Assignment #6 - Understand Someone Else's Architecture

Software Systems Architecture

Alexandre Ferreira Nunes
André Correia da Costa
André Filipe Garcez Moreira de Sousa
Daniel José Mendes Rodrigues
Gonçalo da Costa Sequeira Pinto

March 2024

Contents

1 Introduction

In today's digital landscape, the role of a Software Architect extends beyond the mere creation of systems; it encompasses a deep understanding of the perspectives of those who interact with our creations. This assignment serves as a pivotal exploration into the evaluation of other software systems' architectures, providing a comprehensive understanding of the complexities involved in architectural design.

Our journey begins with a careful review of the Library System architecture developed by another group, in particular Team T37. As aspiring architects, we undertake the responsibility of studying their architecture, recognizing its strengths and areas for improvement. Drawing upon the knowledge accumulated throughout the course, we will provide insightful feedback that not only acknowledges their achievements but also offers constructive recommendations for refinement.

Moving to the second part of the assignment, we shift our focus to the architecture of the *Infinispan* Software System. This selection provides us with an opportunity to delve into the intricacies of a renowned open-source system, where we will present an overview of its architecture, illuminate prevalent architecture patterns, enhance existing diagrams to improve clarity and comprehension, and examine the quality attributes inherent within Infinispan.

2 T37 - Library Architecture Review

2.1 Diagrams

From our point of view, the choice of the type of letter and the overall style was not the best. It brings less clarity to the diagrams and makes them harder to read. Furthermore, the identification of the User as a box, in some diagrams, reduces the clarity of the diagrams.

The layout of most of the diagrams isn't very clear and easy to follow, especially in the last diagram: "Big Picture". The "Big Picture" diagram poses comprehension challenges due to the number of elements and connections and the use of similar names such as "Library DB" and "Libraries DB," to represent different components, making it difficult to distinguish their functions within the system. Additionally, the presence of two entry points, both related to the "User Global Platform", makes the flow of information and processes less clear. The question of resource sharing, such as whether the "Library DB" and the "Library Server" will be shared among all libraries or each will have its own, is not evident, leading to additional confusion. One way to enhance understanding of the diagram would be to divide the system into separate boxes, each previously explained, and highlight the interactions between these boxes, thereby simplifying the representation of the system. It does not use many colours, but the colours presented are clear and understandable.

In terms of consistency, the schematics follow the same style and colour

schemes, and the architecture is constant throughout.

In terms of completeness, in our opinion, there are a lot of connections and just a few are explained. Like in the "Authentication System", the User can authenticate via the System or the Library Server. The diagrams would be even more confusing with everything commented on, but to fully understand the architecture, we would need more details about the connections.

About the deepness of the details, it is very complete. Starting by explaining the main Users of the system and the different usages of the system, then the Library System, followed by interaction between both, brought a gradual improvement. Finally, the "Big Picture" diagram, sums it up all together and it is very detailed.

2.2 Text Description

The text is clear and well-structured. Have some parts highlighted that bring some clarity to the main focus of the topic. Just a sentence that we did not fully understand: "A Visitor is **someone who needs to be Authenticated** and can only use the search catalog or authenticate." We would suggest: "A Visitor is a Non-Authenticated User that can ..."

Additionally, it is consistent with the content demonstrated in the diagrams. It does not have any contradictions, to the naked eye.

Finally, it explains all the components and most of the connections in a very well-structured and completed way. It provides almost all the necessary information to understand the diagrams and resolves the problem related to the clarity of the diagrams.

In particular, we did not fully understand the "Authentication System". It required much more details and an explanation of the need for 2 separate Mobile Apps and some of its specifications.

2.3 Correctness

We do not think that the team forgot any component, but we did not fully understand the need for 2 different levels of authentication. Why would we need a Global Platform Mobile App and a Library Mobile App?

2.4 Summary

In summary, we think that it requires a restructuring for better clarity of the diagrams and in terms of the do-ability of this system. As such, we would say that, on a scale of 1 to 10, It would be a 7. To fully develop this system without any questions, we would require more information and clarity.

3 Infinispan

Infinispan, a robust and flexible data grid platform implemented primarily in Java, supplemented by some parts in Scala, offers a versatile architecture suitable for a wide range of distributed computing applications, acting as a distributed in-memory key/value data store. Understanding its architecture is essential for effectively utilizing its capabilities, whether as a library integrated into existing Java applications or as a standalone remote data grid accessed through network connections. In the upcoming sections, we enumerate and summarize the key aspects of Infinispan's software architecture.

3.1 Versatility: used as a Library or as a Server

When deployed as a library, Infinispan becomes an integral part of the Java application, sharing the same Java Virtual Machine (JVM) and heap memory. This embedding allows seamless interaction with Infinispan components programmatically, enabling developers to harness its distributed data storage and processing capabilities directly within their applications. Different user actions will be served by a different server in the cluster depending on the load balancer, and the Infinspan instances will eventually sync as shown in **Figure 1**.

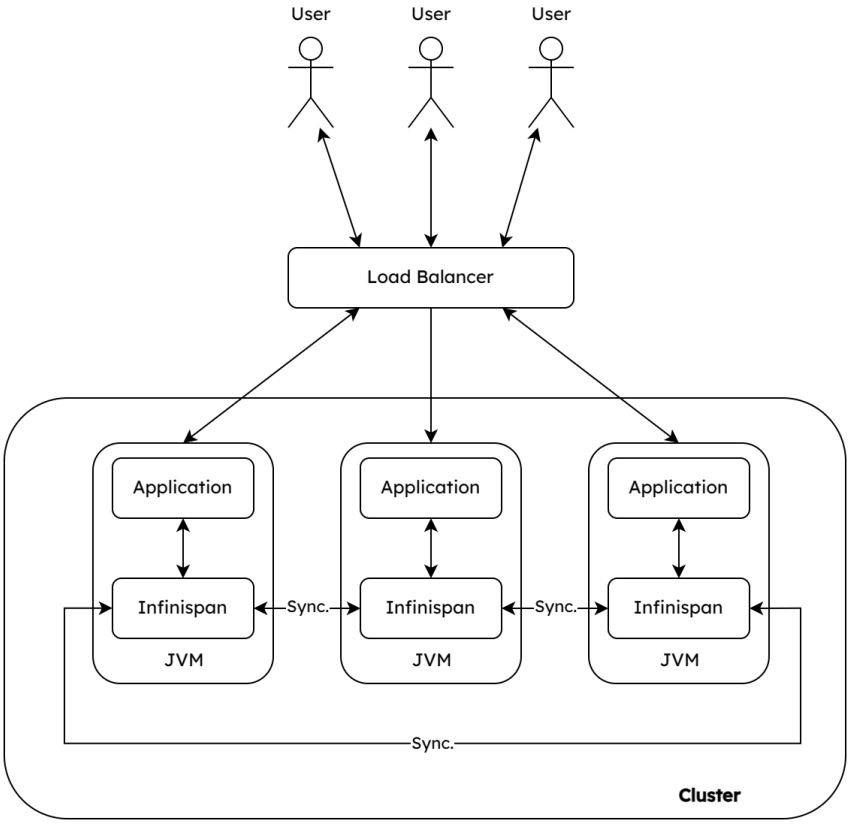

Figure 1: Infinispan as a library

Conversely, utilizing Infinispan as a remote data grid involves deploying multiple instances across a network to form a cluster. Clients can then connect to this cluster over the network, leveraging its distributed data structure transparently across all servers in the cluster **(Figure 2)**. One drawback is that Client-Server Infinispan requests are likely to take longer compared to P2P requests, due to the serialization and network cost in remote calls.

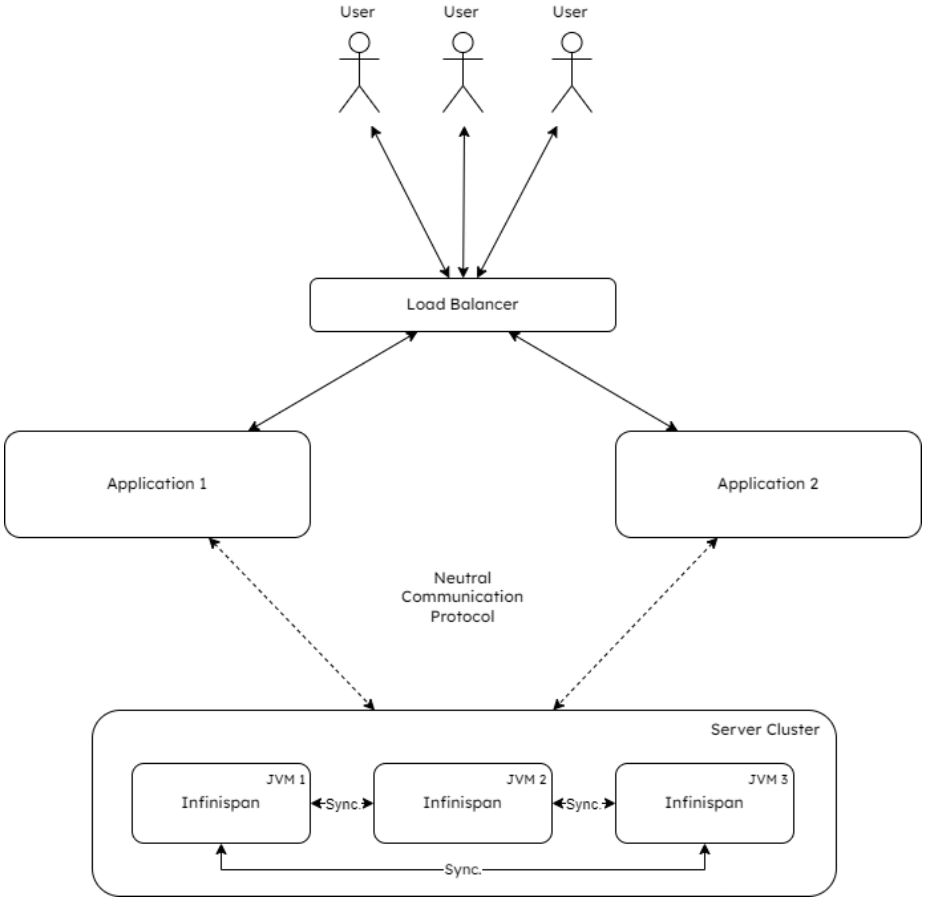

Figure 2: Infinispan as a server

3.2 Peer-to-Peer Architecture

The peer-to-peer architecture of Infinispan ensures that each node in the cluster operates on an equal footing, eliminating single points of failure and bottlenecks. This architecture enables horizontal scalability, where additional nodes can be seamlessly added to the cluster to increase overall capacity. Moreover, it facilitates elastic scaling, allowing the cluster to dynamically adjust its size based on workload demands, without compromising on functionality or performance.

3.3 Radar Gun

In the context of benchmarking Infinispan, existing tools were found to be inadequate for the task, leading to the development of Radar Gun. Radar Gun serves as a comprehensive benchmarking platform capable of evaluating multi-

ple libraries within the same JVM instance of an application, or across multiple instances of the same library. Its functionality encompasses measuring scalability and generating detailed reports based on collected data points. Moreover, it monitors memory consumption on each node. Performance assessment is conducted in terms of transactions per second, providing valuable insights into system efficiency and performance metrics.

3.4 Network Performance

Performance optimization in Infinispan is a multifaceted endeavor, addressing various subsystems that are critical to its operation. One such area is network communication, facilitated by JGroups, an open-source peer-to-peer group communication library. Optimizing network configurations, including protocol selection, buffer sizes, and thread pool sizes, is crucial to ensuring efficient communication between cluster nodes and clients. Tools like Netstat and Wireshark provide insights into network traffic, while Radar Gun aids in load testing and profiling, helping identify and mitigate potential bottlenecks.

Infinispan also uses Netty framework (a wrapper around async Java NIO framework) to create and manage server sockets, which allows:

- efficient resource utilization
- offers several ways of ensuring optimal performance (buffer sizes, thread pools)

3.5 Serialization of data

Data serialization is another important aspect of Infinispan's architecture, where efficiency is paramount. Infinispan employs a custom serialization scheme, utilizing externalizers for known data types to improve serialization and deserialization speed while minimizing the size of transmitted data. Developers also have the flexibility to register externalizers for application-specific objects, enabling efficient handling of custom data types within the Infinispan ecosystem.

3.6 Persisting data to disk

Disk persistence is a key feature of Infinispan, providing options for durability and overflow scenarios. Pluggable cache stores allow for flexible configurations, with upcoming implementations focused on enhancing performance through native code optimizations, such as direct I/O operations and system calls.

Infinispan persists data to disk using two distinct approaches:

- **Online**: the application thread blocks until data is safely written to disk.
- **Offline**: data is flushed to disk periodically and asynchronously.

3.7 Concurrency Control

Concurrency management in Infinispan leverages software transactional memory techniques to handle concurrent access to shared data structures efficiently. This approach minimizes the need for explicit locks and mutexes, optimizing CPU utilization in multicore environments and paving the way for future hardware transactional memory support.

3.8 Threads and Context Switching

Asynchronous operations in Infinispan introduce considerations for thread management and context-switching overhead. Proper configuration of thread pools is essential to efficiently handle concurrent updates and asynchronous tasks, ensuring optimal performance under varying workloads.

3.9 Garbage Collection

Garbage collection strategies play a crucial role in maintaining the performance and responsiveness of Infinispan deployments. Optimizing JVM settings, including garbage collector selection, heap size, and memory allocation, is essential to minimizing pauses and maximizing throughput, especially in environments with large and long-lived objects.

3.10 Main ideas

In summary, Infinispan's architecture is designed to prioritize fault tolerance, scalability, and performance across various subsystems, making it a versatile and efficient solution for distributed data storage and processing in Java environments. By understanding and optimizing its key components, developers can leverage the full potential of Infinispan to build robust and scalable distributed applications.

4 Architecture Patterns

- **Middleware:** Infinispan falls into a category of software called middleware, acting as a glue in between any application processing or business logic and the data storage tier.
- **P2P:** Each Infinispan instance is equal to every other instance, avoiding single points of failure and providing scalability.
- **Client-Server:** In the alternative of being used as a library, Infinispan can be used as a remote data grid server connected via sockets.
- **Shared Repository:** Infinispan employs the shared repository architectural pattern through its distributed data grid, facilitating seamless data

sharing and collaboration among nodes while maintaining consistency and coherence across the cluster.

- **Component-Based Architecture:** Infinispan follows a component-based architecture, where different functionalities are encapsulated within modular components that interact with each other through well-defined interfaces. This design promotes reusability, maintainability, and extensibility, allowing developers to easily customize and extend the system's capabilities by replacing or adding components as needed.

- **Caching Architecture:** Infinispan embodies the caching architectural pattern, serving as a distributed cache that stores frequently accessed data in memory to improve performance and reduce latency. By caching data across multiple nodes in the cluster, Infinispan enhances data access speed and scalability, while also providing mechanisms for eviction and expiration to manage cache size and ensure data freshness.

5 Main quality attributes

Infinispan, a powerful distributed data grid platform, exhibits several key quality attributes essential for its effective operation in various distributed computing scenarios, as will be shown in this section.

- **Scalability:** Infinispan's architecture is inherently scalable and capable of seamlessly expanding its capacity by adding more nodes to the cluster. This elasticity allows the system to accommodate growing workloads without sacrificing performance or functionality. The peer-to-peer nature of Infinispan ensures that each node in the cluster is equal, facilitating horizontal scaling by distributing data and processing across multiple nodes.

- **Fault Tolerance:** Infinispan prioritizes fault tolerance by eliminating single points of failure and bottlenecks. The peer-to-peer architecture ensures redundancy and resilience, enabling the system to continue operating even in the event of node failures or network partitions. As nodes dynamically join or leave the cluster, data redundancy mechanisms ensure that no data is lost, maintaining system integrity and availability.

  Furthermore, it persists data to disk which prevents case scenarios where data is lost due to node failures or overload in memory.

- **Performance:** Performance optimization is a crucial aspect of Infinispan's design, encompassing various subsystems such as network communication, data serialization, and concurrency management. Efficient network protocols, custom serialization schemes, and software transactional memory techniques contribute to maximizing throughput and minimizing latency, ensuring responsive and high-performance data access and processing.

- **Flexibility:** Infinispan offers flexibility in deployment and usage, supporting both embedded and remote deployment modes. As a library integrated into Java applications or as a standalone data grid accessed over the network, Infinispan adapts to diverse use cases and environments. Pluggable cache stores and customizable serialization schemes enable developers to tailor the system to their specific requirements, enhancing flexibility and extensibility.

6 Conclusion

Our analysis of both Team T37's Library System and the Infinispan distributed data grid platform has provided us with significant insights into software architecture principles and their practical implications.

Team T37's Library System exhibited commendable effort in structuring their architecture, yet it revealed areas for improvement, particularly in diagram clarity and completeness of descriptions. Despite these shortcomings, their consistency in presentation suggests a solid foundation for refinement.

On the other hand, Infinispan's architecture emerged as a versatile and robust solution for distributed data storage and processing. Its peer-to-peer model, coupled with flexibility in deployment and sophisticated caching architecture, underscores its suitability for diverse computing scenarios. Notably, Infinispan prioritizes scalability, fault tolerance, and performance optimization, demonstrating its readiness for complex distributed computing environments.

In conclusion, our exploration underscores the vital role of sound architectural design in software development. By understanding and implementing architectural principles effectively, developers can create systems that are not only resilient and scalable but also capable of meeting the evolving demands of modern computing environments with agility and efficiency.