Assessment Rubric
Assignment (which architecture): Automated Library System

Which team are you reviewing: T23

What is your team: T34

Team:

Topic	Strengths and suggestions for the architects
Diagrams	**Excellent**
Clarity	**Excellent**
Consistency	**Good**: In the diagrams, the same connector has multiple meanings and can get confusing. At the same time, different connectors sometimes also have the same meaning.
Completeness	**Excellent**
Sufficient Level of detail	**Very Good**: All diagrams are clear and have legends when needed.
Text Description	**Very Good**
Clarity	**Very Good:** If anything is not clear in the diagrams, the text description will mostly make up for it
Consistency	**Good.**
Sufficient?	**Good:** Some architectural decisions are only mentioned in the diagrams and not discussed in the text even though further detail about them is necessary. Example: card reader.
Correctness	
Anything missed?	**Fair**: There isn't a detailed implementation of collaboration features, inventory management considerations, and how borrowing works.
Will it work?	**Good**: Has well-defined modules, clear use case scenarios, and a structured approach. However, improvements in addressing missed ambiguities could further improve the system.
Presentation	**Very Good:** The document is organized well
Summary	We are confident that we could have implemented this architecture, but some ambiguities would hinder progress.

Explanation:

Diagrams

- Clarity: Are all the figures clear? Is it easy to understand what they mean? Are they laid out in an intuitive way? Is it clear what colors mean? If necessary, is there a legend?
- Consistency: Are the shapes and lines used consistently? Is there anything (a shape or line) that means two different things? Are colors consistent, if used?
- Complete: Is anything missing that would make the diagrams more clear?
- Sufficient level of detail: Do the diagrams give enough detail? For example, are there some components that should be broken down more?

Text Description

- Clarity: How clear is the description?
- Consistency: How consistent is the description with the diagrams?
- Sufficient: Does the text give enough information?

Correctness:

- Did the architects miss anything, such as did they forget a component?
- Will anything not work?

Summary:

- How confident are you that you could implement the design as it is described?

Infinispan

Software Systems Architecture

Group 4 - Class 3

M.EIC 2023/2024

Gustavo Costa	up202004187@edu.fe.up.pt
João Pinheiro	up202008133@edu.fe.up.pt
João Oliveira	up202004407@edu.fe.up.pt
Pedro Fonseca	up202008307@edu.fe.up.pt
Ricardo Cavalheiro	up202005103@edu.fe.up.pt

Index

Architecture Overview of Infinispan

What is Infinispan?

As a middleware solution, Infinispan acts as a crucial bridge between the data storage tiers and application logic. Its main goals are to increase fault tolerance, scalability, and performance. By utilizing Infinispan, developers can utilize it as a distributed NoSQL store that provides a competitive alternative to conventional relational databases, or as an in-memory cache to speed up data access. Because of its extensive ties to Java and Scala, the platform's architecture provides flexibility and simplicity of integration into Java-based ecosystems.

Infinispan Architectural Patterns

Peer-to-Peer Pattern

Infinispan employs a peer-to-peer architecture pattern, where each instance in the cluster equals every other instance, with no single point of failure or bottleneck. This pattern enables horizontal scalability by adding more instances to the cluster and supports elasticity by allowing instances to be added or removed dynamically without affecting overall functionality. Infinispan's peer-to-peer architecture contributes to its high availability and fault tolerance.

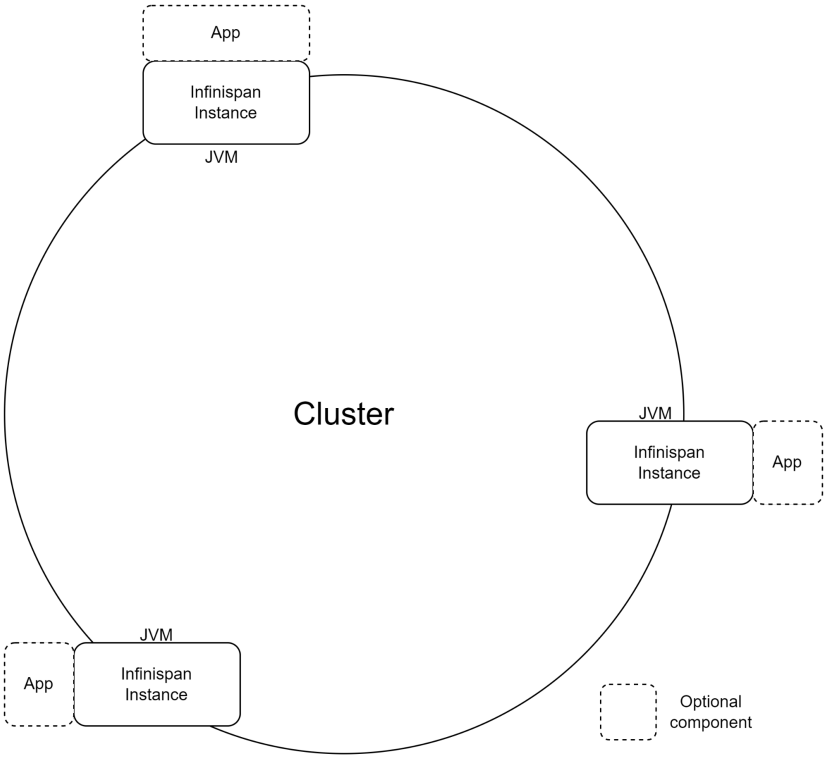

Fig 1: Infinispan’s P2P architecture that allows for both embedded and remote instancing on a JVM.

Shared Repository

Infinispan implements the Shared Repository architecture pattern by providing a distributed, in-memory data grid that allows multiple nodes to share and access data seamlessly. Infinispan acts as a shared repository where data is stored across a cluster of nodes, offering high availability, fault tolerance, and scalability.

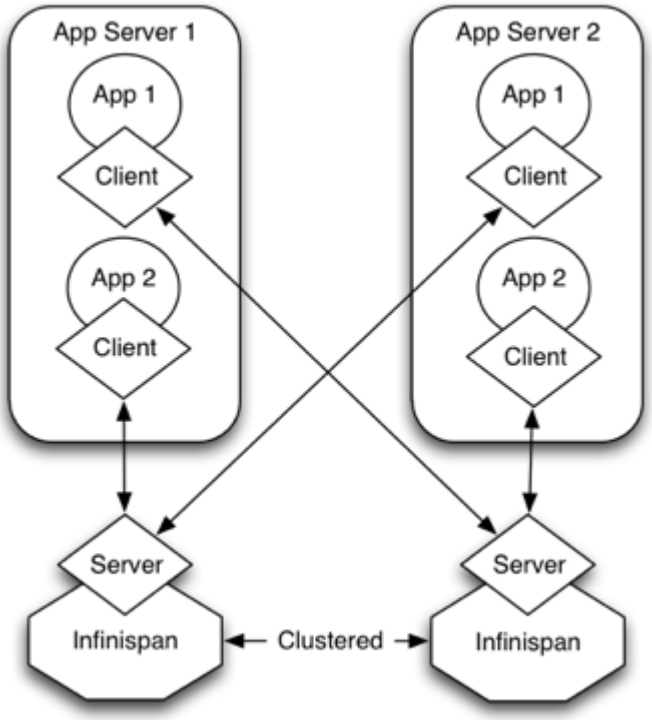

Fig 2: Infinispan's Shared Repository architecture that allows for multiple applications to access the same data

Client-Server Architecture

The client-server architecture in Infinispan provides flexibility and scalability for accessing the distributed data grid. In this mode, applications can interact with a remote Infinispan server via network protocols, enabling access from non-JVM environments or scenarios requiring elastic application tiers. This architecture facilitates load balancing, fault tolerance, and scalability by distributing client requests across multiple server instances.

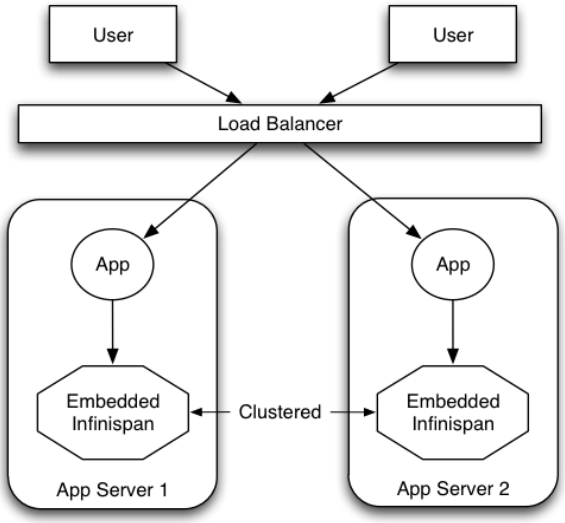

Fig 4: Infinispan’s Client-server architecture that allows users to communicate with servers via a load-balancing mechanism.

Quality Attributes

Infinispan stands as a robust open-source data grid platform, designed to meet the demands of modern distributed computing environments. At its core, Infinispan offers a distributed, in-memory key-value NoSQL store, catering to the needs of various industries, including telecoms, financial services, e-commerce, and more.

Performance

Benchmarking and finding bottlenecks

Infinispan has been fine-tuned for maximum performance, through benchmarking with tools like Radar Gun, YCSB, The Grinder, and Apache JMeter to aid in performance evaluation, scalability testing, and comparative analysis. Radar Gun, in particular, provides a comprehensive benchmarking framework tailored for distributed data structures like Infinispan. This allowed Infinispan to find the bottlenecks, such as network communication, serialization, disk persistence, and concurrency control that demanded scrutiny and optimization to harness the platform's full potential.

Data Serialization

Infinispan optimizes performance by using a custom serialization scheme with compact magic numbers instead of full class definitions, reducing serialization overhead. Predefined

externalizers for internal objects ensure efficient serialization, while developers can register externalizers for application data types to enhance serialization speed further. This approach, supported by the JBoss Marshalling library, improves overall system performance and scalability.

Threads and Context Switching

Infinispan leverages parallelism and asynchronous I/O to maximize hardware utilization. Its core data structures employ software transactional memory (STM) techniques, minimizing the need for explicit locks and synchronization primitives. This approach boosts CPU utilization in multi-core environments and enhances overall performance under load.6

Garbage Collection and JVM Tuning

Garbage collection poses a significant challenge in Java-based systems like Infinispan, impacting both performance and reliability. Proper JVM tuning, including garbage collector selection, heap configuration, and large page utilization, is essential for minimizing GC pauses and maximizing throughput. Infinispan's architecture accommodates various JVM configurations, offering insights and recommendations for optimal performance.

Scalability

Infinispan's architecture allows applications to scale **horizontally** effortlessly. By adding more instances to the cluster, whether they are running on the same physical machine or distributed across multiple machines, the system can effectively distribute the **workload** and handle increased data storage and processing requirements. This horizontal scaling capability ensures the system can accommodate growing datasets and user bases without **sacrificing** performance or reliability.

Reliability

Its architecture prioritizes fault tolerance, ensuring seamless operation even in the face of component failures. This is achieved through the distributed nature of the system, where responsibilities are evenly distributed across all nodes within the network. Consequently, if one node becomes unavailable, others can seamlessly pick up the slack, maintaining system integrity (guaranteeing that there is no **single point of failure**).

Replication

Additionally, Infinispan employs data redundancy techniques, such as **replication**, to safeguard against data loss. Furthermore, mechanisms for node recovery are integrated into the system, facilitating recovery of normal operations following disruptions.

Disk Persistence

Infinispan also offers optional **disk persistence**, enhancing reliability.

Multiple Deployment Options

Embedded and remote are Infinispan's two primary deployment techniques. Using its components programmatically within the application's Java virtual machine (JVM), Infinispan is directly incorporated into Java programs in the embedded mode. Conversely, Infinispan

instances operate independently while in the remote mode, forming clusters that clients can reach across a network.

Conclusion

A combination of cutting-edge concurrency management techniques, effective serialization, and peer-to-peer communication are all utilized in the architecture of Infinispan. Developers can fully utilize Infinispan to create scalable, high-performing distributed applications by comprehending its underlying architecture and performance considerations. JVM tuning, optimization, and benchmarking must be done continuously to ensure top performance and dependability across a range of deployment circumstances.