Availability and Scaling

The base version of the WhatsApp Business API runs on a single Docker container. High Availability and Multiconnect work together to give you more options.

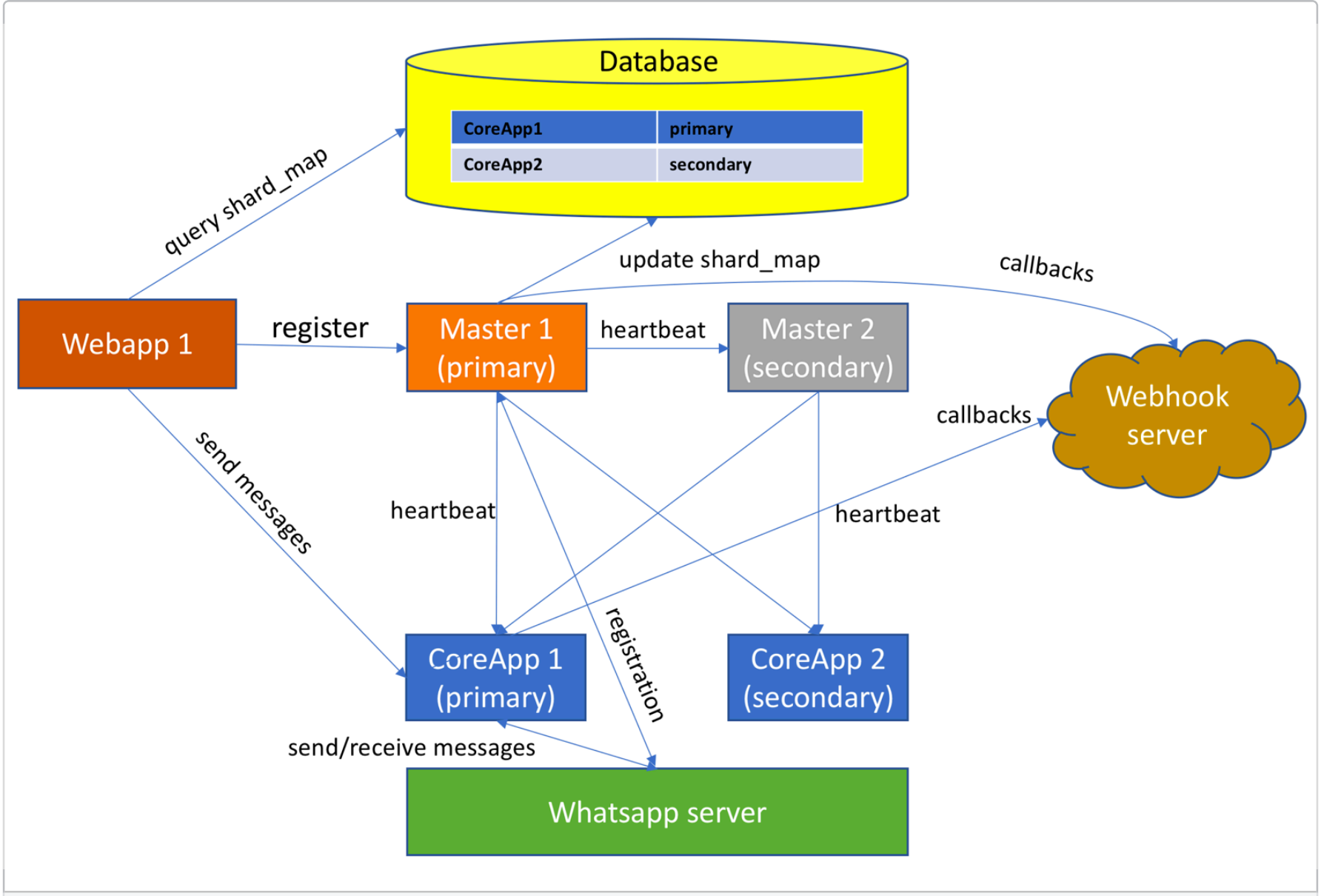
This document covers:

- High Availability Introduction
- Multiconnect Introduction

> For more information on setting up and running these solutions, see the High Availability and Multiconnect documentation.

High Availability Introduction

A high availability cluster requires at least two *Master* nodes and two *Coreapp* nodes as seen in the following diagram:



High Availability Cluster

All of nodes are recommended to run on different machines/racks to avoid single machine/rack failure affecting multiple nodes at the same time.

Starting up

When a cluster starts up, all Master nodes will compete to grab the master lease to become primary. Only one node will succeed and others will become secondary Masters. If there are *N* number of Master nodes in the cluster, there will be one primary Master and *N-1* secondary Masters. The primary Master is responsible for registration, database schema upgrade, configuration changes broadcast, reporting database stats, cluster management, etc. If the primary Master dies and loses the master lease, other secondary m=Masters will compete to take over the primary Master position.

When a Master becomes primary, it will first load the shard map table from the database to learn who is the current primary Coreapp. If there is no primary Coreapp in the cluster, the primary Master will promote one healthy secondary Coreapp to primary Coreapp and update the shard map table in the database so that the Webapp can look up which Coreapp node to send API requests to. In this way, even if all Masters are down, it could still serve API requests in the Coreapp nodes to

achieve High Availability.

When a Coreapp node starts up, it will run as a secondary Coreapp until the primary Master promotes it to be primary Coreapp to connect to the WhatsApp server. After that, it's responsible for handling API requests.

Database-based monitoring

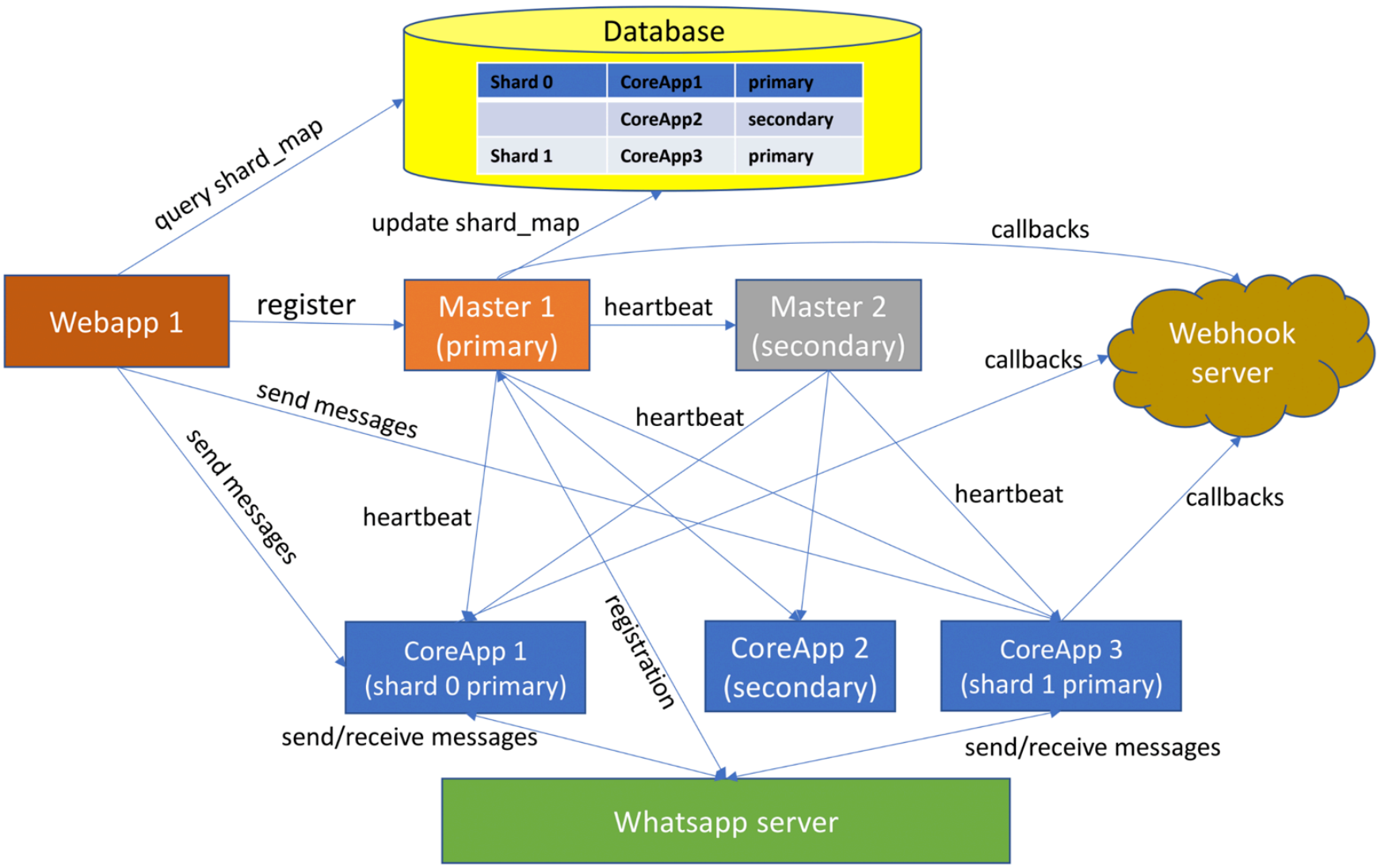
Each Coreapp node will update the database every minute to claim its liveness. The primary Master will check the database periodically to detect unhealthy Coreapp nodes. If a primary Coreapp node hasn't updated the database for more than 2 minutes, the primary Master will consider it unhealthy and promote other Coreapp nodes to primary. In this way, downtime is of about 2 minutes.

Heartbeat-based monitoring

If a cluster has more than one running Master, heartbeat-based monitoring detects node failures faster than database-based monitoring. In heartbeat-based monitoring, all Masters are responsible for monitoring Coreapp nodes by sending heartbeats to them every 5 seconds (configured by `heartbeat_interval`). If a primary Coreapp hasn't responded to the primary Master and one secondary Master for 30 seconds (configured by `unhealthy_interval`), it is considered unhealthy and the primary Master will promote a healthy secondary Coreapp to primary Coreapp. In this way, downtime is about 30 seconds, by default. You may decrease the `unhealthy_interval` value if a lower downtime is preferred. Check the Settings documentation for example payloads.

Multiconnect Introduction

With high availability, only one Docker container is responsible for sending and receiving messages from WhatsApp servers. If messaging traffic exceeds the maximum throughput of a single Docker container, there will be backlog of message sends and message delivery latency will increase. To scale out the WhatsApp Business API Client, multiconnect supports sharding to spread loads across multiple Docker containers. Currently, we only support static sharding with a shard number of 1, 2, 4, 8, 16, or 32. High availability is a special case of multiconnect where the shard number is 1.



A 2 shard multiconnect cluster

In the cluster, there are two Coreapp nodes (`CoreApp 1` and `CoreApp 3`) responsible for sending and receiving messages from WhatsApp server at the same time. Every message will only belong to one shard based on recipient ID.

Sharding

The WhatsApp Business API Client uses sharding to achieve multiconnect. Depending on the number of shards you set up, the database will store a shard map that determines which shard a message should go to depending on the recipient ID (or WhatsApp username). The function to determine this is:

```
shard_id = hash(recipient-id) % shard-number
```

Each shard is mapped to a running Docker container (Coreapp). The Webapp will know which Docker container to send the message request to based on the return of this function. It is recommended you set up shard number + X machines to be able to tolerate X machine failures.

In the 2 shard multiconnect diagram above, messages are routed to `CoreApp 1` and `CoreApp 3` based on the sharding function. `CoreApp 2` is secondary — it is warm, but has no active connection to the WhatsApp servers. Assume `CoreApp 1` gets messages for `shard=0` and `CoreApp 3` gets messages for `shard=1`. If `CoreApp 1` dies, only messages for `shard=0` will be affected. The system would still send and receive messages belonging to `shard=1` using `CoreApp 3`. Similar to high availability, `Master 1` will detect the failure of `CoreApp 1` and failover `shard=0` traffic to `CoreApp 2`. This failover will take approximately 35 seconds.

WhatsApp Business API

Overview

Getting Started

Guides

API Reference

Webhooks

Availability and Scaling

High Availability

Multiconnect

Message Templates

Guidelines

Troubleshooting

Changelog

FAQ