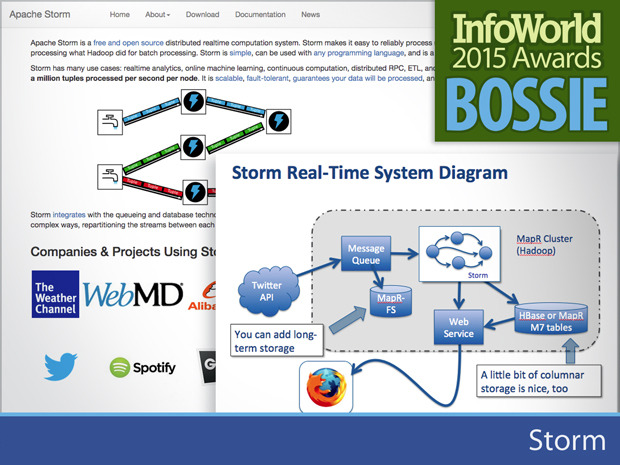
# Storm

[Apache Storm](https://storm.apache.org/) is a Clojure-based distributed computation framework primarily for streaming real-time analytics. Storm is based on the [disruptor pattern](https://lmax-exchange.github.io/disruptor/) for low-latency complex event processing created LMAX. Unlike Spark, Storm can do single events as opposed to “micro-batches,” and it has a lower memory footprint. In my experience, it scales better for streaming, especially when you’re mainly streaming to ingest data into other data sources.

Storm’s profile has been eclipsed by Spark, but Spark is inappropriate for many streaming applications. Storm is frequently used with Apache Kafka.

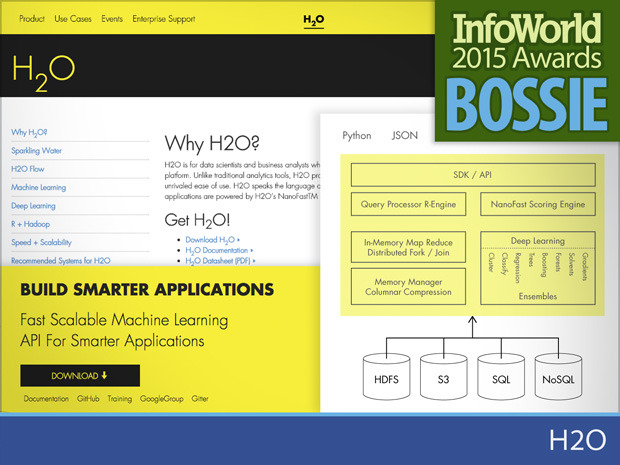


**H2O**

* [H2O](http://h2o.ai/product/)

is a distributed, in-memory processing engine for machine learning that boasts an impressive array of algorithms. Previously only available for R users, version 3.0 adds Python and Java language bindings, as well as a Spark execution engine for the back end. The best way to view H20 is as a very large memory extension of your R environment. Instead of working directly on large data sets, the R extensions communicate via a REST API with the H2O cluster, where H2O does the heavy lifting.

Several useful R packages such as ddply have been wrapped, allowing you to use them on data sets larger than the amount of RAM on the local machine. You can run H2O on EC2, on a Hadoop/YARN cluster, and on Docker containers. With Sparkling Water (Spark plus H2O) you can access Spark RDDs on the cluster side by side to, for example, process a data frame with Spark before passing it to an H2O machine learning algorithm.



**Apex**

[Apex](https://www.datatorrent.com/apex/) is an enterprise-grade, big data-in-motion platform that unifies stream processing as well as batch processing. A native YARN application, Apex processes streaming data in a scalable, fault-tolerant manner and provides all the common stream operators out of the box. One of the best things about Apex is that it natively supports the common event processing guarantees (exactly once, at least once, at most once). Formerly a commercial product by DataTorrent, Apex's roots show in the quality of the documentation, examples, code, and design. Devops and application development are cleanly separated, and user code generally doesn't have to be aware that it is running in a streaming cluster.

A related project, [Malhar](https://github.com/DataTorrent/Malhar" \t "_blank), offers more than 300 commonly used operators and application templates that implement common business logic. The Malhar libraries significantly reduce the time it takes to develop an Apex application, and there are connectors (operators) for storage, file systems, messaging systems, databases, and nearly anything else you might want to connect to from an application. The operators can all be extended or customized to meet individual business's requirements. All Malhar components are available under the Apache license.

**Druid**

[Druid](https://druid.io/), which moved to a commercially friendly Apache license in February of this year, is best described as a hybrid, “event streams meet OLAP” solution. Originally developed to analyze online events for ad markets, Druid allows users to do arbitrary and interactive exploration of time series data. Some of the key features include low-latency ingest of events, fast aggregations, and approximate and exact calculations.

At the heart of Druid is a custom data store that uses specialized nodes to handle each part of the problem. Real-time ingest is managed by real-time nodes (JVMs) that eventually flush data to historical nodes that are responsible for data that has aged. Broker nodes direct queries in a scatter-gather fashion to both real-time and historical nodes to give the user a complete picture of events. Benchmarked at a sustained 500K events per second and 1 million events per second peak, Druid is ideal as a real-time dashboard for ad-tech, network traffic, and other activity streams.

**Flink**

At its core, [Flink](https://flink.apache.org/" \t "_blank) is a data flow engine for event streams. Although superficially similar to Spark, Flink takes a different approach to in-memory processing. First, Flink was designed from the start as a stream processor. Batch is simply a special case of a stream with a beginning and an end, and Flink offers APIs for dealing with each case, the DataSet API (batch) and the DataStream API. Developers coming from the MapReduce world should feel right at home working with the DataSet API, and porting applications to Flink should be straightforward. In many ways Flink mirrors the simplicity and consistency that helped make Spark so popular. Like Spark, Flink is written in Scala.

The developers of Flink clearly thought out usage and operations too: Flink works natively with YARN and Tez, and it uses an off-heap memory management scheme to work around some of the JVM limitations. A peek at the Flink JIRA site shows a healthy pace of development, and you’ll find an active community on the mailing lists and on StackOverflow as well.

**Drill**

[Drill](https://drill.apache.org/) is a distributed system for interactive analysis of large-scale data sets, inspired by [Google's Dremel](http://research.google.com/pubs/pub36632.html). Designed for low-latency analysis of nested data, Drill has a stated design goal of scaling to 10,000 servers and querying petabytes of data and trillions of records.

Nested data can be obtained from a variety of data sources (such as HDFS, HBase, Amazon S3, and Azure Blobs) and in multiple formats (including JSON, Avro, and protocol buffers), and you don't need to specify a schema up front (“schema on read”).

Drill uses ANSI SQL:2003 for its query language, so there's no learning curve for data engineers to overcome, and it allows you to join data across multiple data sources (for example, joining a table in HBase with logs in HDFS). Finally, Drill offers ODBC and JDBC interfaces to connect your favorite BI tools.

**HBase**

[HBase](http://hbase.apache.org/) reached the 1.x milestone this year and continues to improve. Like other nonrelational distributed datastores, HBase excels at returning search results very quickly and for this reason is often used to back search engines, such as the ones at eBay, Bloomberg, and Yahoo. As a stable and mature software offering, HBase does not get fresh features as frequently as newer projects, but that's often good for enterprises.

Recent improvements include the addition of high-availability region servers, support for rolling upgrades, and YARN compatibility. Features in the works include scanner updates that promise to improve performance and the ability to use HBase as a persistent store for streaming applications like Storm and Spark. HBase can also be queried SQL style via the [Phoenix](http://phoenix.apache.org/) project, now out of incubation, whose SQL compatibility is steadily improving. Phoenix recently added a Spark connector and the ability to add custom user-defined functions.

**Hive**

Although stable and mature for several years, [Hive](https://hive.apache.org/) reached the 1.0 version milestone this year and continues to be the best solution when really heavy SQL lifting (many petabytes) is required. The community continues to focus on improving the speed, scale, and SQL compliance of Hive. Currently at version 1.2, significant improvements since its last Bossie include full ACID semantics, cross-data center replication, and a cost-based optimizer.

Hive 1.2 also brought improved SQL compliance, making it easier for organizations to use it to off-load ETL jobs from their existing data warehouses. In the pipeline are speed improvements with an in-memory cache called LLAP (which, from the looks of the JIRAs, is about ready for release), the integration of Spark machine learning libraries, and improved SQL constructs like nonequi joins, interval types, and subqueries.

**Kylin**

[Kylin](https://kylin.incubator.apache.org/) is an application developed at eBay for processing very large OLAP cubes via ANSI SQL, a task familiar to most data analysts. If you think about how many items are on sale now and in the past at eBay, and all the ways eBay might want to slice and dice data related to those items, you will begin to understand the types of queries Kylin was designed for.

Like most other analysis applications, Kylin supports multiple access methods, including JDBC, ODBC, and a REST API for programmatic access. Although Kylin is still in incubation at Apache, and the community nascent, the project is well documented and the developers are responsive and eager to understand customer use cases. Getting up and running with a starter cube was a snap. If you have a need for analysis of extremely large cubes, you should take a look at Kylin.

**CDAP**

[CDAP](http://cdap.io/) (Cask Data Access Platform) is a framework running on top of Hadoop that abstracts away the complexity of building and running big data applications. CDAP is organized around two core abstractions: data and applications. CDAP Datasets are logical representations of data that behave uniformly regardless of the underlying storage layer; CDAP Streams provide similar support for real-time data.

Applications use CDAP services for things such as distributed transactions and service discovery to shield developers from the low-level details of Hadoop. CDAP comes with a data ingestion framework and a few prebuilt applications and “packs” for common tasks like ETL and website analytics, along with support for testing, debugging, and security. Like most formerly commercial (closed source) projects, CDAP benefits from good documentation, tutorials, and examples.

**Ranger**

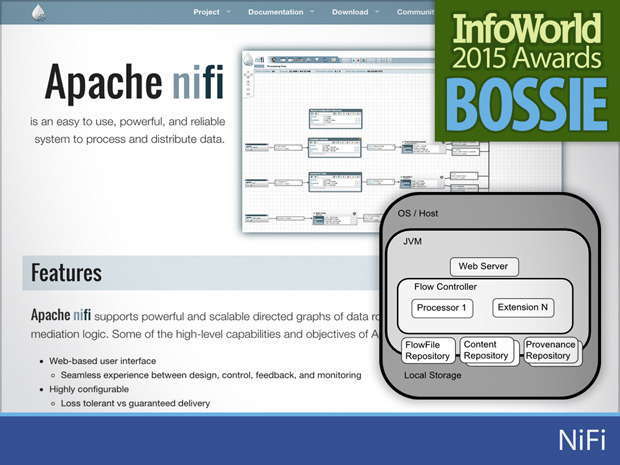
Security has long been a sore spot with Hadoop. It isn’t (as is frequently reported) that Hadoop is “insecure” or “has no security.” Rather, the truth was more that Hadoop had too much security, though not in a good way. I mean that every component had its own authentication and authorization implementation that wasn’t integrated with the rest of platform.

Hortonworks acquired XA/Secure in May, and [a few renames later](http://www.infoworld.com/article/2973381/application-development/apache-ranger-chuck-norris-hadoop-security.html) we have [Ranger](https://ranger.incubator.apache.org/). Ranger pulls many of the key components of Hadoop together under one security umbrella, allowing you to set a “policy” that ties your Hadoop security to your existing ACL-based Active Directory authentication and authorization. Ranger gives you one place to manage Hadoop access control, one place to audit, one place to manage the encryption, and a pretty Web page to do it from.

**Mesos**

[Mesos](http://mesos.apache.org/), developed at the [AMPLab](https://amplab.cs.berkeley.edu/" \t "_blank) at U.C. Berkeley that also brought us Spark, takes a different approach to managing cluster computing resources. The best way to describe Mesos is as a distributed microkernel for the data center. Mesos provides a minimal set of operating system mechanisms like inter-process communications, disk access, and memory to higher-level applications, called “frameworks” in Mesos-speak, that run in what is analogous to user space. Popular frameworks for Mesos include [Chronos](http://nerds.airbnb.com/introducing-chronos/" \t "_blank) and [Aurora](http://aurora.apache.org/) for building ETL pipelines and job scheduling, and a few big data processing applications including Hadoop, Storm, and Spark, which have been ported to run as Mesos frameworks.

Mesos applications (frameworks) negotiate for cluster resources using a two-level scheduling mechanism, so writing a Mesos application is unlikely to feel like a familiar experience to most developers. Although Mesos is a young project, momentum is growing, and with Spark being an exceptionally good fit for Mesos, we're likely to see more from Mesos in the coming years.

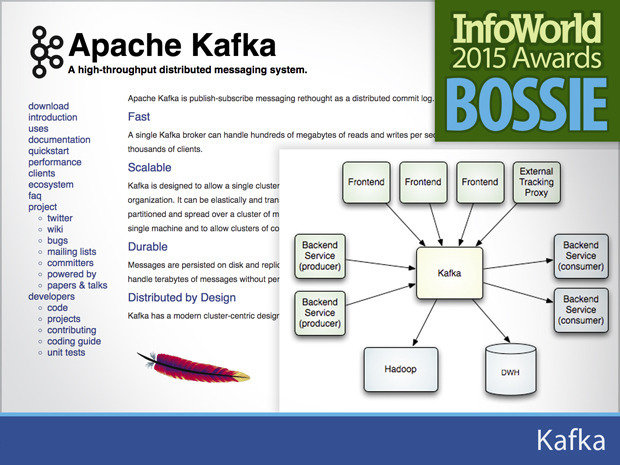


[See larger image](http://images.techhive.com/images/article/2015/09/bossies-2015-nifi-100613958-orig.jpg)

**NiFi**

[NiFi](http://nifi.apache.org/) is an incubating Apache project to automate the flow of data between systems. It doesn't operate in the traditional space that Kafka and Storm do, but rather in the space between external devices and the data center. NiFi was originally developed by the NSA and donated to the open source community in 2014. It has a strong community of developers and users within various government agencies.

NiFi isn't like anything else in the current big data ecosystem. It is much closer to a tradition EAI (enterprise application integration) tool than a data processing platform, although simple transformations are possible. One interesting feature is the ability to debug and change data flows in real time. Although not quite a REPL (read, eval, print loop), this kind of paradigm dramatically shortens the development cycle by not requiring a compile-deploy-test-debug workflow. Other interesting features include a strong “chain of custody,” where each piece of data can be tracked from beginning to end, along with any changes made along the way. You can also prioritize data flows so that time-sensitive information can be received as quickly as possible, bypassing less time-critical events.



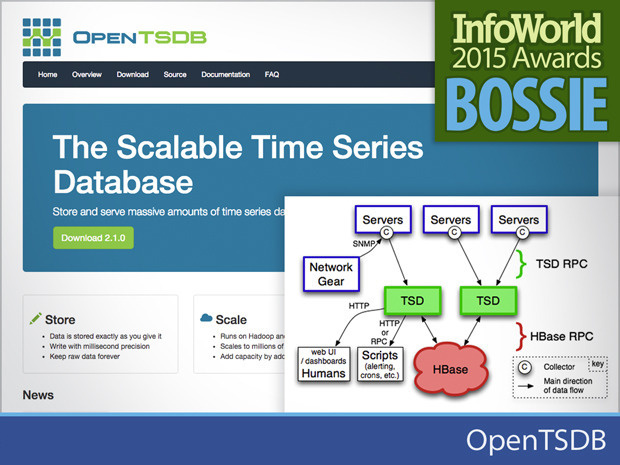
[See larger image](http://images.techhive.com/images/article/2015/09/bossies-2015-kafka-100613954-orig.jpg)

**Kafka**

[Kafka](https://kafka.apache.org/) has emerged as the de-facto standard for distributed publish-subscribe messaging in the big data space. Its design allows brokers to support thousands of clients at high rates of sustained message throughput, while maintaining durability through a distributed commit log.

High throughput comes from Kafka's intelligent partitioning of incoming data streams, which enables parallel reads and writes. These partitioned data streams are then copied to a configurable number of replicas, which in turn are written to disk, preventing data loss and enabling an ability to "replay" the history of the data stream.

When consumers want to read messages, Kafka looks up their offset in the central log and sends them. Because messages are not deleted immediately, adding consumers or replaying historical messages does not impose additional costs. Kafka has been benchmarked at 2 million writes per second by its developers at LinkedIn. Despite Kafka’s sub-1.0 version number, Kafka is a mature and stable product, in use in some of the largest clusters in the world.



Ads by [Kiosked](http://kiosked.com/" \t "_blank)

[See larger image](http://images.techhive.com/images/article/2015/09/bossies-2015-opentsdb-100613959-orig.jpg)

**OpenTSDB**

[OpenTSDB](http://opentsdb.net/) is a time series database built on HBase. It was designed specifically for analyzing data collected from applications, mobile devices, networking equipment, and other hardware devices. The custom HBase schema used to store the time series data has been designed for fast aggregations and minimal storage requirements.

By using HBase as the underlying storage layer, OpenTSDB gains the distributed and reliable characteristics of that system. Users don't interact with HBase directly; instead events are written to the system via the time series daemon (TSD), which can be scaled out as required to handle high-throughput situations. There are a number of prebuilt connectors to publish data to OpenTSDB, and clients to read data from Ruby, Python, and other languages. OpenTSDB isn't strong on creating interactive graphics, but several third-party tools fill that gap. If you are already using HBase and want a simple way to store event data, OpenTSDB might be just the thing.



[See larger image](http://images.techhive.com/images/article/2015/09/bossies-2015-zeppelin-100613963-orig.jpg)

**Zeppelin**

While still in incubation, [Apache Zeppelin](https://zeppelin.incubator.apache.org/) is nevertheless stirring the data analytics and visualization pot. The Web-based notebook enables users to ingest, discover, analyze, and visualize their data. The notebook also allows you to collaborate with others to make data-driven, interactive documents incorporating a growing number of programming languages.

This technology also boasts an integration with Spark and an interpreter concept allowing any language or data processing back end to be plugged into Zeppelin. Currently Zeppelin supports interpreters such as Scala, Python, SparkSQL, Hive, Markdown, and Shell.

Zeppelin is still immature. I wanted to put a demo up but couldn’t find an easy way to disable “shell” as an execution option (among other things). However, it already looks better visually than IPython Notebook, which is the popular incumbent in this space. If you don’t want to spring for DataBricks Cloud or need something open source and extensible, this is the most promising distributed computing notebook around -- especially if you’re a Sparky type

**Impala**

* Using impala, you can process data that is stored in HDFS at lightning-fast speed with traditional SQL knowledge.
* Since the data processing is carried where the data resides (on Hadoop cluster), data transformation and data movement is not required for data stored on Hadoop, while working with Impala.
* Using Impala, you can access the data that is stored in HDFS, HBase, and Amazon s3 without the knowledge of Java (MapReduce jobs). You can access them with a basic idea of SQL queries.
* To write queries in business tools, the data has to be gone through a complicated extract-transform-load (ETL) cycle. But, with Impala, this procedure is shortened. The time-consuming stages of loading & reorganizing is overcome with the new techniques such as ***exploratory data analysis & data discovery*** making the process faster.
* Impala is pioneering the use of the Parquet file format, a columnar storage layout that is optimized for large-scale queries typical in data warehouse scenarios.