Server Detector Service

This service designed to check host server's types. It is initially designed to detect only Nginx servers and extended to check other types of servers via different end point with the same way.

Service Endpoints and Usage:

Endpoint: /detectnginxserver/

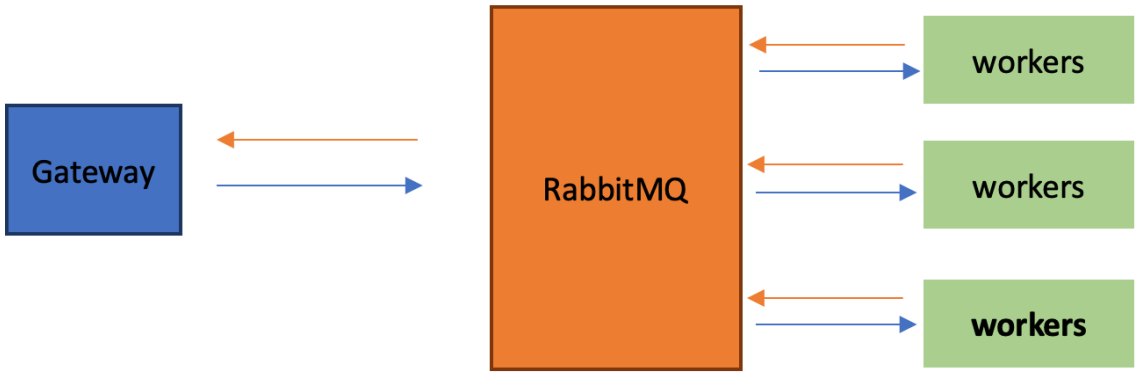
Detects Nginx servers within list of host names and finds their IP address
Input: Json file that has 'hosts' attribute with list of host names
Example Input: "hosts":["demo.nginx.com", "www.microsoft.com", "example.com", "cbr.com", "blog.detectify.com", "demo.nginx.com"]}
Output: Json file with host names and IP addresses that are Nginx servers
{"demo.nginx.com": ["206.251.255.64"], "cbr.com": ["34.201.177.150"], "blog.detectify.com": ["104.196.191.243"]}

Endpoint: /detectserver/

Detects queried servers within list of host names and finds their IP address.
Input: Json file that has 'server_type' and 'hosts' attributes with list of host names
Example Input: {"server_type":"nginx", "hosts":["demo.nginx.com", "www.microsoft.com"]}
Output: Json file with host names and IP addresses that are Nginx servers
{"demo.nginx.com": ["206.251.255.64"]}

Service Architecture and Scalability

Service is built on microservice architecture. Service consists of three independent elements, gateway, message broker (RabbitMQ) and worker.



Service elements:

Gateway:

Catches incoming requests and divide it into smaller jobs and sends to the job queue where workers consume jobs. After sending the jobs, it starts to collect the responses of the finished jobs from a unique queue for the request, combines and returns the response.

Workers:

Workers connect to job queue, processes and sends their responses to unique queue that is created for the request.

RabbitMQ:

Distributes message between gateway and workers. Jobs have permanent queue called 'hosts.to.detect'. On the other hand, results are distributed on temporary channel created uniquely fir each request.

Scalability

Due to the architecture system is scalable. Since each element is independent number of workers and gateways can be increased as necessary.

Improvement Areas:

- In current implementation Gateway collects messages synchronously (due to the given task), asynchronous implementation with server push could increase performance and reliability.

- Queue manager is needed for more reliable service. In current architecture each request has unique queue. A failure in Gateway may result messages with no consumers. This situation makes workers to hang due to not acknowledge messages.

- Cache mechanism such as Redis could be better for workers to share results. It may increase performance and also reliability. And also queue manager becomes unnecessary.

- Throttling and rate must be added if it will used as external service.

- More unit test coverage

How to setup

- Unzip the folder
- Go to the folder from the terminal
- Execute 'docker-compose up'