.
         KafkaConsumerFactory",
      "stream.kafka.decoder.class.name": "org.apache.
         pinot.plugin.stream.kafka.
         KafkaJSONMessageDecoder"
    }
  },
  "metadata": {}
}
```

16.3.3 Query Data

```
-- Count events by type
SELECT event_type, COUNT(*) as cnt
FROM events
GROUP BY event_type
ORDER BY cnt DESC
LIMIT 10;

-- Time-series aggregation
SELECT
  DATETIMECONVERT(timestamp, '1:MILLISECONDS:EPOCH', '1:
     HOURS:EPOCH', '1:HOURS') as hour,
  COUNT(*) as events
FROM events
WHERE timestamp > ago('PT24H')
GROUP BY hour
ORDER BY hour;

-- Top countries by user count
SELECT country, DISTINCTCOUNT(user_id) as users
FROM events
GROUP BY country
ORDER BY users DESC;
```

16.4 URL

https://pinot.apache.org
Documentation: https://docs.pinot.apache.org/

16.5 Books and Resources

- **Apache Pinot Documentation** - comprehensive official guide
- **Pinot Recipes** - common use case patterns

16.6 Hardening and Security

1. **TLS/SSL**: Enable encrypted cluster communication
2. **Authentication**: Configure basic auth or OAuth
3. **Rate Limiting**: Protect against query overload

```
pinot.broker.query.limit.disabled=false
pinot.broker.query.limit.qps=1000
```

4. **Access Control**: Configure table-level permissions
5. **Network Security**: Firewall Controller (9000), Broker (8099), Server (8098) ports

Chapter 17

Apache Superset

17.1 What Superset Is

Apache Superset is a modern, enterprise-ready business intelligence web application for data exploration and visualization. Originally developed at Airbnb, Superset enables users to create interactive dashboards, explore data through SQL, and build rich visualizations without writing code.

17.1.1 Key Features

- **SQL Lab**: Powerful SQL IDE with syntax highlighting and autocomplete
- **Rich Visualizations**: 40+ chart types including time-series, geospatial, and pivot tables
- **Interactive Dashboards**: Drag-and-drop dashboard builder with filters
- **Database Support**: Connect to any SQLAlchemy-compatible database
- **Semantic Layer**: Define metrics and dimensions for self-service analytics
- **Access Control**: Role-based access with row-level security

17.1.2 Supported Databases

PostgreSQL, MySQL, Presto/Trino, Hive, Spark SQL, Druid, Pinot, ClickHouse, BigQuery, Snowflake, Redshift, Doris, DuckDB, and 40+ others.

17.2 Install Instructions

17.2.1 Install with pip

```
# Create virtual environment
python3 -m venv superset-venv
source superset-venv/bin/activate

# Install Superset
pip install apache-superset

# Set secret key (required)
export SUPERSET_SECRET_KEY=$(openssl rand -base64 42)

# Initialize database
superset db upgrade

# Create admin user
superset fab create-admin \
  --username admin \
  --firstname Admin \
  --lastname User \
  --email admin@example.com \
  --password admin

# Load examples (optional)
superset load_examples

# Initialize Superset
superset init

# Start development server
superset run -p 8088 --with-threads --reload
```

Access at: http://localhost:8088

17.2.2 Docker Installation (Recommended for Production)

```
# Clone repository
git clone https://github.com/apache/superset.git
cd superset

# Start with Docker Compose
docker compose -f docker-compose-non-dev.yml up -d

# Access at http://localhost:8088
# Default credentials: admin/admin
```

17.3 Working Example: Sales Dashboard

17.3.1 Connect Database

In Superset UI: Settings → Database Connections → + Database

```
# PostgreSQL connection string
postgresql://user:password@localhost:5432/sales_db

# MySQL
mysql://user:password@localhost:3306/sales_db

# Presto/Trino
trino://user@localhost:8080/hive/default
```

17.3.2 Create Dataset

Settings → Datasets → + Dataset:

1. Select database and schema
2. Choose table (e.g., `sales`)
3. Configure columns (metrics, dimensions, temporal)

17.3.3 Build Chart

Charts → + Chart:

```
-- Example: Revenue by Category (Bar Chart)
-- Dataset: sales
-- Metrics: SUM(amount) as Revenue
-- Dimensions: category
-- Filters: sale_date >= '2024-01-01'
```

17.3.4 Create Dashboard

Dashboards → + Dashboard:

1. Add charts using drag-and-drop
2. Configure layout and sizing
3. Add filter boxes for interactivity
4. Set refresh interval
5. Publish and share

17.4 SQL Lab Features

```
-- SQL Lab supports advanced features

-- CTEs (Common Table Expressions)
WITH monthly_sales AS (
    SELECT DATE_TRUNC('month', sale_date) as month,
           SUM(amount) as revenue
    FROM sales
    GROUP BY 1
)
SELECT * FROM monthly_sales ORDER BY month;

-- Save queries for reuse
-- Create datasets from queries
-- Schedule queries (with Celery)
-- Export results to CSV
```

17.5 Configuration

Create `superset_config.py`:

```
import os

# Security
SECRET_KEY = os.environ.get('SUPERSET_SECRET_KEY')
CSRF_ENABLED = True

# Database
SQLALCHEMY_DATABASE_URI = 'postgresql://user:
   pass@localhost/superset'

# Cache (Redis recommended)
CACHE_CONFIG = {
    'CACHE_TYPE': 'redis',
    'CACHE_REDIS_URL': 'redis://localhost:6379/0'
}

# Feature flags
FEATURE_FLAGS = {
    'ENABLE_TEMPLATE_PROCESSING': True,
    'DASHBOARD_NATIVE_FILTERS': True,
}

# Row limit
ROW_LIMIT = 50000
SQL_MAX_ROW = 100000
```

17.6 URL

https://superset.apache.org
Documentation: https://superset.apache.org/docs/

17.7 Books and Resources

- **Apache Superset Documentation** - comprehensive official guide
- **Superset GitHub Wiki** - community recipes and tips
- **Preset Blog** - best practices from commercial Superset provider

17.8 Hardening and Security

1. **HTTPS Only**: Deploy behind reverse proxy with TLS

```
# In superset_config.py
ENABLE_PROXY_FIX = True
TALISMAN_ENABLED = True
```

2. **Authentication**: Configure OAuth, LDAP, or SAML

```
from flask_appbuilder.security.manager import
    AUTH_OAUTH
AUTH_TYPE = AUTH_OAUTH
```

3. **Row-Level Security**: Restrict data access by user/role

```
# Define RLS filter in Superset UI
# e.g., region = '{{ current_user().region }}'
```

4. **Role-Based Access Control**:
   - Create roles with specific permissions
   - Assign database/dataset access per role
   - Use `Gamma` role for limited access users

5. **SQL Injection Prevention**:
   - Disable `ALLOW_ADHOC_SUBQUERY` for untrusted users
   - Limit database permissions
   - Use read-only database accounts

6. **Rate Limiting**: Prevent abuse

```
RATELIMIT_ENABLED = True
RATELIMIT_DEFAULT = "100 per minute"
```

7. **Audit Logging**: Track user activity

Chapter 18

Apache Parquet

18.1 What Parquet Is

Apache Parquet is a columnar storage file format designed for efficient data storage and retrieval. It provides excellent compression and encoding schemes with enhanced performance to handle complex data in bulk, making it the de facto standard for analytical workloads in the big data ecosystem.

18.1.1 Key Features

- **Columnar Storage**: Store and process only needed columns
- **Efficient Compression**: Column-specific compression (Snappy, Gzip, LZ4, Zstd)
- **Encoding Schemes**: Dictionary, run-length, delta, bit-packing encoding
- **Schema Evolution**: Add, remove, or modify columns over time
- **Predicate Pushdown**: Filter at storage level before reading
- **Universal Support**: Spark, Hive, Presto, Drill, Pandas, DuckDB

18.1.2 Advantages Over Row-Based Formats

- 10-100x compression improvement
- Read only needed columns (projection pushdown)
- Better for analytical queries (aggregations, filters)
- Efficient metadata storage
- Support for nested/complex data structures

18.2 Install Instructions

```
# Python (PyArrow - recommended)
pip install pyarrow pandas

# Alternative: fastparquet
pip install fastparquet

# Verify
python -c "import pyarrow.parquet as pq; print('Parquet
   ready')"
```

18.3 Working Example: Parquet with Python

18.3.1 Write Parquet Files

```
import pandas as pd
import pyarrow as pa
import pyarrow.parquet as pq

# Create sample data
df = pd.DataFrame({
    'id': range(1000000),
    'name': [f'user_{i}' for i in range(1000000)],
    'amount': [i * 1.5 for i in range(1000000)],
    'date': pd.date_range('2024-01-01', periods=1000000,
        freq='s')
})
```

```
# Write Parquet with compression
pq.write_table(
    pa.Table.from_pandas(df),
    'sales.parquet',
    compression='snappy'  # or 'gzip', 'lz4', 'zstd'
)

# Write partitioned dataset
df.to_parquet(
    'sales_partitioned/',
    partition_cols=['date'],
    engine='pyarrow'
)
```

18.3.2 Read Parquet Files

```
# Read entire file
df = pd.read_parquet('sales.parquet')

# Read specific columns only
df = pd.read_parquet('sales.parquet', columns=['id', '
   amount'])

# Read with filter (predicate pushdown)
table = pq.read_table(
    'sales.parquet',
    filters=[('amount', '>', 1000)]
)

# Read partitioned dataset
df = pd.read_parquet('sales_partitioned/')

# Inspect metadata
parquet_file = pq.ParquetFile('sales.parquet')
print(parquet_file.schema)
print(parquet_file.metadata)
```

18.3.3 Parquet with Spark

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Parquet").
   getOrCreate()

# Read Parquet
df = spark.read.parquet("s3://bucket/sales.parquet")

# Write Parquet with partitioning
df.write.mode("overwrite") \
    .partitionBy("year", "month") \
    .parquet("s3://bucket/sales_partitioned/")

# Query directly
spark.sql("SELECT * FROM parquet.`s3://bucket/sales.
   parquet`")
```

18.4 Schema and Metadata

```
# View schema
parquet_file = pq.ParquetFile('sales.parquet')
print(parquet_file.schema_arrow)

# View row group metadata
print(f"Num row groups: {parquet_file.metadata.
   num_row_groups}")
for i in range(parquet_file.metadata.num_row_groups):
    rg = parquet_file.metadata.row_group(i)
    print(f"Row group {i}: {rg.num_rows} rows")
```

18.5 Performance Tips

1. **Choose Right Compression**: Snappy for speed, Zstd for compression ratio
2. **Partition Wisely**: Partition by frequently filtered columns (date)
3. **Row Group Size**: Default 128MB works for most cases

4. **Dictionary Encoding**: Automatic for low-cardinality strings
5. **Column Pruning**: Select only needed columns

18.6 URL

https://parquet.apache.org

18.7 Books and Resources

- **Apache Parquet Documentation** - format specification
- **PyArrow Documentation** - Python usage guide

18.8 Hardening and Security

1. **Encryption at Rest**: Use Parquet modular encryption

```
# Parquet encryption (requires encryption keys)
encryption_config = pq.EncryptionConfiguration(
    footer_key="footer_key",
    column_keys={"amount": "column_key"}
)
```

2. **File Permissions**: Restrict access to Parquet files
3. **Schema Validation**: Validate schemas before processing
4. **Secure Storage**: Use encrypted S3/HDFS

Chapter 19

Apache ZooKeeper

19.1 What ZooKeeper Is

Apache ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. It's the backbone of many distributed systems, providing reliable coordination primitives that are notoriously difficult to implement correctly.

19.1.1 Key Features

- **Configuration Management**: Centralized configuration storage
- **Leader Election**: Elect leaders in distributed systems
- **Distributed Locks**: Coordinate access to shared resources
- **Service Discovery**: Track available services and endpoints
- **Barrier Synchronization**: Coordinate distributed processes
- **Sequential Ordering**: Guaranteed ordering of operations

19.1.2 Data Model

ZooKeeper maintains a hierarchical namespace similar to a file system:

- **ZNodes**: Data nodes (like files/directories combined)
- **Ephemeral Nodes**: Deleted when session ends (for service discovery)
- **Sequential Nodes**: Auto-incrementing names (for locks/queues)
- **Watches**: Notifications when data changes

19.1.3 Systems Using ZooKeeper

Kafka (metadata), HBase (coordination), Hadoop (HA), Solr/SolrCloud, Neo4j, and many others.

19.2 Install Instructions

```
# Download ZooKeeper 3.9.1
cd /opt
sudo wget https://dlcdn.apache.org/zookeeper/zookeeper
   -3.9.1/apache-zookeeper-3.9.1-bin.tar.gz
sudo tar -xzf apache-zookeeper-3.9.1-bin.tar.gz
sudo mv apache-zookeeper-3.9.1-bin zookeeper
sudo chown -R $USER:$USER /opt/zookeeper

# Set environment
echo 'export ZOOKEEPER_HOME=/opt/zookeeper' >> ~/.bashrc
echo 'export PATH=$PATH:$ZOOKEEPER_HOME/bin' >> ~/.
   bashrc
source ~/.bashrc

# Create data directory
mkdir -p /opt/zookeeper/data
```

19.2.1 Configuration

Create `/opt/zookeeper/conf/zoo.cfg`:

```
tickTime=2000
dataDir=/opt/zookeeper/data
clientPort=2181
initLimit=5
syncLimit=2

# For cluster (ensemble) - add for each server:
# server.1=zk1:2888:3888
# server.2=zk2:2888:3888
# server.3=zk3:2888:3888
```

19.2.2 Start ZooKeeper

```
# Start server
zkServer.sh start

# Check status
zkServer.sh status

# Connect with CLI
zkCli.sh -server localhost:2181
```

19.3 Working Example: Configuration Management

```
# In zkCli.sh

# Create configuration znodes
create /config ""
create /config/database '{"host":"db.example.com","port
    ":5432}'
create /config/cache '{"host":"redis.example.com","port
    ":6379}'

# Read configuration
get /config/database
```

```
# Update configuration
set /config/database '{"host":"db2.example.com","port
    ":5432}'

# List children
ls /config

# Watch for changes (one-time trigger)
get -w /config/database

# Delete
delete /config/cache
```

19.3.1 Python Client (Kazoo)

```
from kazoo.client import KazooClient
import json

# Connect
zk = KazooClient(hosts='localhost:2181')
zk.start()

# Create node
zk.ensure_path("/config")
zk.create("/config/app", json.dumps({
    "version": "1.0",
    "feature_flags": {"new_ui": True}
}).encode())

# Read node
data, stat = zk.get("/config/app")
config = json.loads(data.decode())
print(f"Config: {config}")

# Watch for changes
@zk.DataWatch("/config/app")
def watch_config(data, stat):
    if data:
        print(f"Config changed: {json.loads(data.decode
            ())}")
```

```
# Distributed lock
lock = zk.Lock("/locks/mylock", "my-identifier")
with lock:
    print("Holding lock, doing work...")

# Leader election
election = zk.Election("/election", "my-identifier")
election.run(my_leader_function)

zk.stop()
```

Install: `pip install kazoo`

19.4 Ensemble (Cluster) Setup

For production, run 3 or 5 ZooKeeper nodes:

```
# On each server, create myid file
# Server 1:
echo "1" > /opt/zookeeper/data/myid

# Server 2:
echo "2" > /opt/zookeeper/data/myid

# Server 3:
echo "3" > /opt/zookeeper/data/myid

# zoo.cfg on all servers:
server.1=zk1.example.com:2888:3888
server.2=zk2.example.com:2888:3888
server.3=zk3.example.com:2888:3888
```

19.5 URL

https://zookeeper.apache.org

19.6 Books and Resources

- **ZooKeeper: Distributed Process Coordination** (O'Reilly) - definitive guide

- **Apache ZooKeeper Essentials** (Packt) - practical patterns

19.7 Hardening and Security

1. **SASL Authentication**: Enable Kerberos or DIGEST-MD5

```
# In zoo.cfg
authProvider.1=org.apache.zookeeper.server.auth.
   SASLAuthenticationProvider
requireClientAuthScheme=sasl
```

2. **ACLs on ZNodes**: Restrict access

```
# In zkCli.sh
setAcl /config/secrets auth:admin:cdrwa
```

3. **TLS Encryption**: Enable client-server encryption
4. **Network Isolation**: Firewall ports 2181, 2888, 3888
5. **Quorum TLS**: Encrypt server-to-server communication
6. **Four Letter Words**: Disable or whitelist admin commands

```
4lw.commands.whitelist=srvr,stat
```

Chapter 20

Apache Druid

20.1 What Druid Is

Apache Druid is a high-performance, real-time analytics database designed for fast slice-and-dice analytics on large datasets. Originally developed at Metamarkets, Druid excels at OLAP queries with sub-second response times, making it ideal for user-facing analytics applications.

20.1.1 Key Features

- **Sub-Second Queries**: Fast aggregations on billions of rows
- **Real-Time Ingestion**: Stream data from Kafka with low latency
- **Column-Oriented**: Optimized for analytical workloads
- **Time-Based Partitioning**: Native time-series support
- **Approximate Algorithms**: HyperLogLog, DataSketches for fast cardinality
- **High Availability**: No single point of failure

20.1.2 Use Cases

- Clickstream analytics
- Network flow analytics
- Server metrics analysis
- Application performance monitoring
- Business intelligence dashboards

20.2 Install Instructions

```
# Download Druid 28.0.0
cd /opt
sudo wget https://dlcdn.apache.org/druid/28.0.0/apache-
   druid-28.0.0-bin.tar.gz
sudo tar -xzf apache-druid-28.0.0-bin.tar.gz
sudo mv apache-druid-28.0.0 druid
sudo chown -R $USER:$USER /opt/druid

# Start single-server quickstart
cd /opt/druid
./bin/start-druid

# Access Druid Console: http://localhost:8888
```

20.3 Working Example: Event Analytics

20.3.1 Ingest Data from Kafka

Create ingestion spec `kafka-ingestion.json`:

```
{
  "type": "kafka",
  "spec": {
    "dataSchema": {
      "dataSource": "events",
      "timestampSpec": {"column": "timestamp", "format":
          "iso"},
      "dimensionsSpec": {
        "dimensions": ["user_id", "event_type", "country
          "]
      },
      "metricsSpec": [
        {"type": "count", "name": "count"},
        {"type": "longSum", "name": "total_value", "
          fieldName": "value"}
      ],
      "granularitySpec": {
        "type": "uniform",
        "segmentGranularity": "HOUR",
        "queryGranularity": "MINUTE"
      }
    },
    "ioConfig": {
      "topic": "events",
      "consumerProperties": {"bootstrap.servers": "
        localhost:9092"}
    }
  }
}
```

Submit: `POST /druid/indexer/v1/supervisor`

20.3.2 Query Data

```
-- Native Druid SQL
SELECT
  TIME_FLOOR(__time, 'PT1H') AS hour,
  country,
  COUNT(*) AS events,
  SUM(total_value) AS revenue
```

```
FROM events
WHERE __time >= CURRENT_TIMESTAMP - INTERVAL '24' HOUR
GROUP BY 1, 2
ORDER BY events DESC
LIMIT 100;

-- Approximate distinct count
SELECT
  APPROX_COUNT_DISTINCT(user_id) AS unique_users
FROM events
WHERE __time >= CURRENT_TIMESTAMP - INTERVAL '7' DAY;
```

20.4 Architecture

- **Coordinator**: Manages data availability
- **Overlord**: Controls ingestion tasks
- **Broker**: Routes queries
- **Historical**: Stores and queries historical data
- **MiddleManager**: Executes ingestion tasks
- **Router**: Optional query routing

20.5 URL

https://druid.apache.org
Documentation: https://druid.apache.org/docs/latest/

20.6 Books and Resources

- **Apache Druid Documentation** - comprehensive official guide
- **Druid Community Slack** - active community support

20.7 Hardening and Security

1. **TLS/SSL**: Enable for all services
2. **Authentication**: Basic auth, LDAP, or Kerberos
3. **Authorization**: Configure role-based access control
4. **Network Security**: Firewall all Druid ports
5. **Query Guards**: Set query timeout and row limits

Chapter 21

D3.js

21.1 What D3.js Is

D3.js (Data-Driven Documents) is the most powerful and flexible JavaScript library for creating data visualizations in web browsers. Unlike high-level charting libraries that provide pre-built chart types, D3 gives developers complete control over every visual element by binding arbitrary data to the Document Object Model (DOM) and applying data-driven transformations.

D3 has become the foundation for countless visualization libraries and is used by major news organizations, research institutions, and enterprises to create custom, interactive visualizations. Its approach of manipulating standard web technologies (HTML, SVG, CSS) means visualizations work across all modern browsers without plugins.

21.1.1 Core Concepts

D3's power comes from several key concepts:

- **Selections**: jQuery-like selection and manipulation of DOM elements
- **Data Binding**: Associating data arrays with DOM elements
- **Enter/Update/Exit**: Handling data changes elegantly
- **Scales**: Mapping data values to visual properties (position, color, size)
- **Axes**: Automatic generation of reference marks for scales
- **Transitions**: Animated interpolation between states
- **Shapes**: SVG path generators for lines, areas, arcs, etc.

- **Layouts**: Algorithms for hierarchies, networks, geographic projections

21.1.2 D3 Module Architecture

D3 v7 is organized as modular packages:

Module	Purpose
d3-selection	DOM selection and manipulation
d3-scale	Scale functions (linear, log, ordinal, etc.)
d3-axis	Axis generators
d3-shape	Line, area, arc, pie generators
d3-hierarchy	Tree, treemap, pack layouts
d3-force	Force-directed graph layouts
d3-geo	Geographic projections
d3-transition	Animated transitions
d3-fetch	Data loading (CSV, JSON, etc.)
d3-array	Array utilities and statistics

Table 21.1: D3.js core modules

21.2 Install Instructions

21.2.1 NPM Installation (Recommended for Production)

```
# Create project
mkdir d3-visualization && cd d3-visualization
npm init -y

# Install D3 (full bundle)
npm install d3

# Or install specific modules
npm install d3-selection d3-scale d3-axis d3-shape d3-
   array

# For TypeScript support
npm install --save-dev @types/d3
```

21.2.2 CDN Usage (Quick Prototyping)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>D3 Visualization</title>
  <script src="https://d3js.org/d3.v7.min.js"></script>
</head>
<body>
  <div id="chart"></div>
  <script src="app.js"></script>
</body>
</html>
```

21.2.3 ES Module Import

```
// Import entire D3
import * as d3 from 'd3';

// Or import specific modules (tree-shakeable)
import { select, selectAll } from 'd3-selection';
import { scaleLinear, scaleOrdinal } from 'd3-scale';
import { axisBottom, axisLeft } from 'd3-axis';
import { line, area } from 'd3-shape';
```

21.3 Working Example: Real-Time Streaming Dashboard

This example creates an interactive dashboard visualizing streaming data from a Kafka-backed API, demonstrating D3's capabilities for big data visualization.

21.3.1 Project Structure

```
d3-streaming-dashboard/
|-- index.html
|-- src/
|   |-- main.js
|   |-- charts/
|   |   |-- lineChart.js
|   |   |-- barChart.js
|   |   |-- heatmap.js
|   |-- utils/
|       |-- dataStream.js
|       |-- scales.js
|-- styles/
|   |-- dashboard.css
|-- package.json
```

21.3.2 HTML Structure

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,
     initial-scale=1">
  <title>Real-Time Analytics Dashboard</title>
  <link rel="stylesheet" href="styles/dashboard.css">
</head>
<body>
  <header>
    <h1>Streaming Analytics Dashboard</h1>
    <div id="connection-status">Disconnected</div>
  </header>

  <main class="dashboard-grid">
    <section class="chart-container" id="timeseries">
      <h2>Events Over Time</h2>
      <svg id="line-chart"></svg>
    </section>

    <section class="chart-container" id="distribution">
      <h2>Category Distribution</h2>
```

```
      <svg id="bar-chart"></svg>
    </section>

    <section class="chart-container" id="activity">
      <h2>Hourly Activity Heatmap</h2>
      <svg id="heatmap"></svg>
    </section>

    <section class="chart-container" id="metrics">
      <h2>Key Metrics</h2>
      <div class="metrics-grid">
        <div class="metric" id="events-per-sec">
          <span class="value">0</span>
          <span class="label">Events/sec</span>
        </div>
        <div class="metric" id="total-events">
          <span class="value">0</span>
          <span class="label">Total Events</span>
        </div>
        <div class="metric" id="unique-users">
          <span class="value">0</span>
          <span class="label">Unique Users</span>
        </div>
      </div>
    </section>
  </main>

  <script type="module" src="src/main.js"></script>
</body>
</html>
```

21.3.3 Real-Time Line Chart

```
// src/charts/lineChart.js
import * as d3 from 'd3';

export class RealTimeLineChart {
  constructor(selector, options = {}) {
    this.selector = selector;
    this.margin = options.margin || { top: 20, right:
       30, bottom: 30, left: 50 };
    this.maxPoints = options.maxPoints || 100;
```

```
    this.transitionDuration = options.transitionDuration
        || 500;
    this.data = [];

    this.init();
  }

  init() {
    const container = d3.select(this.selector);
    const bbox = container.node().getBoundingClientRect
       ();

    this.width = bbox.width - this.margin.left - this.
       margin.right;
    this.height = bbox.height - this.margin.top - this.
       margin.bottom;

    // Clear existing content
    container.selectAll('*').remove();

    // Create SVG
    this.svg = container
      .attr('width', this.width + this.margin.left +
         this.margin.right)
      .attr('height', this.height + this.margin.top +
         this.margin.bottom)
      .append('g')
      .attr('transform', `translate(${this.margin.left},
         ${this.margin.top})`);

    // Define scales
    this.xScale = d3.scaleTime()
      .range([0, this.width]);

    this.yScale = d3.scaleLinear()
      .range([this.height, 0]);

    // Define line generator
    this.line = d3.line()
      .x(d => this.xScale(d.timestamp))
      .y(d => this.yScale(d.value))
      .curve(d3.curveMonotoneX);
```

```
    // Define area generator for gradient fill
    this.area = d3.area()
      .x(d => this.xScale(d.timestamp))
      .y0(this.height)
      .y1(d => this.yScale(d.value))
      .curve(d3.curveMonotoneX);

    // Create gradient
    const gradient = this.svg.append('defs')
      .append('linearGradient')
      .attr('id', 'area-gradient')
      .attr('x1', '0%').attr('y1', '0%')
      .attr('x2', '0%').attr('y2', '100%');

    gradient.append('stop')
      .attr('offset', '0%')
      .attr('stop-color', '#4CAF50')
      .attr('stop-opacity', 0.3);

    gradient.append('stop')
      .attr('offset', '100%')
      .attr('stop-color', '#4CAF50')
      .attr('stop-opacity', 0);

    // Add clip path for smooth transitions
    this.svg.append('defs')
      .append('clipPath')
      .attr('id', 'clip')
      .append('rect')
      .attr('width', this.width)
      .attr('height', this.height);

    // Add axes
    this.xAxis = this.svg.append('g')
      .attr('class', 'x-axis')
      .attr('transform', `translate(0,${this.height})`);

    this.yAxis = this.svg.append('g')
      .attr('class', 'y-axis');

    // Add chart elements with clip path
    const chartGroup = this.svg.append('g')
      .attr('clip-path', 'url(#clip)');
```

```
    this.areaPath = chartGroup.append('path')
      .attr('class', 'area')
      .attr('fill', 'url(#area-gradient)');

    this.linePath = chartGroup.append('path')
      .attr('class', 'line')
      .attr('fill', 'none')
      .attr('stroke', '#4CAF50')
      .attr('stroke-width', 2);

    // Add tooltip
    this.tooltip = d3.select('body').append('div')
      .attr('class', 'd3-tooltip')
      .style('opacity', 0);

    // Add overlay for mouse tracking
    this.svg.append('rect')
      .attr('class', 'overlay')
      .attr('width', this.width)
      .attr('height', this.height)
      .attr('fill', 'none')
      .attr('pointer-events', 'all')
      .on('mousemove', (event) => this.handleMouseMove(
        event))
      .on('mouseout', () => this.handleMouseOut());

    // Focus elements for tooltip
    this.focus = this.svg.append('g')
      .attr('class', 'focus')
      .style('display', 'none');

    this.focus.append('circle')
      .attr('r', 5)
      .attr('fill', '#4CAF50');

    this.focus.append('line')
      .attr('class', 'x-hover-line')
      .attr('stroke', '#999')
      .attr('stroke-dasharray', '3,3')
      .attr('y1', 0)
      .attr('y2', this.height);
  }
```

```
update(newData) {
  // Add new data point
  this.data.push({
    timestamp: new Date(newData.timestamp),
    value: newData.value
  });

  // Remove old points if exceeding max
  while (this.data.length > this.maxPoints) {
    this.data.shift();
  }

  if (this.data.length < 2) return;

  // Update scales
  this.xScale.domain(d3.extent(this.data, d => d.
     timestamp));
  this.yScale.domain([0, d3.max(this.data, d => d.
     value) * 1.1]);

  // Update axes with transition
  this.xAxis.transition()
    .duration(this.transitionDuration)
    .call(d3.axisBottom(this.xScale).ticks(5));

  this.yAxis.transition()
    .duration(this.transitionDuration)
    .call(d3.axisLeft(this.yScale).ticks(5));

  // Update line and area
  this.linePath
    .datum(this.data)
    .transition()
    .duration(this.transitionDuration)
    .attr('d', this.line);

  this.areaPath
    .datum(this.data)
    .transition()
    .duration(this.transitionDuration)
    .attr('d', this.area);
}
```

```
handleMouseMove(event) {
  if (this.data.length === 0) return;

  this.focus.style('display', null);

  const [mouseX] = d3.pointer(event);
  const x0 = this.xScale.invert(mouseX);
  const bisect = d3.bisector(d => d.timestamp).left;
  const i = bisect(this.data, x0, 1);
  const d0 = this.data[i - 1];
  const d1 = this.data[i];

  if (!d0 || !d1) return;

  const d = x0 - d0.timestamp > d1.timestamp - x0 ? d1
      : d0;

  this.focus.attr('transform',
    `translate(${this.xScale(d.timestamp)},${this.
      yScale(d.value)})`);

  this.focus.select('.x-hover-line')
    .attr('y2', this.height - this.yScale(d.value));

  this.tooltip
    .style('opacity', 1)
    .html(`
      <strong>${d3.timeFormat('%H:%M:%S')(d.timestamp)
        }</strong><br/>
      Value: ${d.value.toLocaleString()}
    `)
    .style('left', (event.pageX + 15) + 'px')
    .style('top', (event.pageY - 28) + 'px');
}

handleMouseOut() {
  this.focus.style('display', 'none');
  this.tooltip.style('opacity', 0);
}

resize() {
  this.init();
```

```
    // Redraw with existing data
    this.data.forEach((d, i) => {
      if (i === this.data.length - 1) {
        this.update(d);
      }
    });
  }
}
```

21.3.4 Interactive Bar Chart

```
// src/charts/barChart.js
import * as d3 from 'd3';

export class InteractiveBarChart {
  constructor(selector, options = {}) {
    this.selector = selector;
    this.margin = options.margin || { top: 20, right:
       20, bottom: 40, left: 60 };
    this.colorScheme = options.colorScheme || d3.
       schemeCategory10;

    this.init();
  }

  init() {
    const container = d3.select(this.selector);
    const bbox = container.node().getBoundingClientRect
       ();

    this.width = bbox.width - this.margin.left - this.
       margin.right;
    this.height = bbox.height - this.margin.top - this.
       margin.bottom;

    container.selectAll('*').remove();

    this.svg = container
      .attr('width', this.width + this.margin.left +
         this.margin.right)
      .attr('height', this.height + this.margin.top +
         this.margin.bottom)
```

```
      .append('g')
      .attr('transform', `translate(${this.margin.left},
        ${this.margin.top})`);

    // Scales
    this.xScale = d3.scaleBand()
      .range([0, this.width])
      .padding(0.2);

    this.yScale = d3.scaleLinear()
      .range([this.height, 0]);

    this.colorScale = d3.scaleOrdinal(this.colorScheme);

    // Axes
    this.xAxis = this.svg.append('g')
      .attr('class', 'x-axis')
      .attr('transform', `translate(0,${this.height})`);

    this.yAxis = this.svg.append('g')
      .attr('class', 'y-axis');

    // Y-axis label
    this.svg.append('text')
      .attr('class', 'y-axis-label')
      .attr('transform', 'rotate(-90)')
      .attr('y', -this.margin.left + 15)
      .attr('x', -this.height / 2)
      .attr('text-anchor', 'middle')
      .text('Count');

    // Tooltip
    this.tooltip = d3.select('body').append('div')
      .attr('class', 'd3-tooltip')
      .style('opacity', 0);
  }

  update(data) {
    // data: [{ category: 'A', value: 100 }, ...]

    // Update scales
    this.xScale.domain(data.map(d => d.category));
    this.yScale.domain([0, d3.max(data, d => d.value) *
```

```
      1.1]);
    this.colorScale.domain(data.map(d => d.category));

    // Update axes
    this.xAxis.transition().duration(500)
      .call(d3.axisBottom(this.xScale))
      .selectAll('text')
      .attr('transform', 'rotate(-45)')
      .style('text-anchor', 'end');

    this.yAxis.transition().duration(500)
      .call(d3.axisLeft(this.yScale).ticks(5).tickFormat
        (d3.format('.2s')));

    // Data join
    const bars = this.svg.selectAll('.bar')
      .data(data, d => d.category);

    // Exit
    bars.exit()
      .transition().duration(300)
      .attr('y', this.height)
      .attr('height', 0)
      .remove();

    // Enter
    const barsEnter = bars.enter()
      .append('rect')
      .attr('class', 'bar')
      .attr('x', d => this.xScale(d.category))
      .attr('width', this.xScale.bandwidth())
      .attr('y', this.height)
      .attr('height', 0)
      .attr('fill', d => this.colorScale(d.category))
      .attr('rx', 4);

    // Update (merge enter + update)
    bars.merge(barsEnter)
      .on('mouseover', (event, d) => {
        d3.select(event.target)
          .transition().duration(100)
          .attr('opacity', 0.8)
          .attr('transform', 'scale(1.02)');
```

```
        this.tooltip
          .style('opacity', 1)
          .html(`
            <strong>${d.category}</strong><br/>
            Count: ${d.value.toLocaleString()}<br/>
            Percentage: ${((d.value / d3.sum(data, x =>
               x.value)) * 100).toFixed(1)}%
          `)
          .style('left', (event.pageX + 15) + 'px')
          .style('top', (event.pageY - 28) + 'px');
      })
      .on('mouseout', (event) => {
        d3.select(event.target)
          .transition().duration(100)
          .attr('opacity', 1)
          .attr('transform', 'scale(1)');

        this.tooltip.style('opacity', 0);
      })
      .on('click', (event, d) => {
        // Dispatch custom event for filtering
        const customEvent = new CustomEvent('
           categorySelected', {
          detail: { category: d.category }
        });
        document.dispatchEvent(customEvent);
      })
      .transition().duration(500)
      .attr('x', d => this.xScale(d.category))
      .attr('width', this.xScale.bandwidth())
      .attr('y', d => this.yScale(d.value))
      .attr('height', d => this.height - this.yScale(d.
         value))
      .attr('fill', d => this.colorScale(d.category));
  }
}
```

21.3.5 Activity Heatmap

```
// src/charts/heatmap.js
import * as d3 from 'd3';

export class ActivityHeatmap {
  constructor(selector, options = {}) {
    this.selector = selector;
    this.margin = options.margin || { top: 30, right:
       30, bottom: 30, left: 60 };
    this.days = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri
       ', 'Sat'];
    this.hours = d3.range(24);

    this.init();
  }

  init() {
    const container = d3.select(this.selector);
    const bbox = container.node().getBoundingClientRect
       ();

    this.width = bbox.width - this.margin.left - this.
       margin.right;
    this.height = bbox.height - this.margin.top - this.
       margin.bottom;

    container.selectAll('*').remove();

    this.svg = container
      .attr('width', this.width + this.margin.left +
         this.margin.right)
      .attr('height', this.height + this.margin.top +
         this.margin.bottom)
      .append('g')
      .attr('transform', `translate(${this.margin.left},
         ${this.margin.top})`);

    // Calculate cell dimensions
    this.cellWidth = this.width / 24;
    this.cellHeight = this.height / 7;

    // Color scale
```

```
    this.colorScale = d3.scaleSequential(d3.
        interpolateYlOrRd)
      .domain([0, 100]);

    // X axis (hours)
    this.svg.append('g')
      .attr('class', 'x-axis')
      .selectAll('text')
      .data(this.hours)
      .enter()
      .append('text')
      .attr('x', (d, i) => i * this.cellWidth + this.
          cellWidth / 2)
      .attr('y', -10)
      .attr('text-anchor', 'middle')
      .attr('font-size', '10px')
      .text(d => d + ':00');

    // Y axis (days)
    this.svg.append('g')
      .attr('class', 'y-axis')
      .selectAll('text')
      .data(this.days)
      .enter()
      .append('text')
      .attr('x', -10)
      .attr('y', (d, i) => i * this.cellHeight + this.
          cellHeight / 2)
      .attr('text-anchor', 'end')
      .attr('dominant-baseline', 'middle')
      .attr('font-size', '11px')
      .text(d => d);

    // Tooltip
    this.tooltip = d3.select('body').append('div')
      .attr('class', 'd3-tooltip')
      .style('opacity', 0);
  }

  update(data) {
    // data: [{ day: 0-6, hour: 0-23, value: number },
        ...]
```

```
    // Update color scale domain
    const maxValue = d3.max(data, d => d.value) || 100;
    this.colorScale.domain([0, maxValue]);

    // Data join for cells
    const cells = this.svg.selectAll('.cell')
      .data(data, d => `${d.day}-${d.hour}`);

    // Enter
    cells.enter()
      .append('rect')
      .attr('class', 'cell')
      .attr('x', d => d.hour * this.cellWidth)
      .attr('y', d => d.day * this.cellHeight)
      .attr('width', this.cellWidth - 2)
      .attr('height', this.cellHeight - 2)
      .attr('rx', 3)
      .attr('fill', '#eee')
      .on('mouseover', (event, d) => {
        this.tooltip
          .style('opacity', 1)
          .html(`
            <strong>${this.days[d.day]} ${d.hour}:00</
               strong><br/>
            Events: ${d.value.toLocaleString()}
          `)
          .style('left', (event.pageX + 15) + 'px')
          .style('top', (event.pageY - 28) + 'px');
      })
      .on('mouseout', () => {
        this.tooltip.style('opacity', 0);
      })
      .merge(cells)
      .transition().duration(300)
      .attr('fill', d => this.colorScale(d.value));

    // Exit
    cells.exit().remove();

    // Add/update legend
    this.updateLegend(maxValue);
  }
```

```
  updateLegend(maxValue) {
    // Remove existing legend
    this.svg.selectAll('.legend').remove();

    const legendWidth = 200;
    const legendHeight = 10;

    const legend = this.svg.append('g')
      .attr('class', 'legend')
      .attr('transform', `translate(${this.width -
        legendWidth}, ${this.height + 15})`);

    // Gradient
    const defs = this.svg.append('defs');
    const gradient = defs.append('linearGradient')
      .attr('id', 'heatmap-gradient');

    gradient.selectAll('stop')
      .data(d3.range(0, 1.1, 0.1))
      .enter()
      .append('stop')
      .attr('offset', d => `${d * 100}%`)
      .attr('stop-color', d => this.colorScale(d *
        maxValue));

    legend.append('rect')
      .attr('width', legendWidth)
      .attr('height', legendHeight)
      .attr('fill', 'url(#heatmap-gradient)');

    legend.append('text')
      .attr('x', 0)
      .attr('y', legendHeight + 12)
      .attr('font-size', '10px')
      .text('0');

    legend.append('text')
      .attr('x', legendWidth)
      .attr('y', legendHeight + 12)
      .attr('text-anchor', 'end')
      .attr('font-size', '10px')
      .text(maxValue.toLocaleString());
  }
```

```
}
```

21.3.6 WebSocket Data Stream

```
// src/utils/dataStream.js
export class DataStream {
  constructor(url, options = {}) {
    this.url = url;
    this.reconnectInterval = options.reconnectInterval
       || 3000;
    this.maxReconnectAttempts = options.
       maxReconnectAttempts || 10;
    this.reconnectAttempts = 0;
    this.listeners = new Map();
    this.ws = null;
  }

  connect() {
    return new Promise((resolve, reject) => {
      try {
        this.ws = new WebSocket(this.url);

        this.ws.onopen = () => {
          console.log('WebSocket connected');
          this.reconnectAttempts = 0;
          this.emit('connected', { status: 'connected'
             });
          resolve();
        };

        this.ws.onmessage = (event) => {
          try {
            const data = JSON.parse(event.data);
            this.emit('data', data);
          } catch (e) {
            console.error('Failed to parse message:', e)
               ;
          }
        };

        this.ws.onclose = () => {
          console.log('WebSocket disconnected');
```

```
        this.emit('disconnected', { status: '
           disconnected' });
        this.attemptReconnect();
      };

      this.ws.onerror = (error) => {
        console.error('WebSocket error:', error);
        this.emit('error', error);
        reject(error);
      };

    } catch (error) {
      reject(error);
    }
  });
}

attemptReconnect() {
  if (this.reconnectAttempts < this.
     maxReconnectAttempts) {
    this.reconnectAttempts++;
    console.log(`Reconnecting... attempt ${this.
       reconnectAttempts}`);
    setTimeout(() => this.connect(), this.
       reconnectInterval);
  } else {
    this.emit('maxReconnectAttemptsReached', {});
  }
}

on(event, callback) {
  if (!this.listeners.has(event)) {
    this.listeners.set(event, []);
  }
  this.listeners.get(event).push(callback);
  return () => this.off(event, callback);
}

off(event, callback) {
  const callbacks = this.listeners.get(event);
  if (callbacks) {
    const index = callbacks.indexOf(callback);
    if (index > -1) callbacks.splice(index, 1);
```

```
    }
  }

  emit(event, data) {
    const callbacks = this.listeners.get(event);
    if (callbacks) {
      callbacks.forEach(cb => cb(data));
    }
  }

  disconnect() {
    if (this.ws) {
      this.ws.close();
      this.ws = null;
    }
  }
}
```

21.3.7 Main Application

```
// src/main.js
import { RealTimeLineChart } from './charts/lineChart.js
  ';
import { InteractiveBarChart } from './charts/barChart.
  js';
import { ActivityHeatmap } from './charts/heatmap.js';
import { DataStream } from './utils/dataStream.js';

class Dashboard {
  constructor() {
    this.charts = {};
    this.metrics = { eventsPerSec: 0, totalEvents: 0,
       uniqueUsers: new Set() };
    this.categoryData = new Map();
    this.heatmapData = [];

    // Initialize heatmap data structure
    for (let day = 0; day < 7; day++) {
      for (let hour = 0; hour < 24; hour++) {
        this.heatmapData.push({ day, hour, value: 0 });
      }
    }
```

```
    this.init();
}

init() {
    // Initialize charts
    this.charts.lineChart = new RealTimeLineChart('#line
        -chart', {
        maxPoints: 50,
        transitionDuration: 300
    });

    this.charts.barChart = new InteractiveBarChart('#bar
        -chart');
    this.charts.heatmap = new ActivityHeatmap('#heatmap'
        );

    // Initialize data stream
    this.dataStream = new DataStream('wss://your-api.com
        /events');

    // Connection status updates
    this.dataStream.on('connected', () => {
        document.getElementById('connection-status').
            textContent = 'Connected';
        document.getElementById('connection-status').
            classList.add('connected');
    });

    this.dataStream.on('disconnected', () => {
        document.getElementById('connection-status').
            textContent = 'Disconnected';
        document.getElementById('connection-status').
            classList.remove('connected');
    });

    // Handle incoming data
    this.dataStream.on('data', (data) => this.handleData
        (data));

    // Category filter event
    document.addEventListener('categorySelected', (e) =>
        {
```

```
      console.log('Filter by category:', e.detail.
          category);
      // Implement filtering logic
    });

    // Handle window resize
    window.addEventListener('resize', () => this.
        handleResize());

    // Connect to data stream
    this.dataStream.connect().catch(err => {
      console.error('Failed to connect:', err);
      // Fall back to simulated data
      this.startSimulation();
    });

    // Start metrics update interval
    setInterval(() => this.updateMetrics(), 1000);
  }

  handleData(event) {
    const timestamp = new Date(event.timestamp);

    // Update line chart
    this.charts.lineChart.update({
      timestamp: timestamp,
      value: event.value || 1
    });

    // Update category counts
    const category = event.category || 'unknown';
    const currentCount = this.categoryData.get(category)
        || 0;
    this.categoryData.set(category, currentCount + 1);

    // Update bar chart
    const barData = Array.from(this.categoryData.entries
        ())
      .map(([category, value]) => ({ category, value }))
      .sort((a, b) => b.value - a.value)
      .slice(0, 10);
    this.charts.barChart.update(barData);
```

```
    // Update heatmap
    const day = timestamp.getDay();
    const hour = timestamp.getHours();
    const index = day * 24 + hour;
    this.heatmapData[index].value++;
    this.charts.heatmap.update(this.heatmapData);

    // Update metrics
    this.metrics.totalEvents++;
    if (event.userId) {
      this.metrics.uniqueUsers.add(event.userId);
    }
  }

  updateMetrics() {
    document.querySelector('#events-per-sec .value').
       textContent =
      this.metrics.eventsPerSec.toLocaleString();
    document.querySelector('#total-events .value').
       textContent =
      this.metrics.totalEvents.toLocaleString();
    document.querySelector('#unique-users .value').
       textContent =
      this.metrics.uniqueUsers.size.toLocaleString();

    // Reset events per second counter
    this.metrics.eventsPerSec = 0;
  }

  startSimulation() {
    // Simulate real-time data for demo
    const categories = ['pageview', 'click', 'purchase',
        'signup', 'logout'];

    setInterval(() => {
      const event = {
        timestamp: new Date().toISOString(),
        value: Math.floor(Math.random() * 100) + 10,
        category: categories[Math.floor(Math.random() *
           categories.length)],
        userId: `user_${Math.floor(Math.random() * 1000)
           }`
      };
```

```
      this.handleData(event);
      this.metrics.eventsPerSec++;
    }, 100);
  }

  handleResize() {
    Object.values(this.charts).forEach(chart => {
      if (chart.resize) chart.resize();
    });
  }
}

// Initialize dashboard when DOM is ready
document.addEventListener('DOMContentLoaded', () => {
  new Dashboard();
});
```

21.3.8 Dashboard Styles

```
/* styles/dashboard.css */
:root {
  --bg-primary: #1a1a2e;
  --bg-secondary: #16213e;
  --text-primary: #eee;
  --text-secondary: #aaa;
  --accent: #4CAF50;
  --border: #0f3460;
}

* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

body {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe
      UI', Roboto, sans-serif;
  background: var(--bg-primary);
  color: var(--text-primary);
  min-height: 100vh;
```

```
}

header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 1rem 2rem;
  background: var(--bg-secondary);
  border-bottom: 1px solid var(--border);
}

#connection-status {
  padding: 0.5rem 1rem;
  border-radius: 20px;
  background: #e74c3c;
  font-size: 0.85rem;
}

#connection-status.connected {
  background: var(--accent);
}

.dashboard-grid {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  gap: 1.5rem;
  padding: 1.5rem;
}

.chart-container {
  background: var(--bg-secondary);
  border: 1px solid var(--border);
  border-radius: 8px;
  padding: 1rem;
}

.chart-container h2 {
  font-size: 1rem;
  color: var(--text-secondary);
  margin-bottom: 1rem;
}

.chart-container svg {
```

```
  width: 100%;
  height: 300px;
}

/* D3 Tooltip */
.d3-tooltip {
  position: absolute;
  background: rgba(0, 0, 0, 0.9);
  color: white;
  padding: 10px 15px;
  border-radius: 6px;
  font-size: 13px;
  pointer-events: none;
  z-index: 1000;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.3);
}

/* Axis styling */
.x-axis, .y-axis {
  font-size: 11px;
  color: var(--text-secondary);
}

.x-axis path, .y-axis path,
.x-axis line, .y-axis line {
  stroke: var(--border);
}

/* Metrics grid */
.metrics-grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 1rem;
}

.metric {
  text-align: center;
  padding: 1.5rem;
  background: var(--bg-primary);
  border-radius: 8px;
}

.metric .value {
```

```
  display: block;
  font-size: 2rem;
  font-weight: bold;
  color: var(--accent);
}

.metric .label {
  font-size: 0.85rem;
  color: var(--text-secondary);
}

/* Responsive */
@media (max-width: 1024px) {
  .dashboard-grid {
    grid-template-columns: 1fr;
  }
}
```

21.4 Integration with Big Data Stack

21.4.1 Fetching Data from Apache Druid

```
// Druid SQL query via REST API
async function queryDruid(sql) {
  const response = await fetch('http://druid-broker
     :8082/druid/v2/sql', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      query: sql,
      resultFormat: 'object'
    })
  });

  return response.json();
}

// Time series data for line chart
const timeseriesData = await queryDruid(`
```

```
  SELECT
    TIME_FLOOR(__time, 'PT1M') AS minute,
    COUNT(*) AS events
  FROM events
  WHERE __time >= CURRENT_TIMESTAMP - INTERVAL '1' HOUR
  GROUP BY 1
  ORDER BY 1
');
```

21.4.2 Server-Sent Events from Kafka

```
// Python backend with FastAPI
// backend/sse_endpoint.py
from fastapi import FastAPI
from fastapi.responses import StreamingResponse
from kafka import KafkaConsumer
import json

app = FastAPI()

async def kafka_event_generator():
    consumer = KafkaConsumer(
        'events',
        bootstrap_servers=['kafka:9092'],
        value_deserializer=lambda m: json.loads(m.decode
           ('utf-8'))
    )

    for message in consumer:
        yield f"data: {json.dumps(message.value)}\n\n"

@app.get("/stream")
async def stream_events():
    return StreamingResponse(
        kafka_event_generator(),
        media_type="text/event-stream"
    )
```

```
// Frontend SSE client
const eventSource = new EventSource('/stream');

eventSource.onmessage = (event) => {
```

```
  const data = JSON.parse(event.data);
  dashboard.handleData(data);
};
```

21.5 URL

https://d3js.org

21.6 Books

- **Interactive Data Visualization for the Web** by Scott Murray (O'Reilly) – The definitive guide to D3.js
- **D3.js in Action** by Elijah Meeks (Manning) – Advanced D3 techniques and patterns
- **Data Visualization with D3.js Cookbook** by Nick Qi Zhu (Packt) – Practical recipes
- **Fullstack D3 and Data Visualization** by Amelia Wattenberger – Modern D3 v6+ approaches

21.7 Hardening

21.7.1 Input Sanitization

```
// Sanitize user-provided data before rendering
function sanitizeData(data) {
  return data.map(item => ({
    // Sanitize strings to prevent XSS
    label: DOMPurify.sanitize(String(item.label)),
    // Ensure numeric values
    value: Number(item.value) || 0,
    // Validate dates
    timestamp: new Date(item.timestamp).getTime() ?
               new Date(item.timestamp) : new Date()
  }));
}
```

```
// Use text() instead of html() for user data
selection.text(d => d.label);  // Safe
selection.html(d => d.label);  // Dangerous with
   untrusted data
```

21.7.2 Content Security Policy

```
# Nginx CSP headers for D3 applications
add_header Content-Security-Policy "
  default-src 'self';
  script-src 'self' 'unsafe-eval' https://d3js.org;
  style-src 'self' 'unsafe-inline';
  img-src 'self' data: blob:;
  connect-src 'self' wss: https:;
" always;
```

21.7.3 Performance Optimization

```
// Use Canvas for large datasets (>10,000 points)
function renderWithCanvas(data) {
  const canvas = document.getElementById('canvas');
  const ctx = canvas.getContext('2d');
  const dpr = window.devicePixelRatio || 1;

  // Handle high-DPI displays
  canvas.width = width * dpr;
  canvas.height = height * dpr;
  ctx.scale(dpr, dpr);

  // Batch render points
  ctx.beginPath();
  data.forEach((d, i) => {
    const x = xScale(d.x);
    const y = yScale(d.y);
    if (i === 0) {
      ctx.moveTo(x, y);
    } else {
      ctx.lineTo(x, y);
    }
  });
  ctx.stroke();
```

```
}

// Data decimation for large time series
function decimateData(data, maxPoints) {
  if (data.length <= maxPoints) return data;

  const step = Math.ceil(data.length / maxPoints);
  return data.filter((_, i) => i % step === 0);
}
```

Chapter 22

Apache ECharts

22.1 What ECharts Is

Apache ECharts is a powerful, interactive charting and data visualization library originally developed by Baidu and now an Apache Software Foundation top-level project. Unlike D3.js which provides low-level primitives, ECharts offers a comprehensive set of pre-built chart types with declarative configuration, making it ideal for rapid dashboard development.

ECharts is particularly well-suited for big data visualization scenarios, offering features like progressive rendering for large datasets, WebGL acceleration for massive point clouds, and built-in support for real-time data updates. Its declarative option-based configuration makes it accessible to developers who need powerful visualizations without learning a visualization grammar.

22.1.1 Key Features

- **Rich Chart Types**: 20+ chart types including line, bar, pie, scatter, candlestick, heatmap, tree, sunburst, sankey, graph, gauge, funnel, and more
- **Dual Rendering**: Canvas (default) for performance, SVG for accessibility and printing
- **Large Data Support**: Progressive rendering handles millions of data points
- **WebGL Acceleration**: GL extension for 3D charts and massive scatter plots

- **Responsive Design**: Auto-resize and mobile-friendly touch interactions
- **Animations**: Smooth transitions and morphing between chart types
- **Internationalization**: Built-in support for multiple languages
- **Accessibility**: ARIA support and keyboard navigation

22.1.2 Architecture

ECharts uses a layered architecture:

Layer	Purpose
Option	Declarative chart configuration
Component	Axis, legend, tooltip, toolbox, etc.
Series	Data series and chart types
Renderer	Canvas/SVG/WebGL rendering
zrender	2D drawing library foundation

Table 22.1: ECharts architecture layers

22.2 Install Instructions

22.2.1 NPM Installation

```
# Create project
mkdir echarts-dashboard && cd echarts-dashboard
npm init -y

# Install ECharts
npm install echarts

# For TypeScript support
npm install --save-dev @types/echarts

# Optional: WebGL extension for large datasets
npm install echarts-gl
```

22.2.2 CDN Usage

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>ECharts Dashboard</title>
  <!-- Full bundle -->
  <script src="https://cdn.jsdelivr.net/npm/echarts@5/
     dist/echarts.min.js"></script>

  <!-- Or minimal bundle with required components -->
  <script src="https://cdn.jsdelivr.net/npm/echarts@5/
     dist/echarts.simple.min.js"></script>
</head>
<body>
  <div id="chart" style="width: 800px; height: 600px;"><
     /div>
</body>
</html>
```

22.2.3 ES Module Import (Tree-Shakeable)

```
// Import only what you need for smaller bundles
import * as echarts from 'echarts/core';

// Import required components
import {
  TitleComponent,
  TooltipComponent,
  GridComponent,
  LegendComponent,
  DataZoomComponent
} from 'echarts/components';

// Import chart types
import {
  LineChart,
  BarChart,
  PieChart,
  ScatterChart
} from 'echarts/charts';
```

```
// Import renderer
import { CanvasRenderer } from 'echarts/renderers';
// Or for SVG: import { SVGRenderer } from 'echarts/
   renderers';

// Register components
echarts.use([
  TitleComponent,
  TooltipComponent,
  GridComponent,
  LegendComponent,
  DataZoomComponent,
  LineChart,
  BarChart,
  PieChart,
  ScatterChart,
  CanvasRenderer
]);
```

22.3 Working Example: Real-Time Analytics Dashboard

This example creates a comprehensive analytics dashboard with multiple chart types, real-time updates, and interactivity.

22.3.1 Project Structure

```
echarts-analytics/
|-- index.html
|-- src/
|   |-- main.ts
|   |-- charts/
|   |   |-- timeSeriesChart.ts
|   |   |-- distributionChart.ts
|   |   |-- geoMap.ts
|   |   |-- gaugeCluster.ts
|   |-- services/
|   |   |-- dataService.ts
|   |-- types/
```

```
|       |-- analytics.ts
|-- styles/
|   |-- dashboard.css
|-- package.json
|-- tsconfig.json
```

22.3.2 TypeScript Types

```typescript
// src/types/analytics.ts
export interface TimeSeriesPoint {
  timestamp: Date;
  value: number;
  category?: string;
}

export interface CategoryData {
  name: string;
  value: number;
}

export interface GeoPoint {
  name: string;
  coords: [number, number];  // [longitude, latitude]
  value: number;
}

export interface MetricGauge {
  name: string;
  value: number;
  min: number;
  max: number;
  thresholds: { value: number; color: string }[];
}
```

22.3.3 Time Series Chart with Multiple Series

```
// src/charts/timeSeriesChart.ts
import * as echarts from 'echarts';
import type { EChartsOption, ECharts } from 'echarts';
import type { TimeSeriesPoint } from '../types/analytics
  ';

export class TimeSeriesChart {
  private chart: ECharts;
  private data: Map<string, TimeSeriesPoint[]> = new Map
    ();
  private maxPoints: number;

  constructor(container: HTMLElement, maxPoints = 100) {
    this.chart = echarts.init(container, 'dark');
    this.maxPoints = maxPoints;
    this.initChart();

    // Handle resize
    window.addEventListener('resize', () => this.chart.
      resize());
  }

  private initChart(): void {
    const option: EChartsOption = {
      backgroundColor: 'transparent',
      title: {
        text: 'Real-Time Metrics',
        left: 'center',
        textStyle: { color: '#ccc' }
      },
      tooltip: {
        trigger: 'axis',
        axisPointer: {
          type: 'cross',
          animation: false,
          label: { backgroundColor: '#505765' }
        }
      },
      legend: {
        data: [],
        bottom: 10,
```

```
      textStyle: { color: '#ccc' }
    },
    toolbox: {
      feature: {
        dataZoom: { yAxisIndex: 'none' },
        restore: {},
        saveAsImage: {}
      },
      right: 20
    },
    dataZoom: [
      {
        type: 'inside',
        start: 0,
        end: 100
      },
      {
        type: 'slider',
        start: 0,
        end: 100,
        height: 20,
        bottom: 40
      }
    ],
    grid: {
      left: '3%',
      right: '4%',
      bottom: '15%',
      containLabel: true
    },
    xAxis: {
      type: 'time',
      splitLine: { show: false },
      axisLine: { lineStyle: { color: '#444' } },
      axisLabel: { color: '#aaa' }
    },
    yAxis: {
      type: 'value',
      splitLine: { lineStyle: { color: '#333' } },
      axisLine: { lineStyle: { color: '#444' } },
      axisLabel: { color: '#aaa' }
    },
    series: []
```

```
    };

    this.chart.setOption(option);
  }

  addSeries(name: string, color: string): void {
    this.data.set(name, []);

    const series = {
      name,
      type: 'line',
      smooth: true,
      symbol: 'none',
      sampling: 'lttb',  // Largest-Triangle-Three-
         Buckets downsampling
      lineStyle: { width: 2, color },
      areaStyle: {
        color: new echarts.graphic.LinearGradient(0, 0,
           0, 1, [
          { offset: 0, color: color + '80' },
          { offset: 1, color: color + '10' }
        ])
      },
      data: []
    };

    this.chart.setOption({
      legend: { data: Array.from(this.data.keys()) },
      series: [series]
    }, { notMerge: false });
  }

  updateData(seriesName: string, point: TimeSeriesPoint)
     : void {
    const seriesData = this.data.get(seriesName);
    if (!seriesData) return;

    seriesData.push(point);

    // Remove old points
    while (seriesData.length > this.maxPoints) {
      seriesData.shift();
    }
```

```
    // Find series index
    const option = this.chart.getOption() as
       EChartsOption;
    const series = option.series as any[];
    const seriesIndex = series.findIndex(s => s.name ===
        seriesName);

    if (seriesIndex !== -1) {
      this.chart.setOption({
        series: [{
          data: seriesData.map(p => [p.timestamp.getTime
             (), p.value])
        }]
      }, {
        seriesIndex,
        notMerge: true,
        lazyUpdate: true
      });
    }
  }

  // Batch update for performance
  batchUpdate(updates: Map<string, TimeSeriesPoint[]>):
     void {
    const seriesUpdates: any[] = [];

    updates.forEach((points, seriesName) => {
      const seriesData = this.data.get(seriesName);
      if (!seriesData) return;

      seriesData.push(...points);
      while (seriesData.length > this.maxPoints) {
        seriesData.shift();
      }

      seriesUpdates.push({
        name: seriesName,
        data: seriesData.map(p => [p.timestamp.getTime()
           , p.value])
      });
    });
```

```
    this.chart.setOption({ series: seriesUpdates });
  }

  dispose(): void {
    this.chart.dispose();
  }
}
```

22.3.4 Distribution Chart (Pie/Sunburst)

```
// src/charts/distributionChart.ts
import * as echarts from 'echarts';
import type { EChartsOption, ECharts } from 'echarts';
import type { CategoryData } from '../types/analytics';

export class DistributionChart {
  private chart: ECharts;
  private chartType: 'pie' | 'sunburst' = 'pie';

  constructor(container: HTMLElement) {
    this.chart = echarts.init(container, 'dark');
    this.initChart();

    window.addEventListener('resize', () => this.chart.
       resize());
  }

  private initChart(): void {
    const option: EChartsOption = {
      backgroundColor: 'transparent',
      title: {
        text: 'Category Distribution',
        left: 'center',
        textStyle: { color: '#ccc' }
      },
      tooltip: {
        trigger: 'item',
        formatter: '{b}: {c} ({d}%)'
      },
      legend: {
        orient: 'vertical',
        left: 'left',
```

```
        top: 'middle',
        textStyle: { color: '#ccc' }
      },
      series: [{
        name: 'Distribution',
        type: 'pie',
        radius: ['40%', '70%'],  // Donut chart
        center: ['60%', '50%'],
        avoidLabelOverlap: true,
        itemStyle: {
          borderRadius: 10,
          borderColor: '#1a1a2e',
          borderWidth: 2
        },
        label: {
          show: true,
          formatter: '{b}\n{d}%',
          color: '#ccc'
        },
        emphasis: {
          label: {
            show: true,
            fontSize: 16,
            fontWeight: 'bold'
          },
          itemStyle: {
            shadowBlur: 10,
            shadowOffsetX: 0,
            shadowColor: 'rgba(0, 0, 0, 0.5)'
          }
        },
        data: []
      }]
    };

    this.chart.setOption(option);
  }

  updateData(data: CategoryData[]): void {
    // Sort by value and take top 10
    const sortedData = [...data]
      .sort((a, b) => b.value - a.value)
      .slice(0, 10);
```

```
    // Color palette
    const colors = [
      '#5470c6', '#91cc75', '#fac858', '#ee6666', '#73
         c0de',
      '#3ba272', '#fc8452', '#9a60b4', '#ea7ccc', '#48
         b8d0'
    ];

    const chartData = sortedData.map((item, index) => ({
      name: item.name,
      value: item.value,
      itemStyle: { color: colors[index % colors.length]
         }
    }));

    this.chart.setOption({
      legend: { data: sortedData.map(d => d.name) },
      series: [{ data: chartData }]
    });
  }

  // Switch between pie and sunburst
  toggleChartType(): void {
    this.chartType = this.chartType === 'pie' ? '
       sunburst' : 'pie';

    if (this.chartType === 'sunburst') {
      // Convert flat data to hierarchical for sunburst
      this.chart.setOption({
        series: [{
          type: 'sunburst',
          radius: ['15%', '80%'],
          label: { rotate: 'radial' }
        }]
      });
    } else {
      this.initChart();
    }
  }

  dispose(): void {
    this.chart.dispose();
```

```
  }
}
```

22.3.5 Gauge Cluster for KPIs

```
// src/charts/gaugeCluster.ts
import * as echarts from 'echarts';
import type { EChartsOption, ECharts } from 'echarts';
import type { MetricGauge } from '../types/analytics';

export class GaugeCluster {
  private chart: ECharts;

  constructor(container: HTMLElement) {
    this.chart = echarts.init(container, 'dark');
    window.addEventListener('resize', () => this.chart.
       resize());
  }

  updateGauges(gauges: MetricGauge[]): void {
    const cols = Math.ceil(Math.sqrt(gauges.length));
    const rows = Math.ceil(gauges.length / cols);

    const series = gauges.map((gauge, index) => {
      const col = index % cols;
      const row = Math.floor(index / cols);

      const centerX = (col + 0.5) / cols * 100;
      const centerY = (row + 0.5) / rows * 100;

      return {
        type: 'gauge',
        center: [`${centerX}%`, `${centerY}%`],
        radius: `${Math.min(40 / cols, 40 / rows)}%`,
        startAngle: 200,
        endAngle: -20,
        min: gauge.min,
        max: gauge.max,
        splitNumber: 5,
        title: {
          offsetCenter: [0, '70%'],
          fontSize: 12,
```

```
        color: '#ccc'
      },
      detail: {
        fontSize: 20,
        offsetCenter: [0, '40%'],
        valueAnimation: true,
        formatter: (value: number) => value.toFixed(1)
          ,
        color: 'inherit'
      },
      axisLine: {
        lineStyle: {
          width: 10,
          color: gauge.thresholds.map((t, i, arr) => [
            (t.value - gauge.min) / (gauge.max - gauge
              .min),
            t.color
          ])
        }
      },
      axisTick: { show: false },
      splitLine: {
        length: 10,
        lineStyle: { color: '#444' }
      },
      axisLabel: {
        distance: 15,
        color: '#aaa',
        fontSize: 10
      },
      pointer: {
        width: 4,
        length: '60%',
        itemStyle: { color: 'auto' }
      },
      data: [{
        value: gauge.value,
        name: gauge.name
      }]
    };
  });

  this.chart.setOption({
```

```
      backgroundColor: 'transparent',
      series
    });
  }

  // Animate value change
  animateValue(gaugeName: string, newValue: number):
     void {
    const option = this.chart.getOption() as
       EChartsOption;
    const series = option.series as any[];

    const seriesIndex = series.findIndex(
      s => s.data && s.data[0]?.name === gaugeName
    );

    if (seriesIndex !== -1) {
      this.chart.setOption({
        series: [{
          data: [{ value: newValue, name: gaugeName }]
        }]
      }, { seriesIndex });
    }
  }

  dispose(): void {
    this.chart.dispose();
  }
}
```

22.3.6 Main Dashboard Application

```
// src/main.ts
import * as echarts from 'echarts';
import { TimeSeriesChart } from './charts/
   timeSeriesChart';
import { DistributionChart } from './charts/
   distributionChart';
import { GaugeCluster } from './charts/gaugeCluster';

// Register dark theme
echarts.registerTheme('dark', {
```

```
  backgroundColor: 'transparent',
  textStyle: { color: '#ccc' },
  title: { textStyle: { color: '#eee' } }
});

class Dashboard {
  private timeSeriesChart: TimeSeriesChart;
  private distributionChart: DistributionChart;
  private gaugeCluster: GaugeCluster;
  private eventSource: EventSource | null = null;

  constructor() {
    // Initialize charts
    this.timeSeriesChart = new TimeSeriesChart(
      document.getElementById('timeseries-chart')!
    );

    this.distributionChart = new DistributionChart(
      document.getElementById('distribution-chart')!
    );

    this.gaugeCluster = new GaugeCluster(
      document.getElementById('gauge-cluster')!
    );

    // Add series to time series chart
    this.timeSeriesChart.addSeries('requests', '#5470c6'
       );
    this.timeSeriesChart.addSeries('errors', '#ee6666');
    this.timeSeriesChart.addSeries('latency', '#91cc75')
       ;

    // Initialize gauges
    this.gaugeCluster.updateGauges([
      {
        name: 'CPU',
        value: 45,
        min: 0,
        max: 100,
        thresholds: [
          { value: 60, color: '#91cc75' },
          { value: 80, color: '#fac858' },
          { value: 100, color: '#ee6666' }
```

```
        ]
      },
      {
        name: 'Memory',
        value: 62,
        min: 0,
        max: 100,
        thresholds: [
          { value: 70, color: '#91cc75' },
          { value: 85, color: '#fac858' },
          { value: 100, color: '#ee6666' }
        ]
      },
      {
        name: 'Disk I/O',
        value: 230,
        min: 0,
        max: 500,
        thresholds: [
          { value: 300, color: '#91cc75' },
          { value: 400, color: '#fac858' },
          { value: 500, color: '#ee6666' }
        ]
      },
      {
        name: 'Network',
        value: 856,
        min: 0,
        max: 1000,
        thresholds: [
          { value: 600, color: '#91cc75' },
          { value: 800, color: '#fac858' },
          { value: 1000, color: '#ee6666' }
        ]
      }
    ]);

    this.connectToDataSource();
  }

  private connectToDataSource(): void {
    // Connect to Server-Sent Events endpoint
    try {
```

```
      this.eventSource = new EventSource('/api/metrics/
          stream');

      this.eventSource.onmessage = (event) => {
        const data = JSON.parse(event.data);
        this.handleMetricsUpdate(data);
      };

      this.eventSource.onerror = () => {
        console.log('SSE connection lost, starting
            simulation');
        this.startSimulation();
      };
    } catch (e) {
      this.startSimulation();
    }
  }

  private handleMetricsUpdate(data: any): void {
    const timestamp = new Date(data.timestamp);

    // Update time series
    this.timeSeriesChart.updateData('requests', {
      timestamp,
      value: data.requestsPerSecond
    });
    this.timeSeriesChart.updateData('errors', {
      timestamp,
      value: data.errorsPerSecond
    });
    this.timeSeriesChart.updateData('latency', {
      timestamp,
      value: data.avgLatencyMs
    });

    // Update distribution
    if (data.categoryBreakdown) {
      this.distributionChart.updateData(
        Object.entries(data.categoryBreakdown).map(([
            name, value]) => ({
          name,
          value: value as number
        }))
```

```
      );
    }

    // Update gauges
    if (data.systemMetrics) {
      this.gaugeCluster.animateValue('CPU', data.
          systemMetrics.cpu);
      this.gaugeCluster.animateValue('Memory', data.
          systemMetrics.memory);
      this.gaugeCluster.animateValue('Disk I/O', data.
          systemMetrics.diskIO);
      this.gaugeCluster.animateValue('Network', data.
          systemMetrics.network);
    }
  }

  private startSimulation(): void {
    const categories = ['API', 'Web', 'Mobile', 'IoT', '
        Batch'];
    const categoryValues = new Map(categories.map(c => [
        c, Math.random() * 1000]));

    setInterval(() => {
      const now = new Date();

      // Simulate metrics
      this.timeSeriesChart.updateData('requests', {
        timestamp: now,
        value: 800 + Math.random() * 400
      });
      this.timeSeriesChart.updateData('errors', {
        timestamp: now,
        value: Math.random() * 20
      });
      this.timeSeriesChart.updateData('latency', {
        timestamp: now,
        value: 50 + Math.random() * 100
      });

      // Update category values
      categories.forEach(cat => {
        const current = categoryValues.get(cat)!;
        categoryValues.set(cat, Math.max(0, current + (
```

```
            Math.random() - 0.5) * 100));
      });

      this.distributionChart.updateData(
        Array.from(categoryValues.entries()).map(([name,
            value]) => ({ name, value }))
      );

      // Update gauges with some variation
      this.gaugeCluster.animateValue('CPU', 30 + Math.
         random() * 40);
      this.gaugeCluster.animateValue('Memory', 50 + Math
         .random() * 30);
      this.gaugeCluster.animateValue('Disk I/O', 150 +
         Math.random() * 200);
      this.gaugeCluster.animateValue('Network', 600 +
         Math.random() * 300);
    }, 1000);
  }
}

// Initialize
document.addEventListener('DOMContentLoaded', () => {
  new Dashboard();
});
```

22.3.7 HTML Dashboard Layout

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,
     initial-scale=1">
  <title>ECharts Analytics Dashboard</title>
  <style>
    * { box-sizing: border-box; margin: 0; padding: 0; }
    body {
      font-family: -apple-system, BlinkMacSystemFont,
         sans-serif;
      background: #1a1a2e;
      color: #eee;
```

```
      min-height: 100vh;
    }
    .dashboard {
      display: grid;
      grid-template-columns: 2fr 1fr;
      grid-template-rows: 1fr 1fr;
      gap: 1rem;
      padding: 1rem;
      height: 100vh;
    }
    .chart-panel {
      background: #16213e;
      border-radius: 8px;
      padding: 1rem;
      overflow: hidden;
    }
    #timeseries-chart { grid-row: span 2; height: 100%;
       }
    #distribution-chart, #gauge-cluster { height: 100%;
       }
  </style>
</head>
<body>
  <div class="dashboard">
    <div class="chart-panel">
      <div id="timeseries-chart" style="width:100%;
         height:100%"></div>
    </div>
    <div class="chart-panel">
      <div id="distribution-chart" style="width:100%;
         height:100%"></div>
    </div>
    <div class="chart-panel">
      <div id="gauge-cluster" style="width:100%;height
         :100%"></div>
    </div>
  </div>
  <script type="module" src="src/main.ts"></script>
</body>
</html>
```

22.4 Large Dataset Handling

22.4.1 Progressive Rendering

```
// Handle millions of points with progressive rendering
const option = {
  dataset: {
    source: largeDataArray,  // Can be millions of
       points
    dimensions: ['timestamp', 'value']
  },
  series: [{
    type: 'line',
    progressive: 400,           // Render 400 points per
        frame
    progressiveThreshold: 3000, // Enable when > 3000
       points
    progressiveChunkMode: 'mod',
    sampling: 'lttb',           // Downsample for
       display
    large: true,
    largeThreshold: 2000
  }]
};
```

22.4.2 WebGL for Massive Scatter Plots

```
// Using echarts-gl for millions of points
import 'echarts-gl';

const option = {
  grid3D: {},
  xAxis3D: { type: 'value' },
  yAxis3D: { type: 'value' },
  zAxis3D: { type: 'value' },
  series: [{
    type: 'scatter3D',
    data: massivePointCloud,  // Millions of [x, y, z]
       points
    symbolSize: 2,
    itemStyle: {
```

```
      opacity: 0.8
    }
  }]
};
```

22.5 Integration with Big Data Stack

22.5.1 Apache Druid Integration

```
// Fetch aggregated data from Druid for ECharts
async function fetchDruidTimeSeries(
  metric: string,
  granularity: string,
  hours: number
): Promise<any[]> {
  const query = {
    queryType: 'timeseries',
    dataSource: 'events',
    granularity,
    intervals: [`PT${hours}H/now`],
    aggregations: [
      { type: 'count', name: 'count' },
      { type: 'doubleSum', fieldName: metric, name: '
         value' }
    ]
  };

  const response = await fetch('http://druid:8082/druid/
     v2', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(query)
  });

  const data = await response.json();

  // Transform for ECharts
  return data.map((item: any) => [
    new Date(item.timestamp).getTime(),
    item.result.value
  ]);
```

```
}

// Update chart with Druid data
const druidData = await fetchDruidTimeSeries('revenue',
   'minute', 24);
chart.setOption({
  series: [{ data: druidData }]
});
```

22.6 URL

https://echarts.apache.org

22.7 Books

- **Apache ECharts Official Documentation** – Comprehensive API reference and examples
- **Data Visualization with ECharts** (Chinese) – In-depth guide to ECharts features
- **ECharts Gallery** (https://echarts.apache.org/examples/) – Extensive example collection

22.8 Hardening

22.8.1 Data Validation

```
// Validate and sanitize chart data
function sanitizeChartData(data: any[]): any[] {
  return data
    .filter(item => {
      // Ensure valid data structure
      if (!item || typeof item !== 'object') return
         false;
      if (item.value !== undefined && typeof item.value
         !== 'number') return false;
      if (!isFinite(item.value)) return false;
      return true;
```

```
    })
    .map(item => ({
      ...item,
      // Sanitize string fields
      name: item.name ? String(item.name).slice(0, 100)
        : '',
      // Clamp numeric values
      value: Math.max(-1e15, Math.min(1e15, item.value))
    }));
}
```

22.8.2 Memory Management

```
// Proper cleanup to prevent memory leaks
class ChartManager {
  private charts: Map<string, ECharts> = new Map();

  create(id: string, container: HTMLElement): ECharts {
    // Dispose existing chart if any
    this.dispose(id);

    const chart = echarts.init(container);
    this.charts.set(id, chart);
    return chart;
  }

  dispose(id: string): void {
    const chart = this.charts.get(id);
    if (chart) {
      chart.dispose();
      this.charts.delete(id);
    }
  }

  disposeAll(): void {
    this.charts.forEach(chart => chart.dispose());
    this.charts.clear();
  }
}

// Clean up on page unload
window.addEventListener('beforeunload', () => {
```

```
  chartManager.disposeAll();
});
```

22.8.3 Performance Best Practices

```
// Batch updates for better performance
chart.setOption(newOption, {
  notMerge: false,        // Merge with existing option
  lazyUpdate: true,       // Delay rendering
  silent: true            // Don't trigger events
});

// Use appendData for streaming scenarios
chart.appendData({
  seriesIndex: 0,
  data: newPoints
});

// Throttle rapid updates
import { throttle } from 'lodash';

const throttledUpdate = throttle((data) => {
  chart.setOption({ series: [{ data }] });
}, 100);
```

Chapter 23

FINOS Perspective

23.1 What Perspective Is

FINOS Perspective is an interactive analytics and data visualization component originally developed by J.P. Morgan and now maintained by the Fintech Open Source Foundation (FINOS). It's specifically designed for streaming data visualization in financial services applications, offering exceptional performance through WebAssembly (WASM) and Apache Arrow integration.

Perspective excels at handling high-frequency data updates typical in trading systems, providing pivot table functionality, cross-filtering, and multiple visualization types with sub-millisecond update latency. Its architecture allows it to handle millions of rows while maintaining interactive performance.

23.1.1 Key Features

- **WebAssembly Engine**: C++ core compiled to WASM for near-native performance
- **Apache Arrow**: Native Arrow columnar format for efficient data transfer
- **Streaming Updates**: Designed for real-time data with minimal latency
- **Pivot Tables**: Interactive pivot, group-by, split-by, and aggregation
- **Multiple Views**: Datagrid, charts, treemaps, and custom plugins
- **Expression Columns**: Computed columns using expression language

- **Cross-Filtering**: Click-to-filter across multiple views
- **Server Mode**: Python/Node.js server for large dataset processing

23.1.2 Architecture

Component	Purpose
perspective	Core engine (C++/WASM)
perspective-viewer	Web component UI
perspective-viewer-datagrid	High-performance data grid
perspective-viewer-d3fc	D3FC-based charts
perspective-python	Python bindings
perspective-jupyterlab	JupyterLab integration

Table 23.1: FINOS Perspective components

23.2 Install Instructions

23.2.1 JavaScript/NPM Installation

```
# Create project
mkdir perspective-dashboard && cd perspective-dashboard
npm init -y

# Install Perspective packages
npm install @finos/perspective
npm install @finos/perspective-viewer
npm install @finos/perspective-viewer-datagrid
npm install @finos/perspective-viewer-d3fc

# For Webpack bundling
npm install @finos/perspective-webpack-plugin --save-dev
```

23.2.2 Python Installation

```
# Install Python package
pip install perspective-python

# For JupyterLab integration
pip install perspective-python[jupyter]
jupyter labextension install @finos/perspective-
   jupyterlab
```

23.2.3 Webpack Configuration

```
// webpack.config.js
const PerspectivePlugin = require("@finos/perspective-
   webpack-plugin");

module.exports = {
  entry: "./src/index.js",
  plugins: [
    new PerspectivePlugin()
  ],
  module: {
    rules: [
      {
        test: /\.css$/,
        use: ["style-loader", "css-loader"]
      }
    ]
  }
};
```

23.3 Working Example: Real-Time Trading Dashboard

This example creates a financial trading dashboard with real-time price updates, order book visualization, and portfolio analytics.

23.3.1 Project Structure

```
trading-dashboard/
|-- src/
|   |-- index.js
|   |-- components/
|   |   |-- PriceGrid.js
|   |   |-- OrderBook.js
|   |   |-- PortfolioChart.js
|   |-- services/
|   |   |-- MarketDataStream.js
|   |-- schemas/
|       |-- trade.js
|-- public/
|   |-- index.html
|-- webpack.config.js
|-- package.json
```

23.3.2 HTML Setup

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,
     initial-scale=1">
  <title>Trading Dashboard - FINOS Perspective</title>
  <style>
    * { box-sizing: border-box; margin: 0; padding: 0; }
    body {
      font-family: -apple-system, BlinkMacSystemFont,
         sans-serif;
      background: #1a1a2e;
      color: #eee;
    }
    .dashboard {
      display: grid;
      grid-template-columns: 1fr 1fr;
      grid-template-rows: 1fr 1fr;
      gap: 8px;
      padding: 8px;
      height: 100vh;
```

```
    }
    .panel {
      background: #16213e;
      border-radius: 4px;
      overflow: hidden;
      position: relative;
    }
    .panel h3 {
      padding: 8px 12px;
      background: #0f3460;
      font-size: 12px;
      text-transform: uppercase;
      letter-spacing: 1px;
    }
    perspective-viewer {
      --plugin--background-color: transparent;
      height: calc(100% - 32px);
    }
  </style>
</head>
<body>
  <div class="dashboard">
    <div class="panel">
      <h3>Live Prices</h3>
      <perspective-viewer id="prices"></perspective-
          viewer>
    </div>
    <div class="panel">
      <h3>Order Book</h3>
      <perspective-viewer id="orderbook"></perspective-
          viewer>
    </div>
    <div class="panel" style="grid-column: span 2;">
      <h3>Trade History</h3>
      <perspective-viewer id="trades"></perspective-
          viewer>
    </div>
  </div>
  <script src="bundle.js"></script>
</body>
</html>
```

23.3.3 Main Application

```
// src/index.js
import perspective from "@finos/perspective";
import "@finos/perspective-viewer";
import "@finos/perspective-viewer-datagrid";
import "@finos/perspective-viewer-d3fc";

// Import styles
import "@finos/perspective-viewer/dist/css/pro.css";

// Define schemas
const PRICE_SCHEMA = {
  symbol: "string",
  bid: "float",
  ask: "float",
  last: "float",
  change: "float",
  changePercent: "float",
  volume: "integer",
  timestamp: "datetime"
};

const ORDER_SCHEMA = {
  symbol: "string",
  side: "string",
  price: "float",
  quantity: "integer",
  total: "float",
  timestamp: "datetime"
};

const TRADE_SCHEMA = {
  tradeId: "string",
  symbol: "string",
  side: "string",
  price: "float",
  quantity: "integer",
  value: "float",
  timestamp: "datetime",
  exchange: "string"
};
```

```
class TradingDashboard {
  constructor() {
    this.tables = {};
    this.viewers = {};
    this.worker = perspective.worker();
    this.symbols = ['AAPL', 'GOOGL', 'MSFT', 'AMZN', '
       META', 'TSLA', 'NVDA', 'JPM'];
  }

  async init() {
    // Create tables
    this.tables.prices = await this.worker.table(
       PRICE_SCHEMA, { index: "symbol" });
    this.tables.orderbook = await this.worker.table(
       ORDER_SCHEMA);
    this.tables.trades = await this.worker.table(
       TRADE_SCHEMA, { limit: 10000 });

    // Initialize viewers
    await this.initPriceViewer();
    await this.initOrderBookViewer();
    await this.initTradeViewer();

    // Start data streams
    this.startPriceStream();
    this.startOrderStream();
    this.startTradeStream();
  }

  async initPriceViewer() {
    const viewer = document.getElementById("prices");
    this.viewers.prices = viewer;

    await viewer.load(this.tables.prices);

    await viewer.restore({
      plugin: "Datagrid",
      columns: ["symbol", "last", "change", "
         changePercent", "bid", "ask", "volume"],
      sort: [["volume", "desc"]],
      plugin_config: {
        columns: {
          change: {
```

```
            color_mode: "bar",
            pos_color: "#4caf50",
            neg_color: "#f44336"
          },
          changePercent: {
            color_mode: "foreground",
            pos_color: "#4caf50",
            neg_color: "#f44336"
          }
        }
      },
      theme: "Pro Dark"
    });
  }

  async initOrderBookViewer() {
    const viewer = document.getElementById("orderbook");
    this.viewers.orderbook = viewer;

    await viewer.load(this.tables.orderbook);

    await viewer.restore({
      plugin: "Y Bar",
      group_by: ["price"],
      split_by: ["side"],
      columns: ["quantity"],
      aggregates: { quantity: "sum" },
      filter: [["symbol", "==", "AAPL"]],
      sort: [["price", "desc"]],
      theme: "Pro Dark"
    });

    // Enable cross-filtering
    viewer.addEventListener("perspective-click", (event)
        => {
      const { config, row } = event.detail;
      if (row && row.symbol) {
        this.filterBySymbol(row.symbol);
      }
    });
  }

  async initTradeViewer() {
```

```
    const viewer = document.getElementById("trades");
    this.viewers.trades = viewer;

    await viewer.load(this.tables.trades);

    await viewer.restore({
      plugin: "Datagrid",
      columns: ["timestamp", "tradeId", "symbol", "side"
          , "price", "quantity", "value", "exchange"],
      sort: [["timestamp", "desc"]],
      plugin_config: {
        columns: {
          side: {
            color_mode: "foreground",
            pos_color: "#4caf50",
            neg_color: "#f44336"
          },
          value: {
            number_color_mode: "bar"
          }
        }
      },
      theme: "Pro Dark"
    });
  }

  filterBySymbol(symbol) {
    this.viewers.orderbook.restore({
      filter: [["symbol", "==", symbol]]
    });
  }

  startPriceStream() {
    // Simulate real-time price updates
    const baseprices = new Map(
      this.symbols.map(s => [s, 100 + Math.random() *
          400])
    );

    setInterval(() => {
      const updates = this.symbols.map(symbol => {
        const base = baseprices.get(symbol);
        const change = (Math.random() - 0.5) * 2;
```

```
        const newPrice = base + change;
        baseprices.set(symbol, newPrice);

        const spread = newPrice * 0.001;
        return {
          symbol,
          bid: newPrice - spread,
          ask: newPrice + spread,
          last: newPrice,
          change: change,
          changePercent: (change / base) * 100,
          volume: Math.floor(Math.random() * 1000000),
          timestamp: new Date()
        };
      });

      this.tables.prices.update(updates);
    }, 100);  // 10 updates per second
  }

  startOrderStream() {
    setInterval(() => {
      const orders = [];
      const symbol = this.symbols[Math.floor(Math.random
         () * this.symbols.length)];
      const midPrice = 150 + Math.random() * 50;

      // Generate order book levels
      for (let i = 0; i < 10; i++) {
        // Bid side
        orders.push({
          symbol,
          side: "BID",
          price: Math.round((midPrice - i * 0.1) * 100)
             / 100,
          quantity: Math.floor(Math.random() * 1000) +
             100,
          total: 0,
          timestamp: new Date()
        });

        // Ask side
        orders.push({
```

```
          symbol,
          side: "ASK",
          price: Math.round((midPrice + i * 0.1) * 100)
             / 100,
          quantity: Math.floor(Math.random() * 1000) +
             100,
          total: 0,
          timestamp: new Date()
        });
      }

      // Calculate totals
      orders.forEach(o => { o.total = o.price * o.
         quantity; });

      this.tables.orderbook.replace(orders);
    }, 500);
  }

  startTradeStream() {
    const exchanges = ['NYSE', 'NASDAQ', 'BATS', 'IEX'];
    let tradeCounter = 0;

    setInterval(() => {
      const symbol = this.symbols[Math.floor(Math.random
         () * this.symbols.length)];
      const price = 100 + Math.random() * 400;
      const quantity = Math.floor(Math.random() * 500) +
          1;

      const trade = {
        tradeId: `TRD${String(++tradeCounter).padStart
           (8, '0')}`,
        symbol,
        side: Math.random() > 0.5 ? "BUY" : "SELL",
        price: Math.round(price * 100) / 100,
        quantity,
        value: Math.round(price * quantity * 100) / 100,
        timestamp: new Date(),
        exchange: exchanges[Math.floor(Math.random() *
           exchanges.length)]
      };
```

```
      this.tables.trades.update([trade]);
    }, 50);  // 20 trades per second
  }
}

// Initialize dashboard
document.addEventListener("DOMContentLoaded", async ()
   => {
  const dashboard = new TradingDashboard();
  await dashboard.init();
});
```

23.3.4 Expression Columns

```
// Add computed columns using Perspective expressions
await viewer.restore({
  columns: ["symbol", "last", "vwap", "spread", "
     spreadBps"],
  expressions: {
    // Volume-Weighted Average Price
    'vwap': '"value" / "quantity"',

    // Bid-Ask Spread
    'spread': '"ask" - "bid"',

    // Spread in basis points
    'spreadBps': '(("ask" - "bid") / "last") * 10000',

    // Price change indicator
    'trend': 'if("change" > 0, "UP", if("change" < 0, "
       DOWN", "FLAT"))',

    // Time bucket for aggregation
    'minute': 'bucket("timestamp", "m")'
  }
});
```

23.4 Python Server Mode

For large datasets that exceed browser memory, use Perspective's server mode.

23.4.1 Python Server

```
# server.py
import asyncio
import perspective
from perspective import Table, PerspectiveManager,
    PerspectiveTornadoHandler
import tornado.web
import tornado.ioloop
import pandas as pd
from datetime import datetime
import random

class TradingDataServer:
    def __init__(self):
        self.manager = PerspectiveManager()
        self.tables = {}

    def create_tables(self):
        # Create schema
        schema = {
            "trade_id": str,
            "symbol": str,
            "side": str,
            "price": float,
            "quantity": int,
            "value": float,
            "timestamp": datetime,
            "exchange": str
        }

        # Create table with 1M row limit
        self.tables["trades"] = Table(schema, limit=1
            _000_000)
        self.manager.host_table("trades", self.tables["
            trades"])
```

```
    async def generate_data(self):
        symbols = ["AAPL", "GOOGL", "MSFT", "AMZN", "
            META", "TSLA"]
        exchanges = ["NYSE", "NASDAQ", "BATS"]
        trade_id = 0

        while True:
            trades = []
            for _ in range(100):  # Batch of 100 trades
                trade_id += 1
                symbol = random.choice(symbols)
                price = 100 + random.random() * 400
                quantity = random.randint(1, 1000)

                trades.append({
                    "trade_id": f"TRD{trade_id:010d}",
                    "symbol": symbol,
                    "side": random.choice(["BUY", "SELL"
                        ]),
                    "price": round(price, 2),
                    "quantity": quantity,
                    "value": round(price * quantity, 2),
                    "timestamp": datetime.now(),
                    "exchange": random.choice(exchanges)
                })

            # Update table
            self.tables["trades"].update(trades)
            await asyncio.sleep(0.1)  # 1000 trades/
                second

def make_app(manager):
    return tornado.web.Application([
        (
            r"/websocket",
            PerspectiveTornadoHandler,
            {"manager": manager, "check_origin": True}
        ),
        (
            r"/(.*)",
            tornado.web.StaticFileHandler,
            {"path": "./public", "default_filename": "
                index.html"}
```

```
        )
    ])

async def main():
    server = TradingDataServer()
    server.create_tables()

    app = make_app(server.manager)
    app.listen(8080)
    print("Server running on http://localhost:8080")

    # Start data generation
    await server.generate_data()

if __name__ == "__main__":
    asyncio.run(main())
```

23.4.2 Client Connection to Server

```
// Connect to Python server
import perspective from "@finos/perspective";

async function connectToServer() {
  // Create WebSocket connection
  const websocket = perspective.websocket("ws://
     localhost:8080/websocket");

  // Open remote table
  const table = await websocket.open_table("trades");

  // Load into viewer
  const viewer = document.getElementById("trades-viewer"
     );
  await viewer.load(table);

  await viewer.restore({
    plugin: "Datagrid",
    group_by: ["symbol"],
    columns: ["trade_id", "price", "quantity", "value"],
    aggregates: {
      trade_id: "count",
      price: "avg",
```

```
      quantity: "sum",
      value: "sum"
    }
  });
}

connectToServer();
```

23.5 Integration with Apache Arrow

23.5.1 Loading Arrow Data

```
import perspective from "@finos/perspective";
import { tableFromIPC } from "apache-arrow";

async function loadArrowData(url) {
  // Fetch Arrow IPC file
  const response = await fetch(url);
  const buffer = await response.arrayBuffer();

  // Create Perspective table from Arrow
  const worker = perspective.worker();
  const table = await worker.table(buffer);

  return table;
}

// Load from Parquet via Arrow
async function loadParquetData(url) {
  // Requires server-side conversion
  const response = await fetch(`/api/parquet-to-arrow?
     file=${url}`);
  const buffer = await response.arrayBuffer();

  const worker = perspective.worker();
  return await worker.table(buffer);
}
```

23.5.2 Streaming Arrow Updates

```
// Efficient streaming with Arrow
class ArrowDataStream {
  constructor(table) {
    this.table = table;
    this.buffer = [];
    this.batchSize = 1000;
    this.flushInterval = 100;
  }

  start() {
    // Batch updates for efficiency
    setInterval(() => {
      if (this.buffer.length > 0) {
        this.table.update(this.buffer);
        this.buffer = [];
      }
    }, this.flushInterval);
  }

  push(record) {
    this.buffer.push(record);

    // Flush if batch size reached
    if (this.buffer.length >= this.batchSize) {
      this.table.update(this.buffer);
      this.buffer = [];
    }
  }
}
```

23.6 JupyterLab Integration

```
# In Jupyter notebook
import perspective
import pandas as pd
import numpy as np

# Create sample data
df = pd.DataFrame({
```

```
    "symbol": np.random.choice(["AAPL", "GOOGL", "MSFT"
        ], 10000),
    "price": np.random.uniform(100, 500, 10000),
    "volume": np.random.randint(100, 10000, 10000),
    "timestamp": pd.date_range("2024-01-01", periods
        =10000, freq="s")
})

# Create Perspective widget
widget = perspective.PerspectiveWidget(
    df,
    plugin="Y Line",
    group_by=["timestamp"],
    split_by=["symbol"],
    columns=["price"],
    aggregates={"price": "avg"}
)

# Display interactive widget
widget
```

23.7 URL

https://perspective.finos.org

23.8 Books

- **FINOS Perspective Documentation** – Official comprehensive guide
- **High-Performance JavaScript** (O'Reilly) – WebAssembly optimization techniques
- **Apache Arrow in Action** – Arrow format and cross-language data sharing

23.9 Hardening

23.9.1 Data Validation

```
// Validate data before loading
function validateTradeData(data) {
  const requiredFields = ['symbol', 'price', 'quantity',
      'timestamp'];
  const numericFields = ['price', 'quantity', 'value'];

  return data.filter(record => {
    // Check required fields
    for (const field of requiredFields) {
      if (record[field] === undefined || record[field]
          === null) {
        console.warn(`Missing field: ${field}`);
        return false;
      }
    }

    // Validate numeric fields
    for (const field of numericFields) {
      if (record[field] !== undefined && !isFinite(
          record[field])) {
        console.warn(`Invalid numeric value: ${field}`);
        return false;
      }
    }

    // Validate timestamp
    if (!(record.timestamp instanceof Date) &&
        isNaN(Date.parse(record.timestamp))) {
      console.warn('Invalid timestamp');
      return false;
    }

    return true;
  });
}
```

23.9.2 Resource Limits

```
// Configure table limits to prevent memory issues
const table = await worker.table(schema, {
  limit: 100000,  // Max 100k rows in browser
  index: "id"     // Enable efficient updates
});

// Monitor memory usage
setInterval(async () => {
  const size = await table.size();
  const view = await table.view();
  const numColumns = await view.num_columns();

  console.log(`Table size: ${size} rows, ${numColumns}
     columns`);

  // Warn if approaching limit
  if (size > 90000) {
    console.warn('Approaching row limit, consider
       archiving old data');
  }
}, 60000);
```

23.9.3 WebSocket Security

```
# server.py - Secure WebSocket handler
from perspective import PerspectiveTornadoHandler
import tornado.web

class SecurePerspectiveHandler(PerspectiveTornadoHandler
   ):
    def check_origin(self, origin):
        # Only allow specific origins
        allowed_origins = [
            "https://dashboard.example.com",
            "http://localhost:3000"
        ]
        return origin in allowed_origins

    async def open(self):
        # Authenticate connection
```

```
        token = self.get_argument("token", None)
        if not self.validate_token(token):
            self.close(code=4001, reason="Unauthorized")
            return
        await super().open()

    def validate_token(self, token):
        # Implement token validation
        return token is not None and len(token) > 0
```

Chapter 24

Cube.js

24.1 What Cube.js Is

Cube.js (now known simply as Cube) is a headless business intelligence platform that provides a semantic layer between your data sources and analytics applications. Rather than building visualizations directly, Cube acts as an API-first analytics engine that handles query optimization, caching, access control, and provides a unified interface across multiple data sources.

The semantic layer approach separates data modeling from visualization, allowing analytics teams to define business metrics once and expose them consistently across dashboards, embedded analytics, and custom applications. This architecture is particularly valuable for organizations building data products or embedding analytics into their applications.

24.1.1 Key Features

- **Semantic Layer**: Define metrics, dimensions, and joins in a data model
- **Query Orchestration**: Automatic query optimization and federation
- **Pre-aggregations**: Materialized rollups for sub-second query performance
- **Caching**: Multi-tier caching with automatic invalidation
- **Access Control**: Row-level and dimension-level security
- **API-First**: REST, GraphQL, and SQL APIs for flexibility
- **Multi-Tenancy**: Built-in support for SaaS analytics

- **Database Agnostic**: Supports 20+ data sources

24.1.2 Architecture

Component	Purpose
Cube Store	Distributed cache and pre-aggregation storage
API Gateway	REST/GraphQL/SQL endpoint
Query Orchestrator	Query planning and execution
Schema Compiler	Data model compilation
Refresh Worker	Background pre-aggregation refresh

Table 24.1: Cube.js architecture components

24.2 Install Instructions

24.2.1 Quick Start with Docker

```
# Create project directory
mkdir cube-analytics && cd cube-analytics

# Create docker-compose.yml
cat > docker-compose.yml << 'EOF'
version: '3.8'
services:
  cube:
    image: cubejs/cube:latest
    ports:
      - 4000:4000
      - 15432:15432  # SQL API
    environment:
      CUBEJS_DEV_MODE: "true"
      CUBEJS_DB_TYPE: postgres
      CUBEJS_DB_HOST: postgres
      CUBEJS_DB_NAME: analytics
      CUBEJS_DB_USER: cube
      CUBEJS_DB_PASS: cube_password
      CUBEJS_API_SECRET: your-secret-key-here
    volumes:
      - ./schema:/cube/conf/schema
```

```
    depends_on:
      - postgres

  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: analytics
      POSTGRES_USER: cube
      POSTGRES_PASSWORD: cube_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql

volumes:
  postgres_data:
EOF

# Start services
docker-compose up -d
```

24.2.2 NPM Installation

```
# Create new Cube project
npx cubejs-cli create analytics-api -d postgres

# Navigate to project
cd analytics-api

# Configure environment
cat > .env << 'EOF'
CUBEJS_DEV_MODE=true
CUBEJS_DB_TYPE=postgres
CUBEJS_DB_HOST=localhost
CUBEJS_DB_PORT=5432
CUBEJS_DB_NAME=analytics
CUBEJS_DB_USER=cube
CUBEJS_DB_PASS=cube_password
CUBEJS_API_SECRET=your-256-bit-secret
CUBEJS_EXTERNAL_DEFAULT=true
CUBEJS_SCHEDULED_REFRESH_DEFAULT=true
EOF
```

```
# Install dependencies
npm install

# Start development server
npm run dev
```

24.3 Working Example: E-Commerce Analytics API

This example creates a complete analytics backend for an e-commerce platform with orders, products, and customer data.

24.3.1 Sample Data Schema

```
-- init.sql
CREATE TABLE customers (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) NOT NULL,
  name VARCHAR(255),
  segment VARCHAR(50),
  country VARCHAR(100),
  created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  category VARCHAR(100),
  subcategory VARCHAR(100),
  price DECIMAL(10,2),
  cost DECIMAL(10,2),
  created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE orders (
  id SERIAL PRIMARY KEY,
  customer_id INTEGER REFERENCES customers(id),
  status VARCHAR(50),
  created_at TIMESTAMP DEFAULT NOW(),
  completed_at TIMESTAMP
```

```
);

CREATE TABLE order_items (
  id SERIAL PRIMARY KEY,
  order_id INTEGER REFERENCES orders(id),
  product_id INTEGER REFERENCES products(id),
  quantity INTEGER NOT NULL,
  price DECIMAL(10,2) NOT NULL,
  discount DECIMAL(10,2) DEFAULT 0
);

-- Create indexes for performance
CREATE INDEX idx_orders_created_at ON orders(created_at)
   ;
CREATE INDEX idx_orders_customer_id ON orders(
   customer_id);
CREATE INDEX idx_order_items_order_id ON order_items(
   order_id);
CREATE INDEX idx_order_items_product_id ON order_items(
   product_id);
```

24.3.2 Cube Data Model - Orders

```
// schema/Orders.js
cube(`Orders`, {
  sql: `SELECT * FROM orders`,

  // Define joins to related cubes
  joins: {
    Customers: {
      relationship: `belongsTo`,
      sql: `${CUBE}.customer_id = ${Customers}.id`
    },
    OrderItems: {
      relationship: `hasMany`,
      sql: `${CUBE}.id = ${OrderItems}.order_id`
    }
  },

  // Measures (aggregations)
  measures: {
    count: {
```

```
    type: `count`,
    drillMembers: [id, createdAt, status]
  },

  totalRevenue: {
    sql: `${OrderItems.lineTotal}`,
    type: `sum`,
    format: `currency`
  },

  averageOrderValue: {
    sql: `${totalRevenue} / NULLIF(${count}, 0)`,
    type: `number`,
    format: `currency`
  },

  completedCount: {
    type: `count`,
    filters: [{ sql: `${CUBE}.status = 'completed'` }]
  },

  completionRate: {
    sql: `${completedCount}::float / NULLIF(${count},
       0) * 100`,
    type: `number`,
    format: `percent`
  },

  averageFulfillmentTime: {
    sql: `AVG(EXTRACT(EPOCH FROM (${CUBE}.completed_at
        - ${CUBE}.created_at)) / 3600)`,
    type: `number`,
    title: `Avg Fulfillment Time (hours)`
  }
},

// Dimensions (groupings)
dimensions: {
  id: {
    sql: `${CUBE}.id`,
    type: `number`,
    primaryKey: true
  },
```

```
    status: {
      sql: `${CUBE}.status`,
      type: `string`
    },

    createdAt: {
      sql: `${CUBE}.created_at`,
      type: `time`
    },

    completedAt: {
      sql: `${CUBE}.completed_at`,
      type: `time`
    },

    // Time-based dimensions
    createdAtMonth: {
      sql: `DATE_TRUNC('month', ${CUBE}.created_at)`,
      type: `time`
    },

    createdAtWeek: {
      sql: `DATE_TRUNC('week', ${CUBE}.created_at)`,
      type: `time`
    },

    dayOfWeek: {
      sql: `EXTRACT(DOW FROM ${CUBE}.created_at)`,
      type: `number`
    },

    hourOfDay: {
      sql: `EXTRACT(HOUR FROM ${CUBE}.created_at)`,
      type: `number`
    }
  },

  // Segments (pre-defined filters)
  segments: {
    completed: {
      sql: `${CUBE}.status = 'completed'`
    },
```

```
    thisMonth: {
      sql: `${CUBE}.created_at >= DATE_TRUNC('month',
         NOW())`
    },
    highValue: {
      sql: `${CUBE}.id IN (
        SELECT order_id FROM order_items
        GROUP BY order_id
        HAVING SUM(price * quantity) > 500
      )`
    }
  },

  // Pre-aggregations for performance
  preAggregations: {
    // Daily rollup
    dailyRevenue: {
      measures: [totalRevenue, count, averageOrderValue
         ],
      dimensions: [status],
      timeDimension: createdAt,
      granularity: `day`,
      partitionGranularity: `month`,
      refreshKey: {
        every: `1 hour`
      }
    },

    // Monthly rollup by customer segment
    monthlyBySegment: {
      measures: [totalRevenue, count],
      dimensions: [Customers.segment, Customers.country
         ],
      timeDimension: createdAt,
      granularity: `month`,
      refreshKey: {
        every: `6 hours`
      }
    }
  }
});
```

24.3.3 Cube Data Model - Customers

```
// schema/Customers.js
cube(`Customers`, {
  sql: `SELECT * FROM customers`,

  joins: {
    Orders: {
      relationship: `hasMany`,
      sql: `${CUBE}.id = ${Orders}.customer_id`
    }
  },

  measures: {
    count: {
      type: `count`,
      drillMembers: [id, email, name]
    },

    totalCustomers: {
      type: `countDistinct`,
      sql: `${CUBE}.id`
    },

    newCustomers: {
      type: `count`,
      filters: [
        {
          sql: `${CUBE}.created_at >= DATE_TRUNC('month
            ', NOW())`
        }
      ]
    },

    lifetimeValue: {
      sql: `${Orders.totalRevenue}`,
      type: `sum`,
      format: `currency`
    },

    averageLifetimeValue: {
      sql: `${lifetimeValue} / NULLIF(${count}, 0)`,
      type: `number`,
```

```
    format: `currency`
  },

  repeatCustomerRate: {
    sql: `
      COUNT(DISTINCT CASE WHEN customer_order_count >
         1 THEN customer_id END)::float /
      NULLIF(COUNT(DISTINCT customer_id), 0) * 100
    `,
    type: `number`,
    format: `percent`
  }
},

dimensions: {
  id: {
    sql: `${CUBE}.id`,
    type: `number`,
    primaryKey: true
  },

  email: {
    sql: `${CUBE}.email`,
    type: `string`
  },

  name: {
    sql: `${CUBE}.name`,
    type: `string`
  },

  segment: {
    sql: `${CUBE}.segment`,
    type: `string`
  },

  country: {
    sql: `${CUBE}.country`,
    type: `string`
  },

  createdAt: {
    sql: `${CUBE}.created_at`,
```

```
      type: `time`
    },

    // Cohort dimension
    signupCohort: {
      sql: `DATE_TRUNC('month', ${CUBE}.created_at)`,
      type: `time`
    },

    // Customer tier based on spending
    tier: {
      type: `string`,
      case: {
        when: [
          {
            sql: `${lifetimeValue} >= 10000`,
            label: `Platinum`
          },
          {
            sql: `${lifetimeValue} >= 5000`,
            label: `Gold`
          },
          {
            sql: `${lifetimeValue} >= 1000`,
            label: `Silver`
          }
        ],
        else: { label: `Bronze` }
      }
    }
  },

  preAggregations: {
    customerMetrics: {
      measures: [count, lifetimeValue,
         averageLifetimeValue],
      dimensions: [segment, country, tier],
      refreshKey: {
        every: `1 day`
      }
    }
  }
});
```

24.3.4 Cube Data Model - Products

```
// schema/Products.js
cube(`Products`, {
  sql: `SELECT * FROM products`,

  joins: {
    OrderItems: {
      relationship: `hasMany`,
      sql: `${CUBE}.id = ${OrderItems}.product_id`
    }
  },

  measures: {
    count: {
      type: `count`
    },

    totalSold: {
      sql: `${OrderItems.quantity}`,
      type: `sum`
    },

    totalRevenue: {
      sql: `${OrderItems.lineTotal}`,
      type: `sum`,
      format: `currency`
    },

    averagePrice: {
      sql: `${CUBE}.price`,
      type: `avg`,
      format: `currency`
    },

    profitMargin: {
      sql: `(${CUBE}.price - ${CUBE}.cost) / NULLIF(${
         CUBE}.price, 0) * 100`,
      type: `avg`,
      format: `percent`
    }
  },
```

```
dimensions: {
  id: {
    sql: `${CUBE}.id`,
    type: `number`,
    primaryKey: true
  },

  name: {
    sql: `${CUBE}.name`,
    type: `string`
  },

  category: {
    sql: `${CUBE}.category`,
    type: `string`
  },

  subcategory: {
    sql: `${CUBE}.subcategory`,
    type: `string`
  },

  price: {
    sql: `${CUBE}.price`,
    type: `number`,
    format: `currency`
  },

  priceRange: {
    type: `string`,
    case: {
      when: [
        { sql: `${CUBE}.price < 25`, label: `Budget`
           },
        { sql: `${CUBE}.price < 100`, label: `Mid-
           Range` },
        { sql: `${CUBE}.price < 500`, label: `Premium`
            }
      ],
      else: { label: `Luxury` }
    }
  }
}
```

```
});

// Order Items cube
cube(`OrderItems`, {
  sql: `SELECT * FROM order_items`,

  joins: {
    Orders: {
      relationship: `belongsTo`,
      sql: `${CUBE}.order_id = ${Orders}.id`
    },
    Products: {
      relationship: `belongsTo`,
      sql: `${CUBE}.product_id = ${Products}.id`
    }
  },

  measures: {
    count: {
      type: `count`
    },

    quantity: {
      sql: `${CUBE}.quantity`,
      type: `sum`
    },

    lineTotal: {
      sql: `${CUBE}.price * ${CUBE}.quantity - ${CUBE}.
         discount`,
      type: `sum`,
      format: `currency`
    },

    averageDiscount: {
      sql: `${CUBE}.discount`,
      type: `avg`,
      format: `currency`
    }
  },

  dimensions: {
    id: {
```

```
      sql: `${CUBE}.id`,
      type: `number`,
      primaryKey: true
    },

    price: {
      sql: `${CUBE}.price`,
      type: `number`
    }
  }
});
```

24.3.5 Multi-Tenancy Configuration

```
// cube.js
module.exports = {
  // Security context from JWT
  contextToAppId: ({ securityContext }) => {
    return `CUBE_APP_${securityContext.tenantId}`;
  },

  // Tenant-specific database
  driverFactory: ({ securityContext }) => {
    return {
      type: 'postgres',
      host: process.env.CUBEJS_DB_HOST,
      database: `tenant_${securityContext.tenantId}`,
      user: process.env.CUBEJS_DB_USER,
      password: process.env.CUBEJS_DB_PASS
    };
  },

  // Row-level security
  queryRewrite: (query, { securityContext }) => {
    if (!securityContext.isAdmin) {
      // Add tenant filter to all queries
      query.filters.push({
        member: 'Orders.tenantId',
        operator: 'equals',
        values: [securityContext.tenantId]
      });
    }
```

```
    return query;
  },

  // Schedule pre-aggregation refresh per tenant
  scheduledRefreshContexts: async () => {
    const tenants = await getTenantList();
    return tenants.map(tenant => ({
      securityContext: { tenantId: tenant.id }
    }));
  }
};
```

24.3.6 Frontend Integration (React)

```
// App.js
import React from 'react';
import cubejs from '@cubejs-client/core';
import { CubeProvider, QueryRenderer } from '@cubejs-
   client/react';
import { Line, Bar, Pie } from 'react-chartjs-2';

// Initialize Cube API client
const cubeApi = cubejs(process.env.
   REACT_APP_CUBEJS_TOKEN, {
  apiUrl: process.env.REACT_APP_CUBEJS_API_URL
});

// Revenue over time chart
function RevenueChart() {
  return (
    <QueryRenderer
      cubeApi={cubeApi}
      query={{
        measures: ['Orders.totalRevenue', 'Orders.count'
           ],
        timeDimensions: [{
          dimension: 'Orders.createdAt',
          granularity: 'month',
          dateRange: 'last 12 months'
        }]
      }}
      render={({ resultSet, isLoading, error }) => {
```

```
        if (isLoading) return <div>Loading...</div>;
        if (error) return <div>Error: {error.message}</
           div>;

        const data = {
          labels: resultSet.chartPivot().map(row => row.
             x),
          datasets: [{
            label: 'Revenue',
            data: resultSet.chartPivot().map(row => row[
               'Orders.totalRevenue']),
            borderColor: '#4CAF50',
            fill: false
          }]
        };

        return <Line data={data} />;
      }}
    />
  );
}

// Customer segmentation chart
function SegmentChart() {
  return (
    <QueryRenderer
      cubeApi={cubeApi}
      query={{
        measures: ['Customers.count', 'Customers.
           lifetimeValue'],
        dimensions: ['Customers.segment']
      }}
      render={({ resultSet }) => {
        if (!resultSet) return null;

        const data = {
          labels: resultSet.tablePivot().map(row => row[
             'Customers.segment']),
          datasets: [{
            data: resultSet.tablePivot().map(row => row[
               'Customers.lifetimeValue']),
            backgroundColor: ['#FF6384', '#36A2EB', '#
               FFCE56', '#4BC0C0']
```

```
          }]
        };

        return <Pie data={data} />;
      }}
    />
  );
}

function App() {
  return (
    <CubeProvider cubeApi={cubeApi}>
      <div className="dashboard">
        <h1>E-Commerce Analytics</h1>
        <div className="chart-grid">
          <RevenueChart />
          <SegmentChart />
        </div>
      </div>
    </CubeProvider>
  );
}

export default App;
```

24.3.7 SQL API Usage

```
-- Connect via psql to Cube SQL API (port 15432)
-- psql -h localhost -p 15432 -U cube

-- Query using familiar SQL syntax
SELECT
  DATE_TRUNC('month', "Orders.createdAt") AS month,
  "Customers.segment" AS segment,
  SUM("Orders.totalRevenue") AS revenue,
  COUNT("Orders.count") AS orders
FROM Orders
CROSS JOIN Customers
WHERE "Orders.createdAt" >= '2024-01-01'
GROUP BY 1, 2
ORDER BY 1, 2;
```

```
-- Use pre-aggregations automatically
SELECT
  "Products.category",
  SUM("OrderItems.lineTotal") AS total_sales,
  SUM("OrderItems.quantity") AS units_sold
FROM OrderItems
CROSS JOIN Products
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10;
```

24.4 Integration with Big Data Stack

24.4.1 Apache Druid as Data Source

```
// .env
CUBEJS_DB_TYPE=druid
CUBEJS_DB_URL=http://druid-broker:8082/druid/v2/sql

// schema/Events.js
cube(`Events`, {
  sql: `SELECT * FROM events`,

  measures: {
    count: {
      type: `count`
    },
    uniqueUsers: {
      sql: `user_id`,
      type: `countDistinctApprox`  // Uses Druid's HLL
    }
  },

  dimensions: {
    eventType: {
      sql: `event_type`,
      type: `string`
    },
    timestamp: {
      sql: `__time`,
      type: `time`
```

```
    }
  }
});
```

24.5 URL

https://cube.dev

24.6 Books

- **Cube.js Documentation** – Official comprehensive guide
- **The Semantic Layer** by Cube Team – Best practices for data modeling
- **Building Analytics APIs** – Practical guide to headless BI

24.7 Hardening

24.7.1 API Security

```
// cube.js - JWT Authentication
const jwt = require('jsonwebtoken');

module.exports = {
  checkAuth: (req, auth) => {
    // Verify JWT token
    const token = auth && auth.split(' ')[1];

    if (!token) {
      throw new Error('No token provided');
    }

    try {
      const decoded = jwt.verify(token, process.env.
         CUBEJS_API_SECRET);
      req.securityContext = decoded;
    } catch (err) {
      throw new Error('Invalid token');
```

```
    }
  },

  // Query complexity limits
  queryRewrite: (query, { securityContext }) => {
    // Limit time range
    const maxDays = securityContext.isAdmin ? 365 : 90;

    if (query.timeDimensions) {
      query.timeDimensions = query.timeDimensions.map(td
          => ({
        ...td,
        dateRange: limitDateRange(td.dateRange, maxDays)
      }));
    }

    // Limit result size
    if (!query.limit || query.limit > 10000) {
      query.limit = 10000;
    }

    return query;
  }
};
```

24.7.2 Pre-aggregation Security

```
// Secure pre-aggregation configuration
preAggregations: {
  secureRollup: {
    measures: [totalRevenue, count],
    dimensions: [status],
    timeDimension: createdAt,
    granularity: `day`,

    // Refresh only during off-peak hours
    refreshKey: {
      every: `1 day`,
      timezone: `UTC`,
      sql: `SELECT MAX(updated_at) FROM orders`
    },
```

```
    // Store in separate schema
    external: true,
    externalRefresh: true,

    // Index for faster queries
    indexes: {
      categoryIndex: {
        columns: [status]
      }
    }
  }
}
```

Chapter 25

Plotly.js

25.1 What Plotly.js Is

Plotly.js is a high-level, declarative charting library that combines the power of D3.js with WebGL for hardware-accelerated rendering. Originally developed by Plotly Inc., it provides a comprehensive set of chart types with built-in interactivity including zooming, panning, hover tooltips, and click events.

Unlike lower-level libraries like D3.js, Plotly uses a declarative approach where you specify what you want rather than how to draw it. This makes it accessible to data scientists and analysts while still offering extensive customization for developers. The library is the foundation for Plotly's Python, R, and Julia visualization libraries.

25.1.1 Key Features

- **40+ Chart Types**: Scientific, statistical, financial, geographic, and 3D charts
- **WebGL Rendering**: Hardware-accelerated for millions of points
- **Declarative API**: JSON-based trace and layout specification
- **Interactive by Default**: Zoom, pan, hover, and selection built-in
- **Export Options**: PNG, SVG, PDF, and WebP export
- **Responsive**: Automatic resize and mobile touch support
- **Animations**: Smooth transitions between states

25.2 Install Instructions

25.2.1 JavaScript/NPM

```
# Full bundle (all chart types)
npm install plotly.js-dist

# Minified bundle
npm install plotly.js-dist-min

# Basic bundle (common charts only - smaller)
npm install plotly.js-basic-dist-min
```

25.2.2 Python (Plotly Express)

```
pip install plotly pandas

# For Jupyter notebook support
pip install jupyter-dash
```

25.3 Working Example: Analytics Dashboard

```
import Plotly from 'plotly.js-dist-min';

// Time series with range slider
const timeSeriesData = [{
  x: timestamps,
  y: values,
  type: 'scatter',
  mode: 'lines',
  name: 'Revenue',
  line: { color: '#4CAF50', width: 2 }
}];

const timeSeriesLayout = {
  title: 'Revenue Over Time',
  xaxis: {
    rangeslider: { visible: true },
    type: 'date'
```

```
  },
  yaxis: { title: 'Revenue ($)' },
  template: 'plotly_dark'
};

Plotly.newPlot('timeseries', timeSeriesData,
   timeSeriesLayout);

// 3D scatter plot with WebGL
const scatter3dData = [{
  x: xData,
  y: yData,
  z: zData,
  mode: 'markers',
  type: 'scatter3d',
  marker: {
    size: 5,
    color: colorValues,
    colorscale: 'Viridis',
    opacity: 0.8
  }
}];

Plotly.newPlot('scatter3d', scatter3dData, {
  scene: {
    xaxis: { title: 'X Axis' },
    yaxis: { title: 'Y Axis' },
    zaxis: { title: 'Z Axis' }
  }
});

// Real-time streaming update
function updateChart(newPoint) {
  Plotly.extendTraces('timeseries', {
    x: [[newPoint.timestamp]],
    y: [[newPoint.value]]
  }, [0]);
}
```

25.3.1 Python with Plotly Express

```
import plotly.express as px
import pandas as pd

# Load data
df = pd.read_parquet('sales_data.parquet')

# Interactive scatter with trendline
fig = px.scatter(
    df,
    x='revenue',
    y='profit',
    color='category',
    size='volume',
    hover_data=['product_name'],
    trendline='ols',
    title='Revenue vs Profit by Category'
)

fig.update_layout(template='plotly_dark')
fig.show()

# Animated time series
fig = px.line(
    df,
    x='date',
    y='value',
    color='region',
    animation_frame='year',
    range_y=[0, df['value'].max() * 1.1]
)
fig.show()
```

25.4 URL

https://plotly.com/javascript/

25.5 Books

- **Interactive Data Visualization with Python and Plotly** (Packt)
- **Plotly Dash** by Adam Schroeder – Building analytics applications

25.6 Hardening

```
// Sanitize data before plotting
function sanitizePlotData(data) {
  return data.map(trace => ({
    ...trace,
    x: trace.x.filter(v => isFinite(v)),
    y: trace.y.filter(v => isFinite(v)),
    text: trace.text?.map(t => DOMPurify.sanitize(String
       (t)))
  }));
}

// Disable editable mode for untrusted users
const config = {
  editable: false,
  displayModeBar: true,
  modeBarButtonsToRemove: ['sendDataToCloud']
};

Plotly.newPlot('chart', data, layout, config);
```

Chapter 26

Vega and Vega-Lite

26.1 What Vega Is

Vega is a declarative visualization grammar that describes visualizations as JSON specifications. Developed by the UW Interactive Data Lab, it provides a complete system for creating, saving, and sharing visualizations without writing code. Vega-Lite is a higher-level grammar built on Vega that uses sensible defaults to enable rapid visualization authoring.

The Vega ecosystem powers many popular tools including Altair (Python), Observable notebooks, and various BI platforms. Its specification-based approach makes visualizations portable, reproducible, and easy to version control.

26.1.1 Vega vs Vega-Lite

Aspect	**Vega**	**Vega-Lite**
Complexity	Low-level, verbose	High-level, concise
Control	Fine-grained	Sensible defaults
Use Case	Custom visualizations	Rapid exploration
Learning Curve	Steeper	Gentle

26.2 Install Instructions

```
# JavaScript
npm install vega vega-lite vega-embed

# Python (Altair)
pip install altair vega_datasets
```

26.3 Working Example

26.3.1 Vega-Lite Specification

```
// Interactive scatter plot with Vega-Lite
const spec = {
  "$schema": "https://vega.github.io/schema/vega-lite/v5
     .json",
  "data": { "url": "/api/sales" },
  "mark": "point",
  "encoding": {
    "x": {
      "field": "revenue",
      "type": "quantitative",
      "scale": { "zero": false }
    },
    "y": {
      "field": "profit",
      "type": "quantitative"
    },
    "color": {
      "field": "category",
      "type": "nominal"
    },
    "size": {
      "field": "volume",
      "type": "quantitative"
    },
    "tooltip": [
      { "field": "product", "type": "nominal" },
      { "field": "revenue", "type": "quantitative", "
         format": "$,.0f" }
```

```
    ]
  },
  "selection": {
    "brush": { "type": "interval" }
  },
  "width": 600,
  "height": 400
};

// Embed in page
vegaEmbed('#chart', spec, { theme: 'dark' });
```

26.3.2 Python with Altair

```
import altair as alt
import pandas as pd

df = pd.read_parquet('sales.parquet')

# Interactive linked charts
brush = alt.selection_interval()

points = alt.Chart(df).mark_point().encode(
    x='revenue:Q',
    y='profit:Q',
    color=alt.condition(brush, 'category:N', alt.value('
       lightgray'))
).add_selection(brush).properties(width=400, height=300)

bars = alt.Chart(df).mark_bar().encode(
    x='count()',
    y='category:N',
    color='category:N'
).transform_filter(brush).properties(width=400, height
   =300)

chart = points | bars
chart.save('dashboard.html')
```

26.4 URL

https://vega.github.io

26.5 Books

- **Visualization Analysis and Design** by Tamara Munzner – Theory behind Vega
- **Altair: Declarative Visualization in Python** – Official documentation

26.6 Hardening

```
// Validate Vega-Lite spec before rendering
import { compile } from 'vega-lite';

function safeRender(spec) {
  try {
    // Validate and compile
    const vegaSpec = compile(spec).spec;

    // Disable external data URLs in production
    if (vegaSpec.data?.url && !isAllowedUrl(vegaSpec.
       data.url)) {
      throw new Error('External data URLs not allowed');
    }

    return vegaEmbed('#chart', spec, {
      loader: { http: { credentials: 'same-origin' } }
    });
  } catch (err) {
    console.error('Invalid spec:', err);
  }
}
```

Chapter 27

Grafana

27.1 What Grafana Is

Grafana is the leading open-source platform for monitoring, observability, and data visualization. It enables teams to query, visualize, alert on, and understand metrics from any data source. Originally focused on time-series data for infrastructure monitoring, Grafana has evolved into a comprehensive analytics platform supporting business intelligence use cases.

27.1.1 Key Features

- **Unified Dashboards**: Single pane of glass for all data sources
- **60+ Data Sources**: Prometheus, InfluxDB, Elasticsearch, PostgreSQL, MySQL, and more
- **Alerting**: Unified alerting with multiple notification channels
- **Annotations**: Event markers on time-series charts
- **Transformations**: Query result processing without backend changes
- **Templating**: Dynamic dashboards with variables
- **Provisioning**: Infrastructure-as-code dashboard deployment

27.2 Install Instructions

27.2.1 Docker Installation

```
# Quick start with Docker
docker run -d \
  --name=grafana \
  -p 3000:3000 \
  -v grafana-storage:/var/lib/grafana \
  grafana/grafana-oss:latest

# With docker-compose
cat > docker-compose.yml << 'EOF'
version: '3.8'
services:
  grafana:
    image: grafana/grafana-oss:latest
    ports:
      - 3000:3000
    environment:
      GF_SECURITY_ADMIN_PASSWORD: admin
      GF_INSTALL_PLUGINS: grafana-clock-panel,grafana-
         piechart-panel
    volumes:
      - grafana-data:/var/lib/grafana
      - ./provisioning:/etc/grafana/provisioning

volumes:
  grafana-data:
EOF

docker-compose up -d
```

27.2.2 APT Installation (Debian/Ubuntu)

```
sudo apt-get install -y apt-transport-https software-
   properties-common wget
sudo mkdir -p /etc/apt/keyrings/
wget -q -O - https://apt.grafana.com/gpg.key | gpg --
   dearmor | sudo tee /etc/apt/keyrings/grafana.gpg > /
   dev/null
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg]
   https://apt.grafana.com stable main" | sudo tee /etc/
   apt/sources.list.d/grafana.list
```

```
sudo apt-get update
sudo apt-get install grafana
sudo systemctl enable --now grafana-server
```

27.3 Working Example: Big Data Monitoring Dashboard

27.3.1 Dashboard Provisioning

```
# provisioning/dashboards/dashboard.yml
apiVersion: 1
providers:
  - name: 'default'
    orgId: 1
    folder: 'Big Data'
    type: file
    disableDeletion: false
    updateIntervalSeconds: 10
    options:
      path: /etc/grafana/provisioning/dashboards
```

27.3.2 Dashboard JSON

```
{
  "title": "Kafka Cluster Monitoring",
  "uid": "kafka-cluster",
  "panels": [
    {
      "title": "Messages Per Second",
      "type": "timeseries",
      "datasource": "Prometheus",
      "targets": [{
        "expr": "sum(rate(
           kafka_server_brokertopicmetrics_messagesinpersec
           [5m])) by (topic)",
        "legendFormat": "{{topic}}"
      }],
      "fieldConfig": {
        "defaults": {
```

```
          "unit": "msg/s",
          "color": { "mode": "palette-classic" }
        }
      }
    },
    {
      "title": "Consumer Lag",
      "type": "gauge",
      "datasource": "Prometheus",
      "targets": [{
        "expr": "sum(kafka_consumergroup_lag) by (
           consumergroup)"
      }],
      "fieldConfig": {
        "defaults": {
          "thresholds": {
            "steps": [
              { "color": "green", "value": 0 },
              { "color": "yellow", "value": 1000 },
              { "color": "red", "value": 10000 }
            ]
          }
        }
      }
    }
  ],
  "templating": {
    "list": [{
      "name": "topic",
      "type": "query",
      "datasource": "Prometheus",
      "query": "label_values(
         kafka_server_brokertopicmetrics_messagesinpersec
         , topic)"
    }]
  }
}
```

27.3.3 Alerting Configuration

```
# provisioning/alerting/rules.yml
apiVersion: 1
groups:
  - name: kafka-alerts
    folder: Big Data
    interval: 1m
    rules:
      - uid: kafka-lag-high
        title: High Consumer Lag
        condition: C
        data:
          - refId: A
            queryType: ''
            datasourceUid: prometheus
            model:
              expr: sum(kafka_consumergroup_lag) by (
                 consumergroup)
          - refId: C
            datasourceUid: '-100'
            model:
              conditions:
                - evaluator:
                    params: [10000]
                    type: gt
        for: 5m
        annotations:
          summary: Consumer lag is above 10k
        labels:
          severity: warning
```

27.4 Integration with Big Data Stack

27.4.1 Apache Druid Data Source

```
# provisioning/datasources/druid.yml
apiVersion: 1
datasources:
  - name: Druid
    type: grafana-druid-datasource
    url: http://druid-broker:8082
    jsonData:
      connection:
        url: http://druid-broker:8082/druid/v2/sql
```

27.5 URL

https://grafana.com

27.6 Books

- **Prometheus: Up and Running** (O'Reilly) – Grafana integration
- **Hands-On Infrastructure Monitoring with Prometheus** (Packt)

27.7 Hardening

```
# /etc/grafana/grafana.ini
[server]
protocol = https
cert_file = /etc/grafana/ssl/grafana.crt
cert_key = /etc/grafana/ssl/grafana.key

[security]
admin_password = $__file{/run/secrets/admin_password}
disable_gravatar = true
cookie_secure = true
strict_transport_security = true

[auth]
disable_login_form = false
```

```
oauth_auto_login = true

[auth.generic_oauth]
enabled = true
name = OAuth
allow_sign_up = true
client_id = $__env{OAUTH_CLIENT_ID}
client_secret = $__env{OAUTH_CLIENT_SECRET}
scopes = openid email profile
auth_url = https://auth.example.com/authorize
token_url = https://auth.example.com/oauth/token
```

Chapter 28

Redash

28.1 What Redash Is

Redash is an open-source data visualization and dashboarding tool designed for data teams who need to query multiple data sources and share insights. It provides a SQL-first approach where analysts write queries directly, making it ideal for teams comfortable with SQL who want quick access to data without complex setup.

28.1.1 Key Features

- **SQL-First**: Write queries in your database's native SQL
- **50+ Data Sources**: PostgreSQL, MySQL, BigQuery, Redshift, Snowflake, etc.
- **Query Snippets**: Reusable SQL fragments
- **Parameterized Queries**: Dynamic queries with user inputs
- **Scheduled Refresh**: Automatic query execution
- **Alerts**: Notifications when query results meet conditions
- **API Access**: Programmatic query execution

28.2 Install Instructions

```
# Docker Compose installation
git clone https://github.com/getredash/setup.git
cd setup
docker-compose up -d

# Or manual Docker setup
cat > docker-compose.yml << 'EOF'
version: '3.8'
services:
  redash:
    image: redash/redash:latest
    command: server
    ports:
      - 5000:5000
    environment:
      REDASH_DATABASE_URL: postgresql://redash:
         redash@postgres/redash
      REDASH_REDIS_URL: redis://redis:6379/0
      REDASH_SECRET_KEY: your-secret-key
    depends_on:
      - postgres
      - redis

  worker:
    image: redash/redash:latest
    command: scheduler
    environment:
      REDASH_DATABASE_URL: postgresql://redash:
         redash@postgres/redash
      REDASH_REDIS_URL: redis://redis:6379/0

  postgres:
    image: postgres:15
    environment:
      POSTGRES_USER: redash
      POSTGRES_PASSWORD: redash

  redis:
    image: redis:7-alpine
EOF
```

```
docker-compose up -d
docker-compose run --rm redash create_db
```

28.3 Working Example

28.3.1 Parameterized Query

```
-- Sales report with date and category filters
SELECT
  DATE_TRUNC('day', order_date) AS date,
  category,
  SUM(revenue) AS total_revenue,
  COUNT(*) AS order_count
FROM orders o
JOIN products p ON o.product_id = p.id
WHERE order_date BETWEEN '{{start_date}}' AND '{{
   end_date}}'
  AND category = '{{category}}'
GROUP BY 1, 2
ORDER BY 1
```

28.3.2 Python API Access

```
import requests

REDASH_URL = 'https://redash.example.com'
API_KEY = 'your-api-key'

# Execute query
response = requests.post(
    f'{REDASH_URL}/api/queries/42/results',
    headers={'Authorization': f'Key {API_KEY}'},
    json={'parameters': {'start_date': '2024-01-01', '
       category': 'Electronics'}}
)

data = response.json()['query_result']['data']['rows']
```

28.4 URL

https://redash.io

28.5 Books

- **Redash Documentation** – Official guide and tutorials

28.6 Hardening

```
# Environment configuration for security
REDASH_ENFORCE_HTTPS=true
REDASH_COOKIE_SECRET=$(openssl rand -hex 32)
REDASH_SECRET_KEY=$(openssl rand -hex 32)
REDASH_PASSWORD_LOGIN_ENABLED=false
REDASH_GOOGLE_CLIENT_ID=your-google-client-id
REDASH_GOOGLE_CLIENT_SECRET=your-google-secret

# Restrict data source access
REDASH_FEATURE_ALLOW_ALL_TO_EDIT_PUBLISHED_QUERIES=false
```

Chapter 29

Metabase

29.1 What Metabase Is

Metabase is an open-source business intelligence tool that enables anyone in an organization to ask questions and learn from data without requiring SQL knowledge. It provides a visual query builder, automatic chart suggestions, and self-service analytics capabilities that make it popular for democratizing data access.

29.1.1 Key Features

- **Visual Query Builder**: Point-and-click query construction
- **SQL Mode**: Advanced users can write native SQL
- **Automatic Insights**: X-ray feature for instant data exploration
- **Data Models**: Define relationships and semantic layer
- **Embedding**: White-label dashboards in applications
- **Pulse & Alerts**: Scheduled reports and threshold notifications
- **Collections**: Organized dashboards and questions

29.2 Install Instructions

29.2.1 Docker Installation

```
# Quick start with Docker
docker run -d \
  --name metabase \
  -p 3000:3000 \
  -v metabase-data:/metabase.db \
  metabase/metabase:latest

# Production with external database
cat > docker-compose.yml << 'EOF'
version: '3.8'
services:
  metabase:
    image: metabase/metabase:latest
    ports:
      - 3000:3000
    environment:
      MB_DB_TYPE: postgres
      MB_DB_HOST: postgres
      MB_DB_PORT: 5432
      MB_DB_DBNAME: metabase
      MB_DB_USER: metabase
      MB_DB_PASS: metabase_password
      MB_ENCRYPTION_SECRET_KEY: your-secret-key
    depends_on:
      - postgres

  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: metabase
      POSTGRES_USER: metabase
      POSTGRES_PASSWORD: metabase_password
    volumes:
      - postgres-data:/var/lib/postgresql/data

volumes:
  postgres-data:
EOF

docker-compose up -d
```

29.2.2 JAR Installation

```
# Download latest release
wget https://downloads.metabase.com/latest/metabase.jar

# Run with Java
java -jar metabase.jar

# With external PostgreSQL database
export MB_DB_TYPE=postgres
export MB_DB_HOST=localhost
export MB_DB_DBNAME=metabase
export MB_DB_USER=metabase
export MB_DB_PASS=password
java -jar metabase.jar
```

29.3 Working Example: Embedded Analytics

```
// Embed Metabase dashboard in React application
import React from 'react';
import jwt from 'jsonwebtoken';

const METABASE_SITE_URL = 'https://metabase.example.com'
   ;
const METABASE_SECRET_KEY = process.env.
   METABASE_SECRET_KEY;

function EmbeddedDashboard({ dashboardId, params }) {
  // Generate signed embedding token
  const payload = {
    resource: { dashboard: dashboardId },
    params: params,
    exp: Math.round(Date.now() / 1000) + (10 * 60)  //
       10 minutes
  };

  const token = jwt.sign(payload, METABASE_SECRET_KEY);

  const iframeUrl = `${METABASE_SITE_URL}/embed/
```

```
    dashboard/${token}#bordered=false&titled=true`;

  return (
    <iframe
      src={iframeUrl}
      frameBorder="0"
      width="100%"
      height="600"
      allowTransparency
    />
  );
}

export default function AnalyticsDashboard({ customerId
  }) {
  return (
    <EmbeddedDashboard
      dashboardId={42}
      params={{ customer_id: customerId }}
    />
  );
}
```

29.4 URL

https://www.metabase.com

29.5 Books

- **Metabase Documentation** – Official comprehensive guide
- **Self-Service Analytics** – Business intelligence best practices

29.6 Hardening

```
# Security environment variables
MB_ENCRYPTION_SECRET_KEY=$(openssl rand -hex 32)
MB_PASSWORD_COMPLEXITY=strong
MB_PASSWORD_LENGTH=12
MB_SESSION_COOKIES=true
MB_COOKIE_SAMESITE=strict

# LDAP/SSO configuration
MB_LDAP_ENABLED=true
MB_LDAP_HOST=ldap.example.com
MB_LDAP_SECURITY=ssl

# Disable public sharing if not needed
MB_ENABLE_PUBLIC_SHARING=false
MB_ENABLE_EMBEDDING=true
MB_EMBEDDING_SECRET_KEY=$(openssl rand -hex 32)
```

Chapter 30

Observable Plot

30.1 What Observable Plot Is

Observable Plot is a modern JavaScript library for exploratory data visualization developed by the creators of D3.js. It provides a concise, declarative API that prioritizes rapid iteration and expressiveness over customization. Plot is designed for data analysis workflows where you need to quickly visualize data to understand patterns before committing to a polished presentation.

30.1.1 Key Features

- **Concise API**: Minimal code for common visualizations
- **Sensible Defaults**: Automatic scales, axes, and legends
- **Composable Marks**: Layer multiple marks in a single plot
- **Transforms**: Built-in binning, grouping, and statistical transforms
- **Faceting**: Multi-panel charts for categorical comparisons
- **Responsive**: Automatic sizing and mobile support
- **Framework Agnostic**: Works with React, Vue, vanilla JS

30.2 Install Instructions

```
npm install @observablehq/plot

# Or with D3 for additional utilities
npm install @observablehq/plot d3
```

30.3 Working Example

```
import * as Plot from '@observablehq/plot';

// Fetch data from big data backend
const salesData = await fetch('/api/sales').then(r => r.
   json());

// Simple scatter plot
const scatterPlot = Plot.plot({
  marks: [
    Plot.dot(salesData, {
      x: 'revenue',
      y: 'profit',
      fill: 'category',
      r: 'volume',
      tip: true
    })
  ],
  color: { legend: true },
  grid: true
});

document.getElementById('scatter').appendChild(
   scatterPlot);

// Time series with confidence interval
const timeSeriesPlot = Plot.plot({
  y: { grid: true, label: 'Revenue ($)' },
  marks: [
    Plot.areaY(salesData, {
      x: 'date',
      y1: 'revenue_low',
```

```
      y2: 'revenue_high',
      fill: '#4CAF50',
      fillOpacity: 0.2
    }),
    Plot.lineY(salesData, {
      x: 'date',
      y: 'revenue',
      stroke: '#4CAF50',
      strokeWidth: 2
    }),
    Plot.ruleY([0])
  ]
});

// Faceted bar chart
const facetedBars = Plot.plot({
  facet: {
    data: salesData,
    x: 'region'
  },
  marks: [
    Plot.barY(salesData, Plot.groupX(
      { y: 'sum' },
      { x: 'category', y: 'revenue', fill: 'category' }
    ))
  ],
  color: { legend: true }
});

// Hexbin for large datasets
const hexbinPlot = Plot.plot({
  marks: [
    Plot.hexbin(
      largeDataset,
      Plot.hexagon({
        x: 'x',
        y: 'y',
        fill: 'count'
      })
    )
  ],
  color: { scheme: 'YlOrRd' }
});
```

30.3.1 React Integration

```
import * as Plot from '@observablehq/plot';
import { useRef, useEffect } from 'react';

function PlotChart({ data, options }) {
  const containerRef = useRef();

  useEffect(() => {
    const plot = Plot.plot({ ...options, marks: options.
       marks(data) });
    containerRef.current.replaceChildren(plot);
    return () => plot.remove();
  }, [data, options]);

  return <div ref={containerRef} />;
}

// Usage
<PlotChart
  data={salesData}
  options={{
    marks: (data) => [
      Plot.barY(data, Plot.groupX({ y: 'sum' }, { x: '
         month', y: 'revenue' }))
    ]
  }}
/>
```

30.4 URL

https://observablehq.com/plot/

30.5 Books

- **Observable Plot Documentation** – Official comprehensive guide
- **D3.js in Action** (Manning) – Underlying concepts

30.6 Hardening

```
// Sanitize data before plotting
function safePlot(data, options) {
  const sanitizedData = data.map(d => {
    const clean = {};
    for (const [key, value] of Object.entries(d)) {
      if (typeof value === 'string') {
        clean[key] = value.slice(0, 1000);  // Limit
           string length
      } else if (typeof value === 'number' && isFinite(
          value)) {
        clean[key] = value;
      } else if (value instanceof Date) {
        clean[key] = value;
      }
    }
    return clean;
  });

  return Plot.plot({ ...options, data: sanitizedData });
}
```

Chapter 31

Summary of the Full Technology Stack

31.1 Overview

This document has covered 31 technologies spanning the entire modern big data and analytics ecosystem. These tools collectively enable organizations to ingest, process, store, analyze, and visualize data at any scale.

31.1.1 Technology Categories

Category	Technologies
Stream Processing	Apache Kafka, Apache Flink
Batch Processing	Apache Spark, Apache Hadoop
Data Lakes	Apache Iceberg
OLAP Databases	Apache Doris, Apache Druid, Apache Pinot
NoSQL Databases	Apache Cassandra, Apache HBase, AsterixDB
Data Integration	Apache NiFi, Apache Hop
Orchestration	Apache Airflow
SQL Analytics	Apache Hive, Apache Drill
Coordination	Apache ZooKeeper
Data Formats	Apache Arrow, Apache Parquet
BI Platforms	Apache Superset, Grafana, Metabase, Redash
Visualization Libraries	D3.js, ECharts, Plotly, Vega, Observable Plot
Analytics APIs	Cube.js, FINOS Perspective

Table 31.1: Technology stack summary

31.2 Decision Matrix

31.2.1 Choosing Stream Processing

Use Case	Best Choice	Why
Event streaming backbone	Kafka	Durability, ecosystem
Complex event processing	Flink	Exactly-once, state management
Simple transformations	Kafka Streams	Lightweight, no cluster

31.2.2 Choosing Analytics Storage

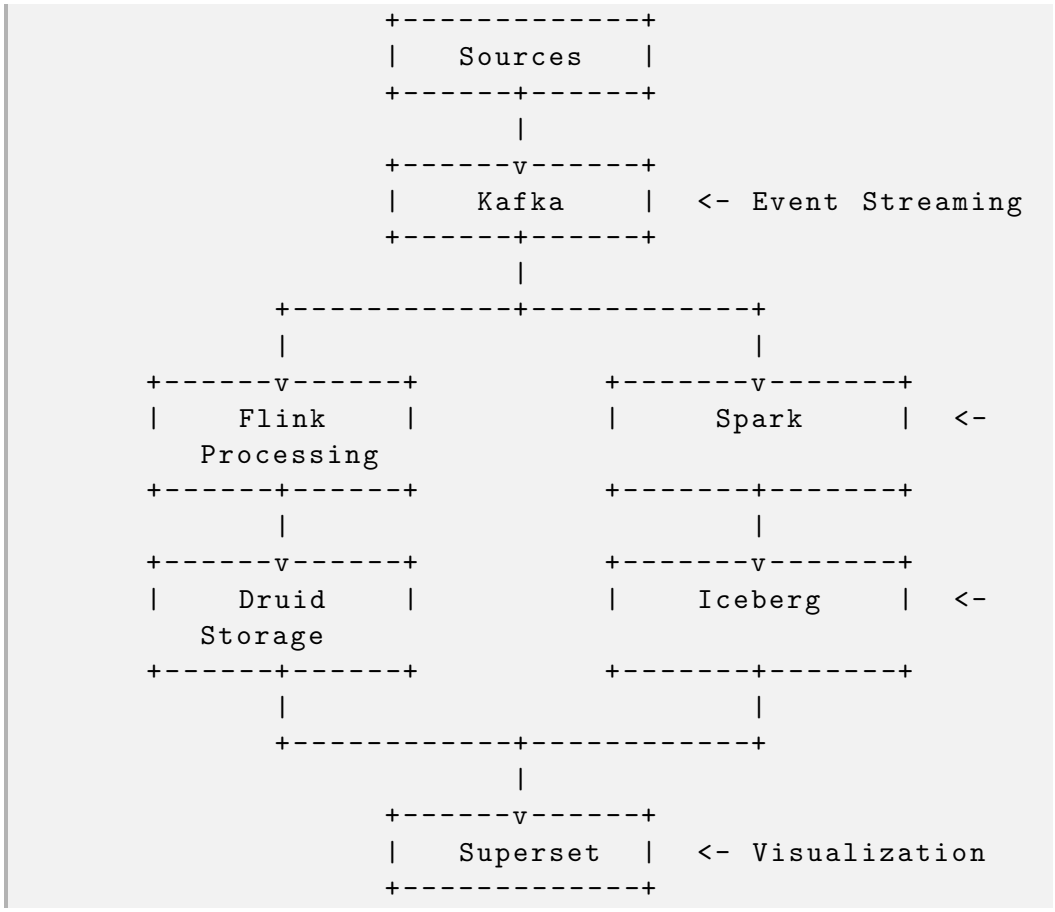
Use Case	Best Choice	Why
Sub-second OLAP	Doris, Pinot	Pre-aggregation, indexes
Time-series analytics	Druid	Time-optimized storage
Data lake analytics	Iceberg + Spark	ACID, schema evolution
Key-value lookups	HBase, Cassandra	Low latency reads

31.2.3 Choosing Visualization

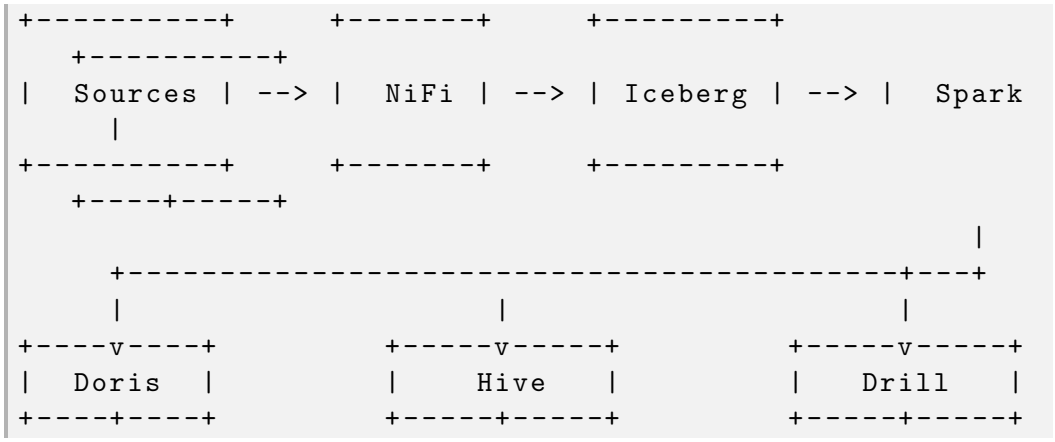
Use Case	Best Choice	Why
Enterprise BI	Superset, Metabase	Full-featured, governance
Infrastructure monitoring	Grafana	Time-series focus, alerting
Custom visualizations	D3.js	Maximum flexibility
Rapid prototyping	Observable Plot, Plotly	Quick iteration
Embedded analytics	Cube.js, Perspective	API-first, performant
Financial dashboards	FINOS Perspective	Streaming, high-frequency

31.3 Reference Architectures

31.3.1 Real-Time Analytics Platform



31.3.2 Data Lake Architecture



```
    |                     |                       |
    +---------------------+-----------------------+
                          |
                  +------v------+
                  |   Superset  |
                  +-------------+
```

31.3.3 Embedded Analytics Architecture

```
+-------------+     +-----------+     +----------+
| PostgreSQL  | --> | Cube.js   | --> | React    |
| Druid       |     | Semantic  |     | Vue      |
| BigQuery    |     | Layer     |     | Angular  |
+-------------+     +----+------+     +----------+
                         |
                +--------+---------+
                |                  |
          +-----v------+     +-----v------+
          |  REST API  |     |  GraphQL   |
          +------------+     +------------+
```

31.4 Security Best Practices

1. **Network Security**: Use TLS for all inter-service communication
2. **Authentication**: Implement SSO/SAML across all tools
3. **Authorization**: Apply principle of least privilege with RBAC
4. **Data Security**: Encrypt data at rest and in transit
5. **Audit Logging**: Enable comprehensive audit trails
6. **Secrets Management**: Use Vault or similar for credentials
7. **Network Isolation**: Segment data plane from control plane

31.5 Operational Considerations

31.5.1 Monitoring Stack

- **Metrics**: Prometheus for all components
- **Visualization**: Grafana dashboards
- **Logging**: ELK stack or Loki
- **Tracing**: Jaeger or Zipkin for distributed tracing
- **Alerting**: Grafana Alerting or PagerDuty integration

31.5.2 High Availability

- Deploy in multiple availability zones
- Use ZooKeeper or etcd for coordination
- Configure appropriate replication factors
- Implement circuit breakers and retry logic
- Regular backup and disaster recovery testing

31.6 Getting Started

31.6.1 Recommended Learning Path

1. Start with **Kafka** for event streaming fundamentals
2. Add **Spark** or **Flink** for data processing
3. Deploy **Superset** or **Metabase** for visualization
4. Implement **Airflow** for workflow orchestration
5. Scale with **Iceberg** and OLAP databases as needed

31.6.2 Proof of Concept Template

```yaml
# docker-compose.yml for minimal POC
version: '3.8'
services:
  kafka:
    image: confluentinc/cp-kafka:latest
    ports: ["9092:9092"]

  flink:
    image: flink:latest
    ports: ["8081:8081"]

  postgres:
    image: postgres:15
    ports: ["5432:5432"]

  superset:
    image: apache/superset:latest
    ports: ["8088:8088"]
```

31.7 Conclusion

The Apache big data ecosystem provides a comprehensive set of tools for building modern data platforms. By understanding each technology's strengths and appropriate use cases, architects can design systems that balance performance, scalability, and operational complexity.

Key principles for success:

- **Start simple**: Begin with proven technologies and add complexity as needed
- **Standardize**: Use consistent patterns across the organization
- **Automate**: Infrastructure as code and CI/CD for all deployments
- **Monitor**: Comprehensive observability from day one
- **Evolve**: Continuously evaluate and adopt new technologies

The technologies covered in this document represent the current state of the art in big data processing and analytics. As the field continues to

evolve, new tools will emerge, but the fundamental patterns and architectural principles will remain relevant.