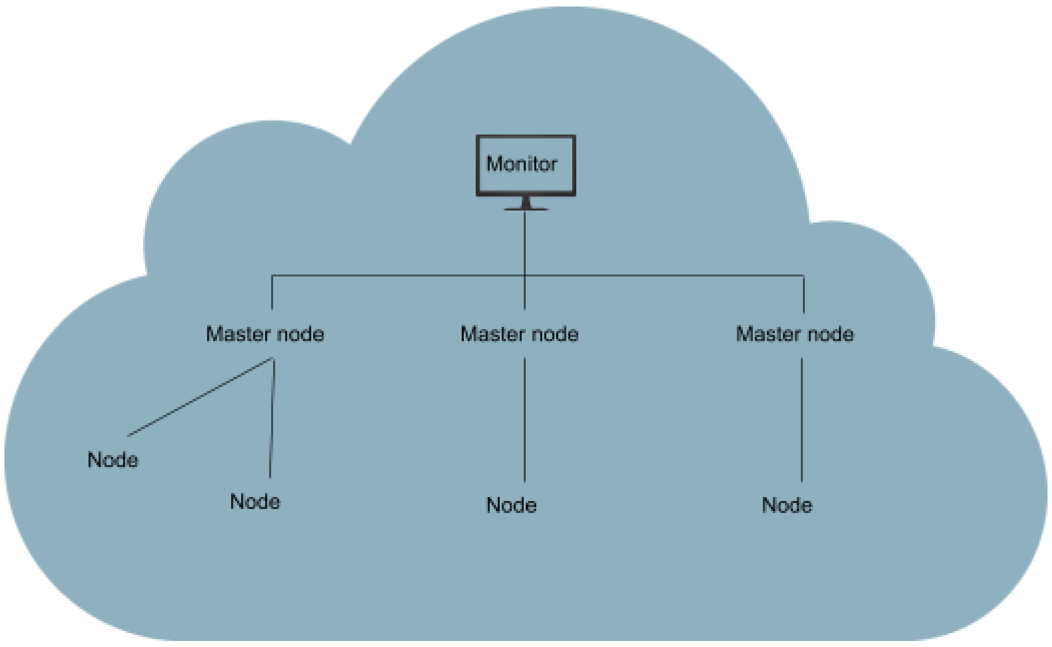
Architecture



The architecture used for the application is *Fog Computing*. The **monitor** is the centered node, connected to a display, that collects and shows the merged top words found across all the nodes.

To light the load of the **monitor**, the remaining nodes are divided into **master** and **edge** nodes. There can be multiple masters that transfer data to the **monitor** or higher masters.

The data is generated both in the **edge** nodes and **master** nodes. For this sample implementation, every data generated is assumed to be a single file with multiple words separated by spaces. The data transmitted instead consists of two files: **counter**, which contains the number of times each word has been found in the analyzed files, and **occurrences**, which contains the references to the documents where each word has been found.

For the data transmission between the nodes, **two** modes have been implemented: **active** and **passive**. Each **master** node and **monitor** can use different modes. With **passive** mode the nodes send the data passively to their master node every now and then, which in turn updates their counter and occurrences and sends them to their master or shows it if it’s the monitor. With **active** mode instead the monitor asks the nodes connected to send their data and waits for their responses.

Implementation

The application has been implemented on **Docker** and tested on the same machine, so there was a limitation regarding the ports and links of the containers. For this reason the containers have been connected through a **docker network**.

The code has been written in **Python**.
The monitor additionally uses a **Flask** application, interconnected with the master socket application, to show the result on display.
The ports used by the masters are X, X+10, X+20, X+30, with X given during **docker run** as an *environment variable*. Additionally the monitor uses the port 8080 for the web service application. Masters ip and first port are passed to the nodes **docker run** as *environment variables.*

Socket communication

The master and node applications communicate through socket TCP connections.
Every master container opens a socket and listens for children connections, which could be of two types: *request* or *register*.

- The *register* request is done when a new node needs to register to the master. The ID and transmission mode is communicated to the node.
- The other one, *request*, asks the master for new commands, which could be:
  - Send data: the children need to send their data
  - Send file: a file was requested as download and the child which owns it needs to send it
  - Change mode: notifies the children that the master changes its transmission mode(active or passive)

For the **active** communication the monitor, when the data is requested, tells the children to send their data through a *request* and waits for their responses.
For the **passive** communication the monitor shows the data that has already been received passively by the nodes.
The transmission mode can be set to the masters as an *environment variable*.