**Docker Swarm - Raft Consensus**

### What is Raft?

Raft is a consensus algorithm that is designed to be easy to understand. It’s equivalent to Paxos in fault-tolerance and performance. The difference is that it’s decomposed into relatively independent sub problems, and it cleanly addresses all major pieces needed for practical systems. We hope Raft will make consensus available to a wider audience, and that this wider audience will be able to develop a variety of higher quality consensus-based systems than are available today.

what is consensus?

Consensus is a fundamental problem in fault-tolerant distributed systems. Consensus involves multiple servers agreeing on values. Once they reach a decision on a value, that decision is final. Typical consensus algorithms make progress when any majority of their servers is available; for example, a cluster of 5 servers can continue to operate even if 2 servers fail. If more servers fail, they stop making progress (but will never return an incorrect result).

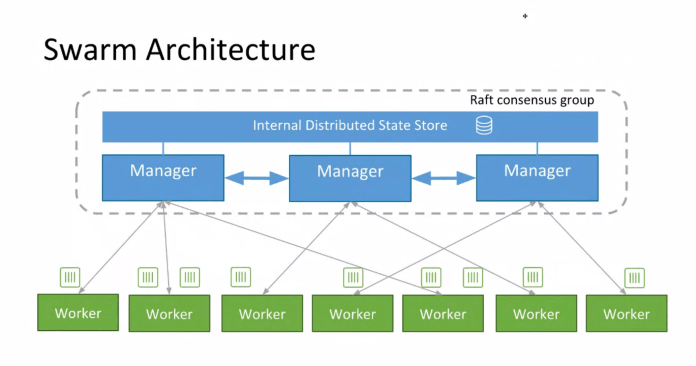
Consensus typically arises in the context of replicated state machines, a general approach to building fault-tolerant systems. Each server has a state machine and a log. The state machine is the component that we want to make fault-tolerant, such as a hash table. It will appear to clients that they are interacting with a single, reliable state machine, even if a minority of the servers in the cluster fail. Each state machine takes as input commands from its log. In our hash table example, the log would include commands like *set x to 3*. A consensus algorithm is used to agree on the commands in the servers’ logs. The consensus algorithm must ensure that if any state machine applies *set x to 3* as the *n*th command, no other state machine will ever apply a different *n*th command. As a result, each state machine processes the same series of commands and thus produces the same series of results and arrives at the same series of states.

let’s add an even better illustration that dynamically shows Raft in action for 2 processes:

* Leader election
* Log replication

using a [Raft](https://raft.github.io/raft.pdf) implementation, the managers maintain a consistent internal state of the entire swarm and all the services running on it

The overall view of a swarm is presented in the picture below.



Managers nodes handle the cluster states while worker nodes are the one that execute the workload. By default, a manager is also a worker.

Among the managers, the leader node is the one that logs all the actions that are done in the cluster (node added/removed, creation of a service, …). Swarm then ensures that the leader’s logs are replicated within each manager so one of them can take the leader role in case the current one becomes unavailable.

Each manager has the same version of the logs, and on each manager the logs are encrypted. We will use a swarm-rafttool from [SwarmKit](https://github.com/docker/swarmkit) in order to decrypt and make them human readable.

Why are Raft logs encrypted in Swarm?

The secret management, introduced in Docker 1.13, enables to securely provide sensitive information to containers running on a Swarm. Basically, an operator creates a secret (usually containing credentials, certificates, and other private information) and then provides this secret to a service. The secret is saved in the Raft logs and then accessible in a temporary filesystem (/run/secrets/SECRET\_NAME) by each container of the service. As the secret is in clear in the Raft logs, having the logs encrypted prevents the attacker from accessing the secret if a manager is compromised.

Alongside the encrypted logs are the public/private keys used for the encryption. This private key (/var/lib/docker/swarm/certificates/swarm-node.key) is used to encrypt the Raft logs and to ensure the secure TLS communication between the nodes.

Lock a Swarm for even more security

In the case a manager is compromised, if the logs (and the encryption keys next to them) are disclosed, it’s easy for an attacker to decrypt the logs and get access to sensitive information. In order to prevent this from happening, the swarm can be locked. Doing so, an unlock key is generated and used to encrypt the public/private keys. This unlock key must be saved offline and provided manually when the docker daemon restart (and also to decrypt the logs as we will see later on).

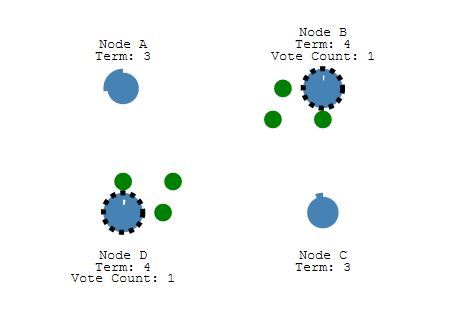
How does swarm-rafttool decrypt the logs?

As we said above, each manager has the swarm’s encrypted Raft logs and the keys used to encrypt/decrypt them. *swarm-rafttool* uses one of those key to decrypt the logs. If the Swarm is encrypted, the logs can still be decrypted providing the unlock-key to the tool.

In the following, we will setup a Swarm and inspect the logs while performing some operations (add a second manager node, create a service, create a secret).

**Raft Visualization**

This one make me understand what **Raft** is and how it works! So nice! kudos creator even me can understand this so this shouldn’t be problem for you.



**Raft Interactive Visualization**

Here’s a Raft cluster running in your browser. You can interact with it to see Raft in action. Five servers are shown on the left, and their logs are shown on the right. We hope to create a screencast soon to explain what’s going on. This visualization ([RaftScope](https://github.com/ongardie/raftscope)) is still pretty rough around the edges;