Apache Oozie

Oozie is a workflow scheduler system to manage Apache Hadoop jobs.

Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions.

Oozie Coordinator jobs are recurrent Oozie Workflow jobs triggered by time (frequency) and data availability.

Oozie is integrated with the rest of the Hadoop stack supporting several types of Hadoop jobs out of the box (such as Java map-reduce, Streaming map-reduce, Pig, Hive, Sqoop and Distcp) as well as system specific jobs (such as Java programs and shell scripts).

Oozie is a scalable, reliable and extensible system.

Oozie is a server based *Workflow Engine* specialized in running workflow jobs with actions that run Hadoop Map/Reduce and Pig jobs.

Oozie is a Java Web-Application that runs in a Java servlet-container.

For the purposes of Oozie, a workflow is a collection of actions (i.e. Hadoop Map/Reduce jobs, Pig jobs) arranged in a control dependency DAG (Directed Acyclic Graph). “control dependency” from one action to another means that the second action can’t run until the first action has completed.

Oozie workflows definitions are written in hPDL (a XML Process Definition Language similar to JBOSS JBPM jPDL).

Oozie workflow actions start jobs in remote systems (i.e. Hadoop, Pig). Upon action completion, the remote systems callback Oozie to notify the action completion, at this point Oozie proceeds to the next action in the workflow.

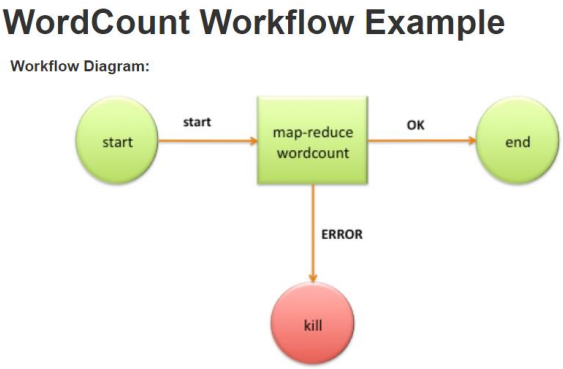
Oozie uses a custom SecurityManager inside it’s launcher to catch exit() calls from the user code. Make sure to delegate checkExit() calls to Oozie’s SecurityManager if the user code uses its own SecurityManager. The Launcher also grants java.security.AllPermission by default to the user code.

Oozie workflows contain control flow nodes and action nodes.

Control flow nodes define the beginning and the end of a workflow (`start`, `end` and `fail` nodes) and provide a mechanism to control the workflow execution path (`decision`, `fork` and `join` nodes).

Action nodes are the mechanism by which a workflow triggers the execution of a computation/processing task. Oozie provides support for different types of actions: Hadoop map-reduce, Hadoop file system, Pig, SSH, HTTP, eMail and Oozie sub-workflow. Oozie can be extended to support additional type of actions.

Oozie workflows can be parameterized (using variables like `${inputDir}` within the workflow definition). When submitting a workflow job values for the parameters must be provided. If properly parameterized (i.e. using different output directories) several identical workflow jobs can concurrently.



Apache Sqoop

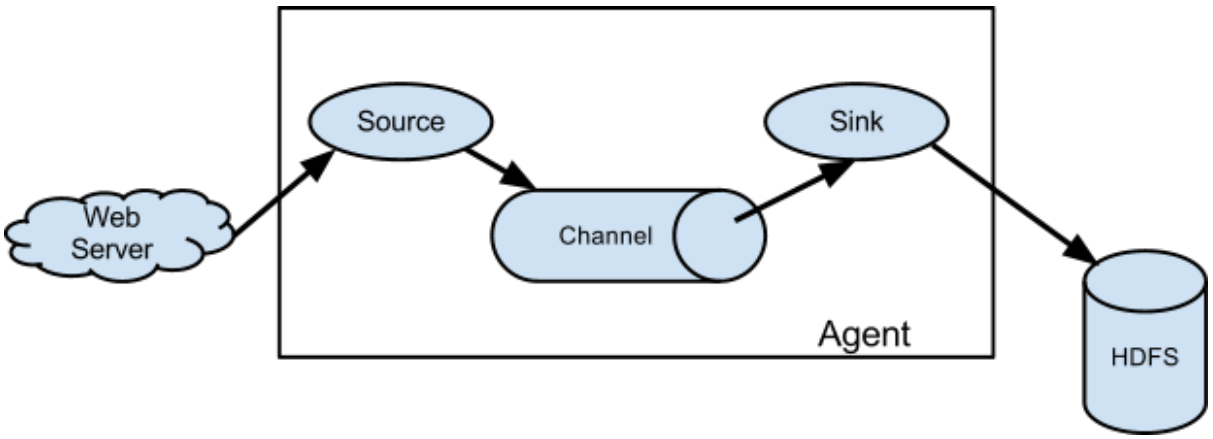
Apache Sqoop – Hadoop The Definitive Guide by Tom White , Chapter 15 -Pages 401 to 407

Apache Flume

Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application.

Data flow model

A Flume event is defined as a unit of data flow having a byte payload and an optional set of string attributes. A Flume agent is a (JVM) process that hosts the components through which events flow from an external source to the next destination (hop).



A Flume source consumes events delivered to it by an external source like a web server. The external source sends events to Flume in a format that is recognized by the target Flume source. For example, an Avro Flume source can be used to receive Avro events from Avro clients or other Flume agents in the flow that send events from an Avro sink. A similar flow can be defined using a Thrift Flume Source to receive events from a Thrift Sink or a Flume Thrift Rpc Client or Thrift clients written in any language generated from the Flume thrift protocol.When a Flume source receives an event, it stores it into one or more channels. The channel is a passive store that keeps the event until it's consumed by a Flume sink. The file channel is one example – it is backed by the local filesystem. The sink removes the event from the channel and puts it into an external repository like HDFS (via Flume HDFS sink) or forwards it to the Flume source of the next Flume agent (next hop) in the flow. The source and sink within the given agent run asynchronously with the events staged in the channel.

Complex flows

Flume allows a user to build multi-hop flows where events travel through multiple agents before reaching the final destination. It also allows fan-in and fan-out flows, contextual routing and backup routes (fail-over) for failed hops.

Reliability

The events are staged in a channel on each agent. The events are then delivered to the next agent or terminal repository (like HDFS) in the flow. The events are removed from a channel only after they are stored in the channel of next agent or in the terminal repository. This is a how the single-hop message delivery semantics in Flume provide end-to-end reliability of the flow.

Flume uses a transactional approach to guarantee the reliable delivery of the events. The sources and sinks encapsulate in a transaction the storage/retrieval, respectively, of the events placed in or provided by a transaction provided by the channel. This ensures that the set of events are reliably passed from point to point in the flow. In the case of a multi-hop flow, the sink from the previous hop and the source from the next hop both have their transactions running to ensure that the data is safely stored in the channel of the next hop.

Recoverability

The events are staged in the channel, which manages recovery from failure. Flume supports a durable file channel which is backed by the local file system. There's also a memory channel which simply stores the events in an in-memory queue, which is faster but any events still left in the memory channel when an agent process dies can't be recovered.

Flume's *KafkaChannel* uses Apache Kafka to stage events. Using a replicated Kafka topic as a channel helps avoiding event loss in case of a disk failure.

Apache Flume – Hadoop The Definitive Guide by Tom White , Chapter 14 -Pages 381 to 390

Cassandra

What is Apache Cassandra?

- Cassandra is a NoSQL database which is distributed and scalable. It is provided by Apache.
- Apache Cassandra is highly scalable, high performance, distributed NoSQL database. Cassandra is designed to handle huge amount of data across many commodity servers, providing high availability without a single point of failure.
- It is an open source, distributed and decentralized/distributed storage system (database).

History of Cassandra

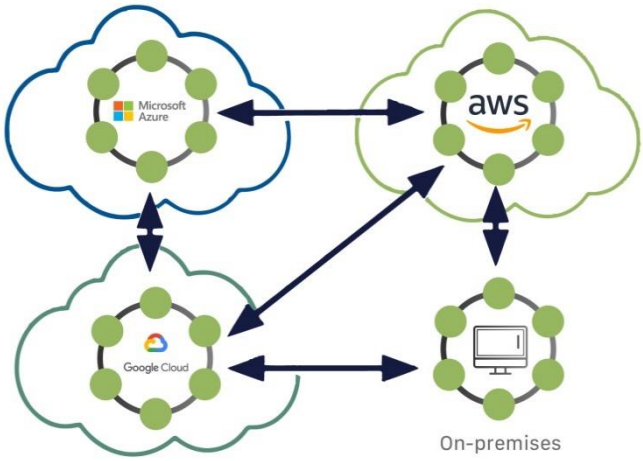
- Cassandra was initially developed at Facebook by two Indians Avinash Lakshman (one of the authors of Amazon's Dynamo) and Prashant Malik.

It was developed to power the Facebook inbox search feature. It was open sourced by Facebook in July 2008.

- It was accepted by Apache Incubator in March 2009.

Important Points of Cassandra

- Cassandra is a column-oriented database.
- Cassandra is scalable, consistent, and fault-tolerant.
- Cassandra is being used by some of the biggest companies like Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.
- Cassandra is deployment agnostic. It doesn't care where you put it – on prem, a cloud provider, multiple cloud providers.



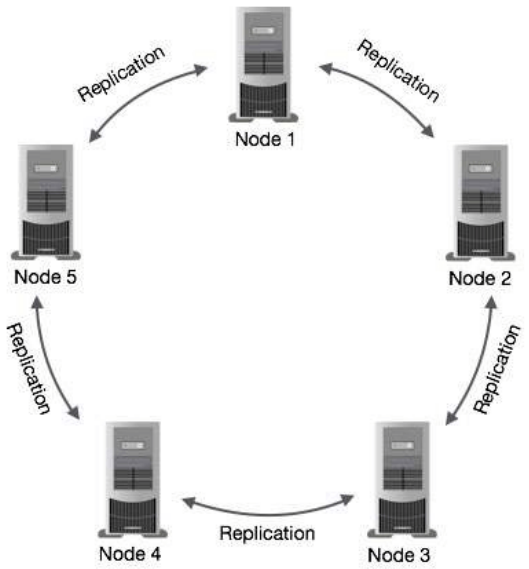
Cassandra Architecture

Cassandra was designed to handle big data workloads across multiple nodes without a single point of failure. It has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.

- In Cassandra, each node is independent and at the same time interconnected to other nodes. All the nodes in a cluster play the same role.
- Every node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
- In the case of failure of one node, Read/Write requests can be served from other nodes in the network.

Data Replication in Cassandra

- In Cassandra, nodes in a cluster act as replicas for a given piece of data.
- Cassandra will return the most recent value to the client. After returning the most recent value, Cassandra performs a read repair in the background to update the stale values.



Components of Cassandra

- **Node:** A Cassandra node is a place where data is stored.
- **Data center:** Data center is a collection of related nodes.
- **Cluster:** A cluster is a component which contains one or more data centers.
- **Commit log:** In Cassandra, the commit log is a crash-recovery mechanism. Every write operation is written to the commit log.
- **Mem-table:** A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.
- **SSTable:** It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- **Bloom filter:** These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.