CGS backend – Replication Architecture

|  |  |
| --- | --- |
| Document Owner: | Yoni Zohar |
| Document File: | CGS\_replication\_architecture.docx |
| Document Version: | 1.0 |
| Document Location: | [link](http://turtle.timetoknow.local:8080/svn/dev/t2kdev/trunk/cgs/doc/design/architecture) |

This document describes the replication architecture of the CGS.

# Settings

We have 3 servers:

1. Primary, which includes:
2. TOMCAT
3. MONGODB
4. CMS (File System)
5. Secondary, which includes:
6. TOMCAT
7. MONGODB
8. CMS (File System)
9. Arbiter, which includes only MONGODB.

Ideally, we would just want two servers: primary and secondary. However, MongoDB's replication mechanism requires at least three servers. The additional server (Arbiter) is very light-weight.

# DNS

The DNS server should point to both primary and secondary, using the following configuration:

For each request, if the primary's Tomcat is up, direct the request to it. Otherwise, direct the request to the secondary.

In case the primary goes down, the secondary is replacing it just temporarily. When the primary comes back up (or is replaced by a new machine), the secondary stops replacing the primary.

# Tomcat

Both primary and secondary have an (identical) instance of the application server.

Usually, all requests are handled by the primary's Tomcat server.

There is nothing to replicate between the Tomcat servers, as they are just application servers.

# MongoDB

We use a replication mechanism of MongoDB called "replica-set". Ideally, we would just want two servers: primary and secondary. However, replica sets must have at least 3 members. The additional server doesn't contain any data. It is an empty MongoDB instance that is required in Mongo's management of the replica set.

## Replica Sets

A replica set is a set of MongoDB instances. One of the members is called "primary" and the rest are "secondaries".

The terms "primary" and "secondary" are used in two different meaning: Once for the distinction between the different **machines** and once for the distinction between their roles in the replica set.

In our replica set we only have 2 "full" members: one primary and one secondary. The replica set mechanism requires at least 3 members, and hence we've added a third member, called "arbiter". The arbiter doesn't contain any data. It is an empty MongoDB instance.

Most of the time, the primary of the replica set is the one found on the primary machine. However, if the primary machine goes down, the MongoDB instance of the secondary machine becomes the replica set primary (until the old primary is back up).

In our configuration, writes are only done by the primary of the replica set, and whenever the primary is up, reads are also performed by it.

## Tomcat-Mongo Configuration

The Tomcat server is configured to access the replica set, and not a specific server in it. This means that potentially, the Tomcat of the primary machine can perform a read request from the MongoDB server of the secondary machine. However, we've prioritize both Tomcats to always read from the primary MongoDB. Usually, the primary MongoDB is the MongoDB which is in the primary machine.

## Automatic Failover

The replica-set mechanism provides automatic failover:

In case the primary goes down, all writes and reads are directed to the secondary, automatically. When a new master is introduced, it replicates all the data from the secondary, and when it is ready, it becomes the primary.

# CMS

Our CMS is simply a specified directory in the file system of the server.

Both the primary and the secondary have designated CMS-directories (with the same full directory path).

The directories are synced with each other (bidirectionally), using a tool called Unison.

## Unison

Unison is a bidirectional synchronization tool.

When it is called, it checks for the differences between the two specified directories and updates each file and subdirectory to the most current version.

We configured Unison not to delete anything. This means that if a file is deleted in one machine, it won't be deleted in the other. This configuration is based on the assumption that assets are never deleted in the CMS.

## Cron

The Unison command is called every minute, using a Linux cron job.

If the previous Unison command is still processing, the current one is ignored, and the next sync will occur after one minute.

## Failover

If the primary machine falls, we need to create a new one.

In the meantime, the DNS will direct all Tomcat requests to the secondary machine. All new assets will be saved in the file system of the secondary machine.

After the new primary is created, we need to copy the CMS directory from the secondary to the primary (using e.g. scp). When we've finished copying, we may bring the Tomcat back up. Now, the DNS will direct all Tomcat requests to the primary.

If new files were added to the CMS during the copying, Unison will sync them to the primary (since it is a **bidirectional** syncing mechanism).