**TheFundsChain: scaling with Hyperledger**

**Availability & Scalability**. DLT Technology: Hyperledger fabric. This note provides some insight for a scalable and highly available DLT platform for TheFundsChain.

High performance and high peak throughput are a strong Hyperledger selling point. Let us examine more in details how our use case may leverage scalability features coming with the Hyperledger environment.

**Getting logical high-availability: redundant peers**

There is no notion of “a logical peer” with Hyperledger. If 2 nodes perform similarly, they constitute two different peers, with different identities. In our model, if for instance a FUND chaincode is maintained by a ISR peer (the issuer), there is one single such node with a given identity.

How about providing several peers, operated by the same organization, with the same role? Let’s for instance assume that Natixis AM, issuer of a FUND chaincode operates 2 ISR nodes as endorsing peers. CACEIS would similarly operate 2 TAG endorsing peers for this FUND.

Such a redundant setting is welcome with a proper endorsing policy (not all signatures are required, only a subset). If a node becomes temporarily unavailable, the other node continues to maintain the chaincode. Such capabilities must be integrated in the preliminary design: endorsement and role checking must integrate the possibility of several nodes with the same role.

To illustrate this point, one could consider, with the example above, CACEIS running 10 peers with SMP “caceis” and Natixis running 2 peers with SMP “natixisAM”. A policy of the form AND(‘caceis.member’,’natixisAM’) would require at least one signature from each group among these 12 peers. An intelligent setup would be for the transmitting peer to load balance invoke messages across these peers.

With such a setup, role checking is rather coarse, as only membership to an organization may be checked. Finer grained checks (our *functional* roles) will have to be checked by the chaincode VSCC service itself.



We must assume – and this should be tested more thoroughly – that once the node comes up again, it synchronizes well with other peers to get the latest state of the chaincode. In such a test, what is interesting is the gossip overhead between peers needed to synchronize.

**Physical high availability: fail-over docker containers**

Another possibility to obtain a redundant configuration is to setup a fail-over cluster of docker containers. Only one node works at a time, the other one being kept on standby. This uses a docker feature known as “swarm” (requires docker engine 1.12.0). With a couchdb configuration, we have 2 containers to maintain in fail-over: the chaincode container and the corresponding couchdb container. Docker swarm is able to “pool” available physical servers to run the required services. For instance, if we run 2 containers for a single chaincode and operate, say, 1000 different chaincodes, we may pool for instance 10 physical servers to run this load collectively. Whenever a node fails, the containers are restarted on one of the remaining nodes.



Warning: docker swarms cannot be used for load-balancing.

Note: properly failing-over a database node requires not only to switch the IP address of the service, but also the attached storage.



Warning: couchdb replication cannot *a priori* be relied upon as the state of the replica may be lagging behind whenever a failure occurs. However, the Hyperledger synchronization gossip could restore the latest state of the chaincode from other peers (to be tested). We would prefer that the use of fail-over does not rely on such mechanism (necessarily much slower).

Sharing storage between couchdb nodes may be achieved by different means: (i) use storage vendor-specific docker plugins to operate shared volumes[[1]](#footnote-1); (ii) deploy couchdb on a distributed filesystem[[2]](#footnote-2).

Note that the default setting for H.F is to require couchdb peer to run locally (on the peer node). This constraint could be relaxed by would obviously affect performances.

The nice thing about using docker to obtain high-availability is that is doesn’t impact our functional design. On the other hand, set up quickly gets hairy and very technical.

**Scaling option (i): sharding**

Our first weapon to scale up is granular sharding: each FUND may be individually deployed on a peer. Therefore, for a large organization, say CACEIS or BPSS, to be able to run 10 000 funds or more, an arbitrary number of physical hosts may be deployed to run the containers, say 10 or more.

However, some chaincodes may not be distributed so easily. For instance, a RegFundDirectory may not be sharded this way. Obviously, we may design the protocol so as not to generate too much traffic on this. What are our options if this is not possible? For instance, we might want the RegFundDirectory to be heavily subject to queries validating agreement seals. Redundant peers (see above) may be an answer. A reverse proxy might efficiently balance query load (but clearly, not so much for changes).

Warning: default setup for Hyperledger is to manage a single instance couchdb. Having instead a cluster may prove tricky to configure. The single and definitive requirement from Hyperledger is that only the chaincode peer process writes to the database.

**Scaling option (ii): couchdb sharding**

Within a single chaincode, we may shard the state database on a couchdb cluster. For instance, the large set of investors holdings that constitutes the state of a Fund chaincode may be sharded across several (remote) couchdb nodes (H.F advises not to do so…).

For that, a couchdb cluster must be configured[[3]](#footnote-3), with a proper reverse proxy such as HAProxy, then “containerized” as a docker deployable component.

Warning: the concept of “eventual consistency” used by couchdb prevents us from using a couchdb cluster for high-availability or load-balancing purpose. Clustering may only be used to partition large datasets.

Warning: a point to be settled by experimentation is if we may share a single couchdb between different chaincodes (each one maintaining its own set of documents). If this is indeed possible, it is not clear as per current documentation how chaincodes are insulated from each other[[4]](#footnote-4).

Note: H.F design document ([..\Platform\HyperledgerFabric\_LedgerV1\_20170315.pptx](../Platform/HyperledgerFabric_LedgerV1_20170315.pptx)) states that this should not be possible.

Warning: remember that couchdb is deployed as an independent container.

**Scaling option (iii): Hyperledger channels**

Hyperledger proposes a “channel” feature to deploy chaincodes among separate population of peers. A channel is therefore a communication mechanism which allows peers and clients to operate. Channels being completely independent, an infrastructure may be safely partitioned into channels. A client may access to several channels.

The granularity we suggest for such channels is the jurisdiction. A channel for France, a channel for Luxembourg for instance.

**Sidekick business processes: analytics & reporting (aka sideDB)**

Chaincode query interface is not very well suited for mass-reporting. If one chaincode represents one fund, how do we easily produce report on many funds?

This is where CouchDB may be convenient, with its replication mechanism and “eventual consistency” feature.

Every chaincode has its own CouchDB instance. Now let’s assume that we declare another instance, with unidirectional replication from all “Fund” chaincode instances. Documents get replicated on this instance on which new, analytic views may be declared by a peer.



1. <https://docs.docker.com/engine/extend/legacy_plugins/#volume-plugins> [↑](#footnote-ref-1)
2. <https://thenewstack.io/methods-dealing-container-storage/> [↑](#footnote-ref-2)
3. <http://docs.couchdb.org/en/2.0.0/cluster/setup.html> [↑](#footnote-ref-3)
4. In this design document, one has to believe that to one chaincode peer corresponds a single couchdb instance. <https://www.google.fr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&ved=0ahUKEwjfg7DnscjUAhXCVBQKHV5hBwQQFgg_MAM&url=https%3A%2F%2Fjira.hyperledger.org%2Fsecure%2Fattachment%2F10585%2FHyperledgerFabric_LedgerV1_20170315.pptx&usg=AFQjCNHdiHMRBVtPcNQoSJuX-CFug8hRBQ> [↑](#footnote-ref-4)