Scrapy

**Components**

**Scrapy Engine**

The engine is responsible for controlling the data flow between all components of the system, and triggering events when certain actions occur. See the Data Flow section below for more details.

**Scheduler**

The Scheduler receives requests from the engine and enqueues them for feeding them later (also to the engine) when the engine requests them.

**Downloader**

The Downloader is responsible for fetching web pages and feeding them to the engine which, in turn, feeds them to the spiders.

**Spiders**

Spiders are custom classes written by Scrapy users to parse responses and extract items (aka scraped items) from them or additional URLs (requests) to follow. Each spider is able to handle a specific domain (or group of domains). For more information see *[Spiders](http://doc.scrapy.org/en/0.24/topics/spiders.html" \l "topics-spiders)*.

**Item Pipeline**

The Item Pipeline is responsible for processing the items once they have been extracted (or scraped) by the spiders. Typical tasks include cleansing, validation and persistence (like storing the item in a database). For more information see *[Item Pipeline](http://doc.scrapy.org/en/0.24/topics/item-pipeline.html" \l "topics-item-pipeline)*.

**Downloader middlewares**

Downloader middlewares are specific hooks that sit between the Engine and the Downloader and process requests when they pass from the Engine to the Downloader, and responses that pass from Downloader to the Engine. They provide a convenient mechanism for extending Scrapy functionality by plugging custom code. For more information see *[Downloader Middleware](http://doc.scrapy.org/en/0.24/topics/downloader-middleware.html" \l "topics-downloader-middleware)*.

**Spider middlewares**

Spider middlewares are specific hooks that sit between the Engine and the Spiders and are able to process spider input (responses) and output (items and requests). They provide a convenient mechanism for extending Scrapy functionality by plugging custom code. For more information see*[Spider Middleware](http://doc.scrapy.org/en/0.24/topics/spider-middleware.html" \l "topics-spider-middleware)*.

**Data flow**

The data flow in Scrapy is controlled by the execution engine, and goes like this:

1. The Engine opens a domain, locates the Spider that handles that domain, and asks the spider for the first URLs to crawl.
2. The Engine gets the first URLs to crawl from the Spider and schedules them in the Scheduler, as Requests.
3. The Engine asks the Scheduler for the next URLs to crawl.
4. The Scheduler returns the next URLs to crawl to the Engine and the Engine sends them to the Downloader, passing through the Downloader Middleware (request direction).
5. Once the page finishes downloading the Downloader generates a Response (with that page) and sends it to the Engine, passing through the Downloader Middleware (response direction).
6. The Engine receives the Response from the Downloader and sends it to the Spider for processing, passing through the Spider Middleware (input direction).
7. The Spider processes the Response and returns scraped Items and new Requests (to follow) to the Engine.
8. The Engine sends scraped Items (returned by the Spider) to the Item Pipeline and Requests (returned by spider) to the Scheduler
9. The process repeats (from step 2) until there are no more requests from the Scheduler, and the Engine closes the domain.

**Event-driven networking**

Scrapy is written with [Twisted](http://twistedmatrix.com/trac/), a popular event-driven networking framework for Python. Thus, it’s implemented using a non-blocking (aka asynchronous) code for concurrency.

MongoDB

Lo siguiente es una breve descripción de las características principales de MongoDB:

Consultas Ad hoc[editar]

MongoDB soporta la búsqueda por campos, consultas de rangos y expresiones regulares. Las consultas pueden devolver un campo específico del documento pero también puede ser una función JavaScript definida por el usuario.

Indexación[editar]

Cualquier campo en un documento de MongoDB puede ser indexado, al igual que es posible hacer índices secundarios. El concepto de índices en MongoDB es similar a los encontrados en base de datos relacionales.

Replicación[editar]

MongoDB soporta el tipo de replicación maestro-esclavo. El maestro puede ejecutar comandos de lectura y escritura. El esclavo puede copiar los datos del maestro y sólo se puede usar para lectura o para copia de seguridad, pero no se pueden realizar escrituras. El esclavo tiene la habilidad de poder elegir un nuevo maestro en caso de que se caiga el servicio con el maestro actual.

Balanceo de carga[editar]

MongoDB se puede escalar de forma horizontal usando el concepto de “shard”.10 El desarrollador elige una llave shard, la cual determina cómo serán distribuidos los datos en una colección. Los datos son divididos en rangos (basado en la llave shard) y distribuidos a través de múltiples shard. Un shard es un maestro con uno o más esclavos. MongoDB tiene la capacidad de ejecutarse en múltiple servidores, balanceando la carga y/o duplicando los datos para poder mantener el sistema funcionando en caso que exista un fallo de hardware. La configuración automática es fácil de implementar bajo MongoDB y se pueden agregar nuevas máquinas a MongoDB con el sistema de base de datos corriendo.

Almacenamiento de archivos[editar]

MongoDB puede ser utilizado con un sistema de archivos, tomando la ventaja de la capacidad que tiene MongoDB para el balanceo de carga y la replicación de datos utilizando múltiples servidores para el almacenamiento de archivos. Esta función (que es llamada GridFS11 ) está incluida en los drivers de MongoDB y disponible para los lenguajes de programación que soporta MongoDB. Esta base de datos expone funciones para la manipulación de archivos y contenido a los desarrolladores. En un sistema con múltiple servidores, los archivos pueden ser distribuidos y copiados entre los mismos varias veces y de una forma transparente, de esta forma se crea un sistema eficiente que maneja fallos y balanceo de carga.

Agregación[editar]

La función MapReduce puede ser utilizada para el procesamiento por lotes de datos y operaciones de agregación. Esta función permite que los usuarios puedan obtener el tipo de resultado que se obtiene cuando se utiliza el comando SQL “group-by”.

Ejecución de JavaScript del lado del servidor[editar]

MongoDB tiene la capacidad de realizar consultas utilizando JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas.

<http://es.wikipedia.org/wiki/MongoDB>

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

### Document Database

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.



The advantages of using documents are:

* Documents (i.e. objects) correspond to native data types in many programming languages.
* Embedded documents and arrays reduce need for expensive joins.
* Dynamic schema supports fluent polymorphism.

### Key Features

#### High Performance

MongoDB provides high performance data persistence. In particular,

* Support for embedded data models reduces I/O activity on database system.
* Indexes support faster queries and can include keys from embedded documents and arrays.

#### High Availability

To provide high availability, MongoDB’s replication facility, called replica sets, provide:

* automatic failover.
* data redundancy.

A [replica set](http://docs.mongodb.org/manual/core/replication-introduction/" \l "replication-introduction) is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.

#### Automatic Scaling

MongoDB provides horizontal scalability as part of its core functionality.

* Automatic [sharding](http://docs.mongodb.org/manual/core/sharding-introduction/" \l "sharding-introduction) distributes data across a cluster of machines.
* Replica sets can provide eventually-consistent reads for low-latency high throughput deployments.