Hadoop Ecosystem Crash Course

What is the Hadoop Ecosystem?

The Hadoop ecosystem is a collection of open-source tools and frameworks designed to store, process, and analyze massive datasets in a distributed computing environment. It's built around Apache Hadoop and has grown into a comprehensive big data platform.

The Core: Hadoop Components

1. HDFS (Hadoop Distributed File System)

Purpose: Distributed storage system for big data

How it works:

- Splits large files into blocks (default 128MB or 256MB)
- Replicates blocks across multiple nodes (default 3 copies)
- Provides fault tolerance through replication
- Optimized for large files and sequential reads

Key Concepts:

- **NameNode**: Master server that manages metadata and file system namespace
- **DataNode**: Worker nodes that store actual data blocks
- **Secondary NameNode**: Performs periodic checkpoints (not a backup!)

Use Cases:

- Storing petabytes of data
- Log aggregation
- Data lake storage
- Archiving large datasets

Example Commands:

```
# Upload file to HDFS
hdfs dfs -put localfile.txt /user/data/

# List files
hdfs dfs -ls /user/data/

# Download file
hdfs dfs -get /user/data/file.txt

# Check disk usage
hdfs dfs -du -h /user/data/
```

```
# Remove file
hdfs dfs -rm /user/data/file.txt
```

2. YARN (Yet Another Resource Negotiator)

Purpose: Cluster resource management and job scheduling

Architecture:

- **ResourceManager**: Global resource scheduler (one per cluster)
- **NodeManager**: Per-node agent managing resources (one per node)
- **ApplicationMaster**: Per-application coordinator
- **Container**: Unit of resource allocation (CPU, memory, disk)

What it does:

- Allocates resources to applications
- Schedules tasks across cluster
- Monitors application execution
- Supports multiple processing frameworks

Schedulers:

- **FIFO**: First-in, first-out (simple but unfair)
- **Capacity**: Multiple queues with guaranteed capacity
- **Fair**: Dynamic fair sharing between users/apps

3. MapReduce

Purpose: Original batch processing framework

Covered in previous crash course, but key points:

- Parallel processing of large datasets
- Two-phase computation: Map → Reduce
- Disk-based processing (slower than in-memory)
- Best for batch jobs, not real-time

Status: Still used but largely replaced by Spark for most use cases

Processing & Querying

4. Apache Spark

Purpose: Fast, general-purpose distributed computing engine

Why Spark over MapReduce:

- **10-100x faster**: In-memory processing
- **Easier to use**: High-level APIs in Python, Scala, Java, R
- **Unified platform**: Batch, streaming, ML, graph processing

- **Interactive**: REPL for ad-hoc analysis

Core Components:

- **Spark Core**: Basic functionality and RDDs
- **Spark SQL**: Structured data processing
- **Spark Streaming**: Real-time stream processing
- **MLlib**: Machine learning library
- **GraphX**: Graph processing

Example (PySpark):

```
from pyspark.sql import SparkSession

# Create Spark session
spark = SparkSession.builder.appName("example").getOrCreate()

# Read data
df = spark.read.csv("hdfs://data/file.csv", header=True)

# Process
result = df.filter(df['age'] > 25) \
           .groupBy('country') \
           .count()

# Write output
result.write.csv("hdfs://output/")
```

When to use: Almost any big data processing task (batch, streaming, ML)

5. Apache Hive

Purpose: SQL-on-Hadoop data warehouse

What it does:

- Provides SQL interface to data in HDFS
- Converts HiveQL queries to MapReduce/Spark jobs
- Manages metadata in Hive Metastore
- Supports partitioning and bucketing

Architecture:

- **Hive Metastore**: Stores schema and metadata
- **HiveServer2**: Accepts queries from clients
- **Execution Engine**: MapReduce, Tez, or Spark

Example:

```
-- Create table
CREATE TABLE users (
    id INT,
    name STRING,
    age INT,
    country STRING
)
PARTITIONED BY (year INT, month INT)
STORED AS PARQUET;

-- Load data
LOAD DATA INPATH '/data/users.csv' INTO TABLE users;

-- Query
SELECT country, COUNT(*) as user_count
FROM users
WHERE age > 25
GROUP BY country
ORDER BY user_count DESC;
```

When to use:

- SQL analysts working with big data
- Data warehousing on Hadoop
- ETL batch jobs
- Creating tables from HDFS files

6. Apache Pig

Purpose: High-level data flow scripting language

What it does:

- Provides Pig Latin language (procedural, not SQL)
- Converts scripts to MapReduce jobs
- Good for ETL and data transformations

Example:

```
-- Load data
users = LOAD '/data/users.txt' USING PigStorage(',')
        AS (id:int, name:chararray, age:int, country:chararray);

-- Filter
adults = FILTER users BY age > 25;

-- Group
by_country = GROUP adults BY country;

-- Count
```

```
counts = FOREACH by_country GENERATE
         group as country,
         COUNT(adults) as user_count;

-- Store
STORE counts INTO '/output/results';
```

When to use:

- Complex ETL pipelines
- When SQL is too rigid
- Data scientists comfortable with scripting

Status: Less popular now, Spark often preferred

7. Apache Impala

Purpose: Real-time SQL queries on Hadoop

Key Features:

- **Massively Parallel Processing (MPP)** SQL engine
- **Low latency**: Sub-second to seconds (vs minutes for Hive)
- **In-memory processing**: No MapReduce overhead
- Uses Hive Metastore but bypasses MapReduce

When to use:

- Interactive analytics
- BI dashboards on Hadoop data
- When Hive is too slow

Alternatives: Presto, Apache Drill

8. Apache Flink

Purpose: Stream processing and batch processing

Key Features:

- True stream processing (not micro-batching like Spark Streaming)
- Event time processing
- Exactly-once semantics
- Low latency (milliseconds)

When to use:

- Real-time analytics
- Complex event processing
- Stateful stream processing
- When Spark Streaming latency is too high

Data Ingestion

9. Apache Flume

Purpose: Collecting, aggregating, and moving log data

Architecture:

- **Source**: Receives data (e.g., log files, syslog)
- **Channel**: Buffers data (memory or file-based)
- **Sink**: Writes data to destination (HDFS, HBase, Kafka)

Example Configuration:

```
# Define source, channel, sink
agent.sources = logsource
agent.channels = memchannel
agent.sinks = hdfssink

# Source configuration
agent.sources.logsource.type = exec
agent.sources.logsource.command = tail -F /var/log/app.log

# Channel configuration
agent.channels.memchannel.type = memory
agent.channels.memchannel.capacity = 10000

# Sink configuration
agent.sinks.hdfssink.type = hdfs
agent.sinks.hdfssink.hdfs.path = /logs/app/%Y-%m-%d/
agent.sinks.hdfssink.hdfs.fileType = DataStream

# Bind source and sink to channel
agent.sources.logsource.channels = memchannel
agent.sinks.hdfssink.channel = memchannel
```

When to use:

- Log collection from servers
- Moving data from edge to Hadoop
- Reliable data pipelines

10. Apache Sqoop

Purpose: Transferring data between Hadoop and relational databases

What it does:

- Import data from RDBMS (MySQL, PostgreSQL, Oracle) to HDFS/Hive
- Export data from Hadoop back to RDBMS
- Parallel data transfer

- Incremental imports

Example:

```
# Import entire table
sqoop import \
  --connect jdbc:mysql://localhost/mydb \
  --username user \
  --password pass \
  --table employees \
  --target-dir /user/data/employees

# Import with query
sqoop import \
  --connect jdbc:mysql://localhost/mydb \
  --username user \
  --password pass \
  --query "SELECT * FROM employees WHERE \$CONDITIONS AND dept='sales'" \
  --split-by employee_id \
  --target-dir /user/data/sales_employees

# Export to database
sqoop export \
  --connect jdbc:mysql://localhost/mydb \
  --username user \
  --password pass \
  --table employees_summary \
  --export-dir /user/data/summary
```

When to use:

- ETL from traditional databases
- Migrating data to Hadoop
- Syncing Hadoop results back to databases

11. Apache Kafka

Purpose: Distributed streaming platform

What it does:

- Publishes and subscribes to streams of records
- Stores streams durably
- Processes streams in real-time
- Acts as a buffer between producers and consumers

Key Concepts:

- **Topic**: Category of messages
- **Producer**: Writes messages to topics
- **Consumer**: Reads messages from topics

- **Broker**: Kafka server
- **Partition**: Parallelism unit within a topic

Example (Python):

```
from kafka import KafkaProducer, KafkaConsumer

# Producer
producer = KafkaProducer(bootstrap_servers='localhost:9092')
producer.send('web-logs', b'user clicked button')

# Consumer
consumer = KafkaConsumer('web-logs',
                         bootstrap_servers='localhost:9092',
                         group_id='log-processors')

for message in consumer:
    print(f"Received: {message.value}")
```

When to use:

- Real-time data pipelines
- Event streaming
- Decoupling data producers from consumers
- Building real-time applications

NoSQL Storage

12. Apache HBase

Purpose: Distributed, column-oriented NoSQL database on HDFS

Key Features:

- Random, real-time read/write access
- Billions of rows, millions of columns
- Automatic sharding
- Strong consistency

Data Model:

- **Table**: Collection of rows
- **Row**: Identified by row key
- **Column Family**: Group of columns
- **Cell**: Intersection of row and column, versioned

Example (HBase Shell):

```
# Create table
create 'users', 'personal', 'professional'

# Put data
put 'users', 'user1', 'personal:name', 'John'
put 'users', 'user1', 'personal:age', '30'
put 'users', 'user1', 'professional:title', 'Engineer'

# Get data
get 'users', 'user1'

# Scan table
scan 'users'
```

When to use:

- Random read/write access to big data
- Time-series data
- Real-time analytics
- Need for strong consistency

Alternatives: Apache Cassandra, Google Bigtable

13. Apache Kudu

Purpose: Fast analytics on fast data

Key Features:

- Columnar storage with fast scans
- Fast random access (like HBase)
- Fast updates and deletes
- Integrates well with Impala and Spark

When to use:

- Need both analytics and updates
- Real-time analytics on changing data
- Time-series with updates

Coordination & Workflow

14. Apache ZooKeeper

Purpose: Distributed coordination service

What it provides:

- Configuration management
- Naming service
- Distributed synchronization

- Leader election
- Group services

Used by: HBase, Kafka, Hadoop HA, Storm, and many others

Key Concepts:

- **ZNode**: Data node in ZooKeeper namespace
- **Watch**: Notification mechanism for changes
- **Ensemble**: Cluster of ZooKeeper servers

When to use:

- Coordinating distributed systems
- Service discovery
- Distributed locks
- Configuration management

15. Apache Oozie

Purpose: Workflow scheduler for Hadoop jobs

What it does:

- Schedules workflows of Hadoop jobs
- Supports MapReduce, Pig, Hive, Sqoop, etc.
- Manages job dependencies
- Time-based and data-based triggers

Example Workflow:

```
<workflow-app name="data-pipeline" xmlns="uri:oozie:workflow:0.5">
    <start to="sqoop-import"/>

    <action name="sqoop-import">
        <sqoop xmlns="uri:oozie:sqoop-action:0.4">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <command>import --table users --target-dir /data/users</command>
        </sqoop>
        <ok to="hive-process"/>
        <error to="fail"/>
    </action>

    <action name="hive-process">
        <hive xmlns="uri:oozie:hive-action:0.5">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <script>process_users.hql</script>
        </hive>
        <ok to="end"/>
        <error to="fail"/>
```

```
    </action>

    <kill name="fail">
        <message>Workflow failed, error
message[${wf:errorMessage(wf:lastErrorNode())}]</message>
    </kill>

    <end name="end"/>
</workflow-app>
```

When to use:

- Orchestrating complex Hadoop workflows
- Scheduling batch jobs
- Managing job dependencies

Modern Alternative: Apache Airflow

16. Apache Airflow

Purpose: Modern workflow orchestration platform

Why it's better than Oozie:

- Python-based DAGs (more flexible than XML)
- Rich UI for monitoring
- Extensible with custom operators
- Better error handling and retries
- Active development

Example DAG:

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.providers.apache.spark.operators.spark_submit import
SparkSubmitOperator
from datetime import datetime, timedelta

default_args = {
    'owner': 'data-team',
    'depends_on_past': False,
    'start_date': datetime(2024, 1, 1),
    'retries': 3,
    'retry_delay': timedelta(minutes=5),
}

dag = DAG(
    'data_pipeline',
    default_args=default_args,
    schedule_interval='@daily',
)
```

```
# Task 1: Import data
import_data = BashOperator(
    task_id='import_data',
    bash_command='sqoop import --table users --target-dir /data/users',
    dag=dag,
)

# Task 2: Process with Spark
process_data = SparkSubmitOperator(
    task_id='process_data',
    application='/scripts/process_users.py',
    dag=dag,
)

# Task 3: Export results
export_results = BashOperator(
    task_id='export_results',
    bash_command='sqoop export --table user_summary --export-dir /output',
    dag=dag,
)

# Define dependencies
import_data >> process_data >> export_results
```

When to use: Almost always prefer over Oozie for new projects

Resource Management & Security

17. Apache Ranger

Purpose: Centralized security administration

Features:

- Fine-grained authorization for Hadoop components
- Centralized policy management
- Audit logging
- Data masking and filtering

When to use:

- Enterprise security requirements
- Compliance (GDPR, HIPAA, etc.)
- Multi-tenant clusters

18. Apache Atlas

Purpose: Data governance and metadata management

Features:

- Metadata management
- Data lineage tracking
- Data classification
- Search and discovery

When to use:

- Tracking data provenance
- Regulatory compliance
- Understanding data relationships

19. Apache Ambari

Purpose: Hadoop cluster provisioning, management, and monitoring

Features:

- Web-based UI for cluster management
- Install and configure Hadoop components
- Monitor cluster health
- Manage configurations
- Visual metrics and alerts

When to use:

- Managing Hadoop clusters
- Simplifying operations
- Monitoring cluster health

How Components Work Together

Typical Data Pipeline

```
Data Sources (Databases, Logs, APIs)
         ↓
    [Ingestion Layer]
    Sqoop (RDBMS) | Flume (Logs) | Kafka (Streams)
         ↓
    [Storage Layer]
         HDFS
         ↓
    [Processing Layer]
    Spark | Hive | Flink
         ↓
    [Output Layer]
    HBase (Real-time) | HDFS (Batch) | BI Tools
         ↓
    [Orchestration]
    Airflow | Oozie
```

Example Use Cases by Industry

E-Commerce:

- Kafka → Flink → HBase: Real-time recommendations
- Sqoop → HDFS → Spark → Hive: Daily sales analytics
- Airflow: Orchestrating ETL pipelines

Finance:

- Kafka → Spark Streaming → HBase: Fraud detection
- HDFS → Spark MLlib: Risk modeling
- Ranger: Security and compliance

Social Media:

- Flume → HDFS: Log aggregation
- Spark → HBase: User graph analysis
- Impala: Ad-hoc analytics for analysts

Choosing the Right Tools

For Storage:

- **Large files, sequential access**: HDFS
- **Random access, real-time**: HBase
- **Fast analytics with updates**: Kudu

For Processing:

- **General-purpose, performance**: Spark
- **SQL on Hadoop**: Hive (batch) or Impala (interactive)
- **Stream processing**: Flink or Kafka Streams
- **Complex ETL**: Pig or Spark

For Ingestion:

- **Databases**: Sqoop
- **Log files**: Flume
- **Real-time streams**: Kafka
- **REST APIs**: Custom applications

For Orchestration:

- **Simple workflows**: Oozie
- **Complex workflows**: Airflow

The Modern Hadoop Stack (2024-2025)

Core:

- HDFS (storage)
- YARN (resource management)
- Spark (processing)

Essential Add-ons:

- Kafka (streaming)
- Hive (SQL interface)
- Airflow (orchestration)
- HBase/Kudu (NoSQL)

Cloud Alternatives:

- AWS EMR (managed Hadoop)
- Google Dataproc
- Azure HDInsight
- Databricks (Spark-focused)

Hadoop vs Cloud Data Platforms

When to use Hadoop:

- On-premise requirements
- Existing Hadoop investment
- Data sovereignty concerns
- Full control over infrastructure

When to use Cloud:

- Starting fresh
- Want managed services
- Need elasticity
- Focus on applications, not infrastructure

Trend: Many organizations migrating from on-prem Hadoop to cloud platforms (S3 + EMR, Databricks, Snowflake)

Quick Reference

Component	Purpose	Speed	Best For
HDFS	Storage	N/A	Large files
MapReduce	Batch processing	Slow	Legacy batch jobs
Spark	Processing	Fast	Most processing tasks
Hive	SQL on Hadoop	Medium	Data warehousing
Impala	Real-time SQL	Fast	Interactive queries
Flink	Stream processing	Very Fast	Real-time analytics

Component	Purpose	Speed	Best For
HBase	NoSQL database	Fast	Random access
Kafka	Message queue	Very Fast	Event streaming
Sqoop	Data transfer	Medium	Database import/export
Flume	Log collection	Fast	Log aggregation
Airflow	Orchestration	N/A	Workflow management

Conclusion

The Hadoop ecosystem has evolved from a simple MapReduce framework into a comprehensive big data platform. While the landscape is complex, understanding these core components helps you:

- Design effective data pipelines
- Choose the right tool for each task
- Build scalable data architectures
- Make informed technology decisions

Key Takeaway: You don't need to use everything. Start with HDFS + Spark + Hive, then add components as specific needs arise.