DSD Implementation: Highly Available, Fault Tolerant, Containerized Application leveraging Couchbase, AWS and Docker

Rohit Rohit
r_rohit@live.concordia.ca
Student ID: 40268994

Darlene Nazareth
d_naza@live.concordia.ca
Student ID: 40267132

Sherwyn Dsouza
sh_dso@live.concordia.ca
Student ID: 40266995

Bhavya Bugude
b_bugude@live.concordia.ca
Student ID: 40270772

Raghav Manchanda
r_mancha@live.concordia.ca
Student ID: 40276920

December 18, 2023

1 Abstract

This project explores the implementation of key distributed system design (DSD) concepts, including replication, fault tolerance, containerization, and load balancing. Utilizing technologies like Couchbase for database management, Docker for containerization, Nginx for web serving, reverse proxying, load balancing, and AWS services such as Elastic Load Balancing (ELB) and Elastic Compute Cloud (EC2), we aim to effectively implement these DSD features for handling large-scale data.

2 Introduction

This project explores the deployment of a distributed system application, leveraging the synergies among Couchbase, AWS, and Docker. Emphasizing crucial aspects of distributed system design, such as load balancing and fault tolerance, our methodology integrates the advantages of containerization with the robust high availability and resilience offered by Docker, EC2 and ELB. This combination establishes a system configuration that is not only highly available but also resilient to faults. Our objective is to showcase a holistic approach for effectively managing extensive data in a distributed setting, addressing prevalent challenges in distributed systems, including traffic

distribution, system stability, and data consistency.

3 Preliminaries

This section reviews some of the relevant background knowledge and detailed concepts that will be used throughout this report.

3.1 Couchbase

Couchbase is a NoSQL, distributed database designed for high-performance and scalable applications. With its support for both key-value and document-oriented data models, Couchbase is relevant in distributed designs by providing seamless scalability, data distribution, and flexible schema to accommodate the evolving needs of modern, distributed architectures, making it suitable for applications with dynamic and growing data requirements.

3.2 AWS EC2

Amazon EC2 (Elastic Compute Cloud) is a web service that provides resizable compute capacity in the cloud. It is highly relevant in distributed designs due to its scalability, flexibility, and on-demand provisioning of virtual servers. EC2 enables the deployment of applications across a distributed network, allowing users to easily scale resources based on demand and build resilient and fault-tolerant systems in a cloud-based, distributed environment.

3.3 AWS ELB

Amazon Elastic Load Balancing(ELB) automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, containers, IP addresses, Lambda functions, and virtual appliances. It can handle the varying load of your application traffic in a single Availability Zone or across multiple Availability Zones.

3.4 Docker

Docker is a platform for developing, shipping, and running applications in containers. It is highly relevant in distributed designs as it offers containerization, allowing applications and their dependencies to be packaged together. Docker facilitates seamless deployment across various environments, enhances scalability, and promotes consistency in distributed systems, making it a valuable tool for building and managing containerized applications in distributed architectures.

3.5 Nginx

Nginx is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. Nginx is free and open-source software. A large fraction of web servers use NGINX, often as a load balancer.

3.6 Node.js

Node.js is a runtime environment that enables server-side execution of JavaScript.

4 System Diagram

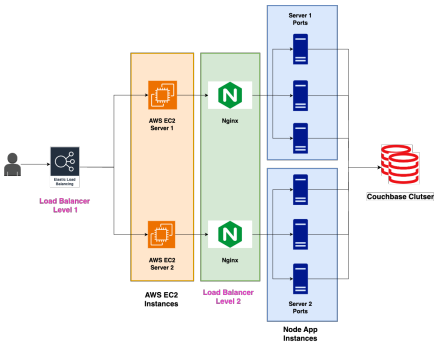

Fig 1: Overview of System Diagram

The above system diagram illustrates a comprehensive and scalable architecture leveraging AWS components, Docker containers, Nginx[7], Node.js, and Couchbase. An Elastic Load Balancer (ELB) serves as the primary entry point, distributing incoming traffic across two Amazon EC2 instances for load balancing and high availability.

Each EC2 instance hosts three Docker containers, each running a Node.js application. Crucially, within each EC2 instance, Nginx is deployed to perform load balancing across the three Docker containers, optimizing resource utilization and ensuring even distribution of requests. This dual-layered load balancing approach, with ELB at the instance level and Nginx at the container level, enhances the system's scalability and resilience.

The Node.js applications within the Docker containers seamlessly connect to a Couchbase cluster, a distributed NoSQL database, offering high performance and scalability. This architecture, blending AWS services and containerization technologies, results in a robust, scalable, and fault-tolerant system, well-equipped to handle varying workloads while ensuring continuous availability and optimal performance.

5 Demo Scenario

5.1 Setting Up the Couchbase Environment

We began by pulling the Couchbase server image from the Docker hub using the command `docker pull couchbase`. Subsequently, we launched multiple Couchbase nodes as Docker containers, each mapped to unique host ports to avoid conflicts and ensure proper node communication. The nodes are named from `couchbase_node1` to `couchbase_node6` with docker run command.

After the containers were running, we retrieved their IP addresses using `docker inspect`, which is crucial for cluster configuration and inter-node communication.

5.2 Cluster Configuration

With the nodes up and running, we proceeded to create two clusters, each comprising three nodes. This was achieved through the Couchbase web console, where we configured the clusters and ensured they were functioning correctly.

5.3 Data Management

Once the clusters were operational, we focused on data management tasks. We created a bucket in the first cluster and populated it with documents, amounting to a data chunk of 60 MB, to simulate a realistic data distribution

scenario. An empty bucket was then created in the second cluster to prepare for data replication.

5.4 Replication Process

We established a replication reference from Cluster 1 to Cluster 2 to initiate one-way data replication[2]. This was verified by observing the data presence in Cluster 2's bucket post-replication. Further, we set up a bi-directional replication reference, allowing data to be replicated back to Cluster 1 when new data was introduced into Cluster 2.

5.5 Data Sharding

In the scenario of fetching data from a Wikipedia document stored in Couchbase, which comprises roughly half a gigabyte, the utilization of data sharding[3] in distributed system design is evident. By distributing all Wikipedia items equally among the nodes in Cluster 1 while occupying comparable disk space on each node, data sharding is implemented effectively. This approach involves breaking down the large data set of Wikipedia items into smaller, manageable shards, and then distributing these shards evenly across multiple nodes within Cluster 1. This strategy not only enhances performance by allowing parallel processing of queries across distributed shards but also ensures scalability as the system can accommodate larger volumes of data. Furthermore, in Cluster 2, where only a single node exists, the data from all the Wikipedia items is replicated onto this lone node, illustrating the principle of replication in data distribution. This replication strategy ensures fault tolerance and high availability of data, even in scenarios where only a single node is present, showcasing the flexibility and adaptability inherent in the design of distributed systems using data sharding techniques.

5.6 Fault Tolerance

In the context of Couchbase's distributed system design, the implementation of fail over illustrates a fundamental aspect of fault tolerance and data redundancy. When a node within Cluster 1 fails, employing fail over mechanisms ensures continuity and data availability by redistributing the data originally stored on the failed node evenly among the remaining operational nodes. This redistribution process prevents any loss of data and maintains query accessibility even in the event of node failure. Subsequently, employing the Add Back-Full Recovery process to recover and rebalance the previously failed node facilitates the restoration of its functionality within the cluster. During this recovery phase, data from other nodes is replicated back to the recovered node, decreasing the data load on the source nodes while reintegrating the recovered node into the cluster's data distribution scheme. This fail over and recovery mechanism in Couchbase exemplifies the resilience and adaptability of distributed systems by ensuring data availability, load balancing, and system stability even in the face of node failures or disruptions.

5.7 Deployment of Couchbase on AWS EC2

EC2 provides resizable compute capacity in the cloud, which is ideal

for hosting our Couchbase nodes [5]. Each node was deployed on a separate EC2 instance, allowing us to leverage the underlying hardware's full computing, memory, and network capabilities. We utilized EC2's security groups to strictly control the traffic allowed to reach our Couchbase nodes, enhancing the security of our setup. To upload our Wikipedia Dataset(7GB) to Couchbase, we made use of the its CLI tool *cbimport*:

```
cbimport csv
--cluster <cluster_uri>
--username <username>
--password '<password>'
--bucket <bucket_name>
--scope-collection-exp
<scope_name>.<collection_name>
--dataset <dataset_path>
--generate-key '#UUID#'
--infer-types
```

5.8 Deployment of web apps on AWS EC2

To seamlessly connect with our deployed Couchbase instance we made use of the Node.js SDK of Couchbase. We then initiate the deployment of our web application infrastructure by launching two EC2 instances and strategically placing them in distinct availability zones to ensure the robustness and high availability of our application. This configuration guarantees that our application remains accessible under diverse circumstances. Subsequently, we SSH into and clone our GitHub repository onto the instance, housing our application configuration, which encompasses folders dedicated to our Node.js with Express app, including a *Dockerfile* and an Nginx folder with configuration files and an associated *Dockerfile*. Executing the below docker-compose command along with a unique `SERVER_ID` environment variable orchestrates the containerization process, utilizing the *docker-compose.yml* file from our repository.

5.9 Load Balancing with AWS ELB

To efficiently distribute the incoming traffic across our EC2 instances, we leveraged ELB[6]. This automatically scales its request-handling capacity in response to incoming application traffic, which makes it perfectly suited for managing the uneven load that can occur in a distributed database environment. The ELB was configured to perform health checks on our nodes, ensuring that only healthy instances received traffic.

6 Conclusion

This system is now guaranteed to be highly available and able to withstand spiky and massive amounts of traffic over prolonged durations of operation, given our multi-load balancer configuration deployed. By showcasing the effectiveness of Couchbase and Docker in creating a distributed database system with robust replication and fault tolerance capabilities, this project underscores the benefits of containerization, enabling the effortless deployment of multiple server and database nodes. Leveraging Couchbase's replication feature facilitated seamless data synchronization across diverse clusters. The pivotal role played by AWS EC2 and ELB in this project cannot be overstated. ELB, in particular, significantly improved traffic

distribution across instances, elevating the system's responsiveness and fault tolerance. This experiment illuminates Couchbase's robust capabilities in managing data distribution and highlights Docker's potential in streamlining the deployment of distributed systems. Incorporating AWS services, including EC2 and ELB, introduces a critical layer of availability and reliability essential for any distributed system. The insights and expertise gained from this project establish a solid foundation for further exploration into advanced load balancing and replication strategies, unlocking the full potential of cloud computing and containerization.

References

[1] Couchbase, Inc. *Couchbase Documentation*. Available at: `https://docs.couchbase.com/`

[2] Couchbase, Inc, *Cross Data Center Replication (XDCR)* Avaiable at: `https://docs.couchbase.com/server/current/learn/clusters-and-availability/xdcr-overview.html`

[3] Couchbase, Inc, *Cross Data Center Replication (XDCR)* Avaiable at: `https://couchbase.com/resources/concepts/what-is-database-sharding/`

[4] Docker, Inc. *Docker Documentation*. Available at: `https://docs.docker.com/`

[5] Amazon Web Services, Inc. *Amazon EC2 Documentation*. Available at: `https://docs.aws.amazon.com/ec2/`

[6] Amazon Web Services, Inc. *Elastic Load Balancing Documentation*. Available at: `https://docs.aws.amazon.com/elasticloadbalancing/`

[7] Nginx, Inc. *Nginx Documentation*. Available at: `https://docs.nginx.com/`