[Storm](http://www.storm-project.net/) （<http://www.storm-project.net>）

Distributed and fault-tolerant realtime computation

# [Integrates](http://www.storm-project.net/about/integrates.html)

Storm integrates with any queueing system and any database system. Storm's spout abstraction makes it easy to integrate a new queuing system. Example queue integrations include:

1. [Kestrel](https://github.com/nathanmarz/storm-kestrel)
2. [RabbitMQ / AMQP](https://github.com/Xorlev/storm-amqp-spout)
3. [Kafka](https://github.com/nathanmarz/storm-contrib/tree/master/storm-kafka)
4. [JMS](https://github.com/ptgoetz/storm-jms)

Likewise, integrating Storm with database systems is easy. Simply open a connection to your database and read/write like you normally would. Storm will handle the parallelization, partitioning, and retrying on failures when necessary.

# [Simple API](http://www.storm-project.net/about/simple-api.html)

Storm has a simple and easy to use API. When programming on Storm, you manipulate and transform streams of tuples, and a tuple is a named list of values. Tuples can contain objects of any type; if you want to use a type Storm doesn't know about it's [very easy](https://github.com/nathanmarz/storm/wiki/Serialization) to register a serializer for that type.

There are just three abstractions in Storm: spouts, bolts, and topologies. A **spout** is a source of streams in a computation. Typically a spout reads from a queueing broker such as Kestrel, RabbitMQ, or Kafka, but a spout can also generate its own stream or read from somewhere like the Twitter streaming API. Spout implementations already exist for most queueing systems.

A **bolt** processes any number of input streams and produces any number of new output streams. Most of the logic of a computation goes into bolts, such as functions, filters, streaming joins, streaming aggregations, talking to databases, and so on.

A **topology** is a network of spouts and bolts, with each edge in the network representing a bolt subscribing to the output stream of some other spout or bolt. A topology is an arbitrarily complex multi-stage stream computation. Topologies run indefinitely when deployed.

Storm has a "local mode" where a Storm cluster is simulated in-process. This is useful for development and testing. The "storm" command line client is used when ready to submit a topology for execution on an actual cluster.

The [storm-starter](https://github.com/nathanmarz/storm-starter) project contains example topologies for learning the basics of Storm. Learn more about how to use Storm by reading the [tutorial](https://github.com/nathanmarz/storm/wiki/Tutorial) and the [documentation](https://github.com/nathanmarz/storm/wiki).

# [Scalable](http://www.storm-project.net/about/scalable.html)

Storm topologies are inherently parallel and run across a cluster of machines. Different parts of the topology can be scaled individually by tweaking their parallelism. The "rebalance" command of the "storm" command line client can adjust the parallelism of running topologies on the fly.

Storm's inherent parallelism means it can process very high throughputs of messages with very low latency. Storm was benchmarked at processing **one million 100 byte messages per second per node** on hardware with the following specs:

* **Processor:** 2x Intel E5645@2.4Ghz
* **Memory:** 24 GB

# [Fault tolerant](http://www.storm-project.net/about/fault-tolerant.html)

Storm is fault-tolerant: when workers die, Storm will automatically restart them. If a node dies, the worker will be restarted on another node.

The Storm daemons, Nimbus and the Supervisors, are designed to be stateless and fail-fast. So if they die, they will restart like nothing happened. This means you can *kill -9* the Storm daemons without affecting the health of the cluster or your topologies.

Read more about Storm's fault-tolerance [on the manual](https://github.com/nathanmarz/storm/wiki/Fault-tolerance).

# [Guarantees data processing](http://www.storm-project.net/about/guarantees-data-processing.html)

Storm guarantees every tuple will be fully processed. One of Storm's core mechanisms is the ability to track the lineage of a tuple as it makes its way through the topology in an extremely efficient way. Read more about how this works [here](https://github.com/nathanmarz/storm/wiki/Guaranteeing-message-processing).

Storm's basic abstractions provide an at-least-once processing guarantee, the same guarantee you get when using a queueing system. Messages are only replayed when there are failures.

Using [Trident](https://github.com/nathanmarz/storm/wiki/Trident-tutorial), a higher level abstraction over Storm's basic abstractions, you can achieve exactly-once processing semantics.

# [Use with any language](http://www.storm-project.net/about/multi-language.html)

Storm was designed from the ground up to be usable with any programming language. At the core of Storm is a [Thrift](http://thrift.apache.org/) [definition](https://github.com/nathanmarz/storm/blob/master/src/storm.thrift) for defining and submitting topologies. Since Thrift can be used in any language, topologies can be defined and submitted from any language.

Similarly, spouts and bolts can be defined in any language. Non-JVM spouts and bolts communicate to Storm over a [JSON-based protocol](https://github.com/nathanmarz/storm/wiki/Multilang-protocol) over stdin/stdout. Adapters that implement this protocol exist for [Ruby](https://github.com/nathanmarz/storm/blob/master/src/multilang/rb/storm.rb), [Python](https://github.com/nathanmarz/storm/blob/master/src/multilang/py/storm.py), [Javascript](https://github.com/Lazyshot/storm-node), [Perl](https://github.com/gphat/io-storm), and [PHP](https://github.com/lazyshot/storm-php).

*storm-starter* has an [example topology](https://github.com/nathanmarz/storm-starter/blob/master/src/jvm/storm/starter/WordCountTopology.java) that implements one of the bolts in Python.

# [Easy to deploy and operate](http://www.storm-project.net/about/deployment.html)

Storm clusters are easy to deploy, requiring a minimum of setup and configuration to get up and running. Storm's out of the box configurations are suitable for production. Read more about how to deploy a Storm cluster [here](https://github.com/nathanmarz/storm/wiki/Setting-up-a-Storm-cluster).

If you're on EC2, the [storm-deploy](https://github.com/nathanmarz/storm-deploy) project can provision, configure, and install a Storm cluster from scratch at just the click of a button.

Additionally, Storm is easy to operate once deployed. Storm has been designed to be [extremely robust](http://www.storm-project.net/about/fault-tolerant.html) – the cluster will just keep on running, month after month.

# [Free and open source](http://www.storm-project.net/about/free-and-open-source.html)

Storm is a free and open source project licensed under the [Eclipse Public License](http://www.eclipse.org/legal/epl-v10.html). The EPL is a very permissive license, allowing you to use Storm for either open source or proprietary purposes. Storm is hosted [on GitHub](http://github.com/nathanmarz/storm).

Storm has a large and growing ecosystem of libraries and tools to use in conjunction with Storm. The [storm-contrib](https://github.com/nathanmarz/storm-contrib) project is a central repository for community contributions. However, many community members also choose to host Storm-related projects outside of storm-contrib. Everything in storm-contrib is licensed under the Eclipse Public License. storm-contrib contains everything from:

1. *Spouts*: These spouts integrate with queueing systems such as JMS, Kafka, Redis pub/sub, and more.
2. *storm-state*: storm-state makes it easy to manage large amounts of in-memory state in your computations in a reliable by using a distributed filesystem for persistence
3. *Database integrations*: There are helper bolts for integrating with various databases, such as MongoDB, RDBMS's, Cassandra, and more.
4. Other miscellaneous utilities

The [Storm wiki](https://github.com/nathanmarz/storm/wiki) has links to other notable Storm-related projects hosted outside of storm-contrib.