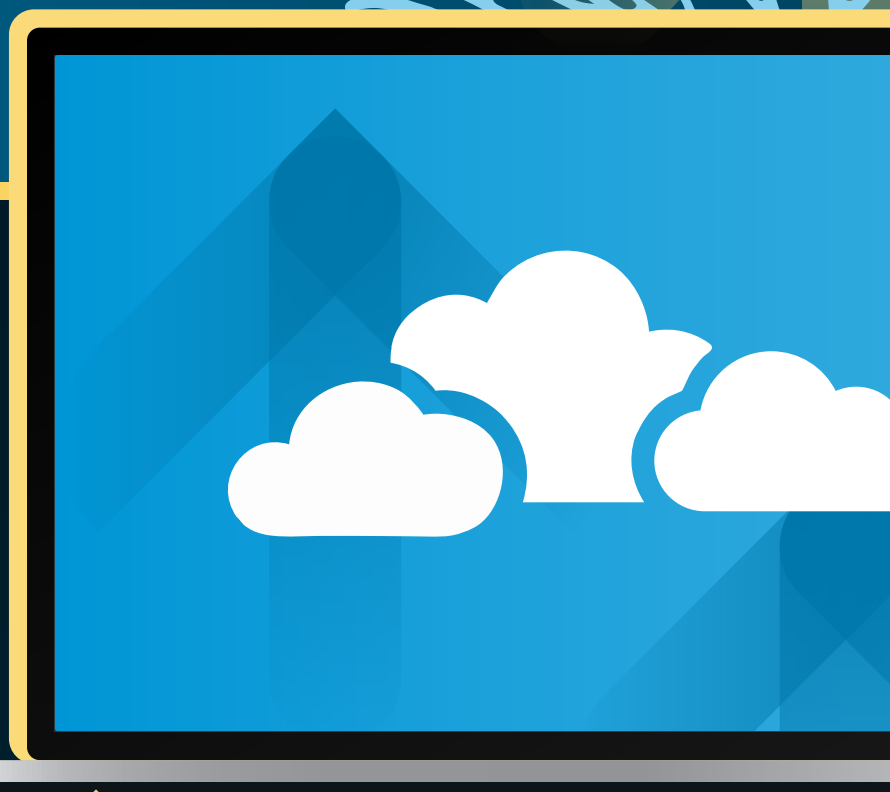
payara® COMMUNITY

Scalable Enterprise Java *for the Cloud*



Luqman **Saeed**

Otavio **Santana**

Michael **Brey**

Dario **Vega**

Contents

The New Era of Enterprise Java

When discussing software development in Java for the enterprise, it is impossible to overlook Jakarta EE. It has a long history, beginning with its origins at Sun Microsystems, then moving to Oracle, and eventually becoming fully open-source under the auspices of the Eclipse Foundation. Today, it serves as an open standard for the entire platform, focusing on new market demands, such as cloud-native development.

When considering long-term investment, Jakarta EE provides both stability and continuity, serving as the foundation of the organization. Attempting to unify both aspects directly is not possible; this is where open source emerged, with proven methodologies and practices that made it achievable.

In modern applications, it is essential to consider cloud capabilities, which simplify infrastructure and enhance application elasticity.

It is the strength of Jakarta EE: it delivers uniformity to the standard through open source, enabling organizations, companies, and individuals to move together in a single direction.

In this book, the Java Enterprise for Cloud-Native Applications refers to Jakarta EE, where we will introduce and explain NoSQL capabilities with a practical sample.

Why NoSQL Matters in the Cloud Age

When discussing the technology of human history, it is natural to recognize the role the written word has played. Through writing, we have preserved knowledge, stories, music, and more — records that remain accessible today because of what humanity has learned to document. In many ways, the written word can be seen as the first open standard in history, allowing people to communicate with one another based on shared signals.

Over time, the need for efficient ways to read, retrieve, and find information has become evident, driven by the vast amount of data that humans have been generating for centuries.

Databases emerged to facilitate the handling of this information. They represent an evolution of the temples, libraries, and even caves once used to store knowledge. From this evolution came new approaches to databases, including solutions designed to support distributed systems.

NoSQL is one of these approaches. Its goal is not to replace the classic relational database—with its maturity, decades of literature, and numerous success stories—but rather to add new options to the database landscape.

When comparing NoSQL to relational databases, one of the first differences is the lack of a universal standard. Unlike SQL, there is no single communication method that is supported across all NoSQL systems. Another key difference is the way data structures are defined, giving rise to specific types of NoSQL databases:

Key-value: Similar to a Map in Java, where information is retrieved by the key, returning a single BLOB of data.

Wide-column: Queries are limited to an ID, but allow retrieval of smaller pieces of information, stored in units called columns.

Document: Closer to relational databases, these use a structure similar to XML or JSON files.

Graph: Enables deeper relationships than relational databases by modeling connections between entities through edges, which can have direction and properties.

Time-series: Optimized for storing and serving time series, using pairs of times and associated values.

These represent only a portion of the existing NoSQL types. For instance, vector databases also belong to this category, and today there are more than 200 NoSQL solutions available.

This abundance raises challenges. Each database often requires learning a different API, which increases the burden on teams and developers who must adapt to new interfaces. Fortunately, efforts are underway to simplify this communication, aiming to deliver a better developer experience and reduce cognitive load in the Java world. Jakarta NoSQL: the specification, where the goal is to simplify and make the Java developer's life easier when handling NoSQL databases.

But before we can explore Jakarta NoSQL in practice, we need a foundation—a runtime environment that brings Jakarta EE capabilities to the cloud.

Running Jakarta EE in the Cloud

Cloud deployment introduces both opportunities and constraints. The days of monolithic application servers running for months without restart are behind us. Modern cloud platforms expect applications to start quickly, scale horizontally, and fit into container orchestration systems.

Jakarta EE addresses these requirements through multiple specifications—Jakarta REST (formerly JAX-RS) for RESTful services, Jakarta CDI for dependency injection, and dozens more that work together to simplify enterprise development. But the specifications alone don't run your application. You need a “container” that encapsulates implementations of these specifications. Or a runtime. Payara is one of such implementations. There are others, such as OpenLiberty and WildFly. For this book, we will use Payara as the Jakarta EE runtime.

Payara is a suite of products designed for different enterprise Java deployment scenarios. For this book, we focus on Payara Micro—a lightweight, cloud-native runtime that packages your application and server into a single executable JAR. No installation, no separate server process, just run the JAR and your application starts.

This approach aligns with container deployments and modern DevOps practices. Fast startup times make it practical for Kubernetes environments where pods need to scale up in seconds, not minutes.

Setting Up Your Project

You can scaffold a new Jakarta EE 11 application at https://start.payara.fish; in this case, you can skip the following manual setup.

Start with a standard Maven WAR structure. The pom.xml imports Jakarta EE 11 and MicroProfile APIs, both of which are marked with the provided scope, as Payara Micro supplies these at runtime.

The Payara BOM manages dependency versions, ensuring compatibility across all Payara-related artifacts. You specify the Payara version once, and all dependencies align with that release.

We're using Java 21. Payara supports modern Java versions, providing access to features such as records, pattern matching, virtual threads, and language improvements that enhance Java development productivity.

Development Workflow

Payara Micro includes a development mode that alters your workflow. Instead of the traditional build-package-deploy-restart cycle, dev mode watches your code and handles changes for you.

Add the Payara Micro Maven plugin to your pom.xml:

```xml
<plugin>
    <groupId>fish.payara.maven.plugins</groupId>
    <artifactId>payara-micro-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <payaraVersion>${payara.version}</payaraVersion>
    </configuration>
</plugin>
```

Start dev mode:

```shell
mvn package payara-micro:dev
```

Payara Micro starts, deploys your application, and begins watching your source files. Change a Java file, save it. The plugin detects the change, recompiles, and redeploys. Your browser refreshes when the application redeploys. Session state survives redeployments — your login state and form data remain intact through updates.

This creates a tight feedback loop. Write code, save, see results within seconds.

Production Deployment

When ready to ship, create an uber JAR that bundles everything:

mvn clean package

```shell
java -jar payara-micro.jar --deploy target/hello-world-0.1-SNAPSHOT.war
--outputUberJar hello-world-micro.jar
```

You now have hello-world-micro.jar--- a self-contained executable. Ship it to Docker, Kubernetes, or any environment with a JVM. Run it with:

```shell
java -jar hello-world-micro.jar
```

One JAR, one command, your Jakarta EE application is live. With this foundation in place, we can turn our attention to connecting Jakarta EE applications with NoSQL databases through Jakarta NoSQL and Oracle NoSQL Database.

Oracle NoSQL

Oracle NoSQL simplifies the challenges of working with NoSQL databases, especially for developers familiar with relational systems, by reducing the cognitive load. It achieves this by leveraging concepts and approaches that are common in the relational world, making it easier for experienced developers to understand and successfully adopt NoSQL technology. At the same time, developers who already have experience with NoSQL databases will recognize and appreciate the use of established NoSQL concepts, giving them a familiar and robust development environment.

Oracle NoSQL is an open-source solution licensed under the Apache 2.0 license.

The Oracle NoSQL Database is designed for today's most demanding applications, offering low-latency responses, flexible data models, and elastic scaling for dynamic workloads. It supports JSON, Table, and key-value data types, and can run both on-premises and as a cloud service with on-demand throughput and storage provisioning.

A key feature for modern, geo-distributed applications is **Global Active Tables (also known as Multi-Region tables)**. With Global Active Tables, you can develop applications that span multiple geographic regions with active-active data synchronization. This means you can build highly available, low-latency services for users worldwide, while Oracle NoSQL manages seamless data replication and conflict resolution—simplifying development for globally scaled solutions.

For document-centric workloads, Oracle NoSQL provides a SQL-like query language focused on managing and retrieving JSON documents efficiently. This helps developers use familiar, declarative techniques to access and manipulate document data without steep learning curves.

Oracle NoSQL Database Cloud Service is a fully managed offering that runs on Oracle's Gen 2 Cloud Infrastructure hardware.

To support different development needs, Oracle NoSQL provides multiple SDKs:

Java (including Spring Data and Jakarta NoSQL, which is the main focus of this e-book)

Node.js

.NET

Go

Python

Rust

*In addition, Oracle NoSQL offers plugins that **enhance the development experience:***

- Oracle NoSQL Database IntelliJ Plugin (hosted on GitHub)
- Oracle NoSQL Database Visual Studio Plugin (available in the Visual Studio Marketplace)

These plugins allow you to:

- View tables in a structured tree format using Table Explorer.
- Inspect columns, indexes, primary keys, and shard keys for any table.
- Create tables through form-based schema entry or with DDL statements.
- Create indexes.
- Execute SELECT SQL queries and view results in tabular format.
- Run DML statements to insert, update, or delete data.
- And much more.

Oracle NoSQL also provides a Docker image, making it easier to improve code quality and write tests against it.

With Oracle NoSQL, you can build applications in the cloud using Oracle Cloud or run them locally, giving you the flexibility to choose the environment that best suits your needs—including support for globally distributed, resilient architectures with Global Active Tables.

Oracle NoSQL for Developers: Fast, Familiar, and Globally Ready

Oracle NoSQL Database is engineered to accelerate developer productivity by lowering the learning curve for NoSQL data management. Come from a relational database background. You'll find your existing knowledge transferable—Oracle NoSQL thoughtfully leverages concepts that are already familiar to you, helping reduce cognitive load and jumpstart your adoption of NoSQL technologies. For developers already experienced in NoSQL, Oracle NoSQL offers well-known paradigms and advanced capabilities that make building robust, modern applications both powerful and intuitive.

Key Benefits

- **Open-Source & Enterprise-Ready:** Licensed under Apache 2.0, Oracle NoSQL gives you freedom, transparency, and no vendor lock-in.
- **Low-Latency & Elastic Scaling:** Designed to meet today's demanding, data-driven workloads with effortless scalability and lightning-fast responses.
- **Flexible Data Models:** Work natively with JSON, Table, and key-value data—all within one modern, high-performance platform.
- **Developer-Focused APIs:** Seamless development experience with SDKs for Java (including Spring Data and Jakarta NoSQL), Node.js, .NET, Go, Python, and Rust. This e-book explores how to maximize productivity with Jakarta NoSQL based on the Java SDK.
- **SQL-like Query Language for Documents:** Manage and manipulate your JSON documents using a familiar, declarative query language—making it easy to filter, update, and retrieve document data efficiently with minimal context switching.
- **Enhanced Tooling:** Boost your productivity with the Oracle NoSQL Database IntelliJ Plugin and Oracle NoSQL Database Visual Studio Plugin.

  These tools let you:

  - Explore tables and indexes visually
  - Inspect schemas, keys, and indexes
  - Create objects with forms or DDL
  - Execute queries and DML in an interactive UI
  - Easily view and troubleshoot data in tabular format
- **Local Development Made Easy:** With official Docker images, running Oracle NoSQL for local testing and CI/CD is convenient and straightforward.

Global Availability for Geo-Distributed Applications

Today's applications are distributed and global by default. Oracle NoSQL is designed for geo-distributed workloads, supporting Global Active Tables (also known as Multi-Region tables). This capability lets you build resilient, low-latency applications across multiple data centers or geographic regions, with active-active data synchronization to ensure your users always have fast, consistent access to your data wherever they are. Developers can leverage this seamless global replication and automatic conflict management without deep expertise in distributed systems.

Choose Your Deployment: Cloud or On-Premises

Run Oracle NoSQL Database as a fully managed cloud service on Oracle's Gen 2 Cloud Infrastructure, with on-demand throughput and storage that scales as your needs grow, or deploy it on-premises for maximum control.

Join the growing community of developers building cloud-native, resilient, and flexible applications with Oracle NoSQL. Whether you're transitioning from relational, experienced in NoSQL, or building your first geo-distributed app, Oracle NoSQL makes it easy for you to focus on your business logic—not data infrastructure.

Show Me The Code

With the introduction of new technologies designed to make your life easier, it's time to combine them. In this example, we will work on a library REST API.

The first step is to ensure that we have the necessary infrastructure set up, starting with the database. As mentioned previously, there are several ways to deploy Oracle NoSQL, such as on Oracle Cloud Infrastructure. As the first contact, we will run locally using a Docker command:

```shell
docker run -d --name oracle-instance -p 9999:8888 ghcr.io/oracle/nosql:latest-ce
```

Using the previous command starts the database and redirects Docker's internal port 8888 to your local port 9999, allowing us to connect to the database via port 9999.

With the database running, the next step is to prepare the Java application configuration; thus, go to: https://start.payara.fish/

At the configuration:

- Define as a Maven project
- Jakarta EE, define Jakarta EE 10 or Jakarta EE 11, Web profile
- Define Payara Micro
- Be free to define the package and set Java 21 as the minimum
- On MicroProfile, include both: MicroProfile config and Open API

It's time to generate the application.

The first step is to add a Maven dependency, incorporating Eclipse JNoSQL with a connector to Oracle NoSQL and the MapStructure. The goal is to map between layers, such as entity and DTO.

In the configurations, we will set the properties to include credentials. Once we are using Eclipse MicroProfile, we can overwrite those configurations by using the System environment, applying the best practices of Cloud Native as outlined in The Twelve Factor App.

```
#eclipse jnosql properties
jnosql.keyvalue.database=books
jnosql.document.database=books
jnosql.oracle.nosql.host=http://
localhost:9999
```

With the database instance running, the application setup complete, and the properties configured, the next step is to generate the code. We will organize our package structure by feature.

Starting with the domain, we will define the Book entity, outlining the structure that will be persisted in the database. In this case, we only need three annotations: the `Entity` annotation to designate the class as persistable in the Oracle NoSQL database, the `Id` annotation to define an identifier, and additional annotations for the attributes that will be stored in the database.

Java

```
@Entity
public class Book {
    @Id
    private String id = java.util.
UUID.randomUUID().toString();
    @Column
    private String title;
    @Column
    private BookGenre genre;
    @Column
    private int publicationYear;
    @Column
    private String author;
    @Column
    private List<String> tags;
}
```

With the database communication, we will use Jakarta Data using the Repository interface, where we can easily define an interface to communicate a NoSQL database.

```java
@Repository
public interface BookRepository extends BasicRepository<Book, String> {
}
```

Finally, on the presentation layer, where we will expose the service as BookResource, here, we can combine the resource with Open API thanks the integration of Jakarta EE and MicroProfile projects:

The integration is done, the next step is generating the JAX-RS API to expose the API, you can check the hold code at the reference, the focus here is to show how easy it so integrate the Enterprise Java with NoSQL database with Oracle NoSQL and Payara.

The last step on your journey is to execute and test the API. Start implementing the application and ensure the database is running. After that, let's run on curl commands:

To insert a book:

```shell
curl -X POST http://localhost:8080/books \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Domain-Driven Design",
    "author": "Eric Evans",
    "isbn": "978-0321125217",
    "publishedYear": 2003
  }'
```

With all insertions, you can return the books:

```shell
curl -X GET http://localhost:8080/books
```

Based on the ID we can return, update, and delete the book based on those commands:

```shell
curl -X GET http://localhost:8080/books/{id}

curl -X PUT http://localhost:8080/books/{id} \
  -H "Content-Type: application/json" \
  -d '{
    "title": "DDD: Tackling Complexity in the Heart of Software",
    "author": "Eric Evans",
    "isbn": "978-0321125217",
    "publishedYear": 2004
  }'

curl -X DELETE http://localhost:8080/books/{id}
```

From the early days of Java at Sun Microsystems to today's fully open ecosystem under the Eclipse Foundation, the story of Jakarta EE is one of evolution and endurance. What began as an enterprise framework to standardize development has matured into a foundation for cloud-native applications, combining the reliability of decades of enterprise experience with the agility required by modern architectures.

In this new landscape, **Jakarta NoSQL** emerges as a bridge between structured standards and the dynamic world of non-relational databases. It embraces the diversity of data models while restoring what enterprise developers value most—consistency, portability, and productivity. By defining a unified API for NoSQL, it extends the spirit of Jakarta EE to new paradigms, empowering developers to focus on domain logic rather than vendor-specific details.

When paired with **Payara Micro**, the runtime becomes as lightweight as the modern cloud demands. When combined with **Oracle NoSQL**, it provides a scalable, performant, and flexible data layer that is ready for real-world workloads.

In essence, this material demonstrates how the **Jakarta ecosystem unites open standards, open source, and cloud innovation** to simplify the role of the Java Software Engineer. By exploring and combining Payara Micro and Oracle NoSQL, we can enhance cloud-native capabilities, making them more scalable.

References

Source code: https://github.com/otaviojava?tab=repositories
Learn more about Oracle NoSQL: https://apexapps.oracle.com/pls/apex/r/dbpm/livelabs/view-workshop?wid=4105
Try Payara Platform: https://payara.fish/free-trials/