SE−227

Hands-on #4

January 2, 2020

Question 1

What's relationship between *Apache Zookeeper* and *paxos protocol*?

According to the architecture graph provided, the zookeeper aims at simplifying the hostname resolving processes. Zookeeper plays its role by registering lots of zookeeper watchers in each individual containers. Every zookeeper watcher will update the "/etc/hosts" file in the docker container, which interferes the host name resolving stuff.

But we may notice that each docker container has its own zookeeper watcher. They are all controlled by the *Zookeeper Cluster*. It is necessary to maintain the consensus and correctness among all nodes. That will require *Paxos Protocol* to ensure that.

So in conclusion, parts of the implementation of *Apache Zookeeper* relies on the thoughts of *paxos protocol*.

Question 2

Many databases or file systems also provide distributed replication support. However, many real-world applications choose Zookeeper for configurations management. Why? What feature of Zookeeper makes it an excellent choice for configuration management?

I suppose Zookeeper has two major advantages storing the configurations.

First, it could safely store configurations in distributed nodes, which means the possibility of a clean crash would be quite low. The crash of single node won't affect anything.

Plus, it is based on **consensus** but not **consistency**. That means each node could store different versions of configurations, but they will finally come out with a consensus decision. That quite meets our need in configuration storage. We could tolerate a consensus configuration that's not the latest, but we would never accept nodes using various configuration versions, which could cause conflicts and serious faults.

Question 3

Please implement the naming service by yourself. You need to submit your source codes and all extended docker files in the final answer package and briefly describe your implementation in the document.

Well... Since programming with Java or C forces me to consider things unrelated to the main idea itself, I will use Python with *kazoo* to manipulate the Zookeeper API.

See the "server.py" for more details.

In that python file, we first registers the client and opens the connection (Line 9 to 19).

Then, we will firstly connects to the root cluster node (named '/'), which by default is the root node (Line 21 to 25).

After that, we should use a stack to recursively get all children nodes, and get their semantic path-based names (Line 25 to 39).

Finally, we will use a dictionary to store and allocate hosts for those nodes. Then, we dump that into JSON string and store them in the Zookeeper nodes (Line 41 to 63).

And the client will runs a "client.py" to update the hosts from the ZooKeeper cluster.

It requires an extra system argument to detect its node name (Line 9 to 14).

Then, it will continuously connects to the cluster and gets the latest binary data from the server (Line 16 to 32).

Finally, we can decode the binary stored JSON data and updates the hosts file in each docker container (Line 33 to 46).

After that, we should be able to easily package them into the Dockerfile.

Question 4

Please partition the database according to the previous description and briefly introduce your solution.

I don't see there's any problem according to the implementation in my Hands-on #3 submission. In the "docker-compose.yml" configuration, three differently named services are actually based on the same MongoDB images. Thus, these services won't have to share the same container, the bottleneck should be eased a lot.

And, it would be more robust and gain more fault-tolerance ability when one MongoDB cluster fails, under which circumstance the system won't be totally unavailable and maintain the availability at a basic level.

Usage Description

Some simple descriptions.

Since the Python scripts will stop working every time it was called, we must regularly call them to ensure its in-time update. So there are two bash scripts provided for that.

Files are located at "./handson4/scheduler_server.sh" and "./handson4/scheduler_client.sh".

In order to use them, you should call them with two parameters, the interval time between to every attempt at updating the HOSTS file, and the node name for each one.

Notice that you need to authenticate this script to write to the HOSTS file. Windows environment could use the PowerShell script instead.

In order to solidify that daemon scripts into the Docker image, we can use this script to initialize the docker image:

> docker run -i -t <Image's name> /bin/bash -c "./scheduler_client.sh <Interval Time> <Node's Name> & > client.log"
>
> docker run -i -t <Image's name> /bin/bash -c "./scheduler_server.sh <Interval Time> <Node's Name> & > server.log"

And that will automatically run the daemon process in the background, and prints the log in respective files.