

Helidon Project

Helidon is a cloud-native, open-source framework for writing Java microservices that run on a fast web core powered by Netty now, and Loom later.

January 26, 2022, Version 1.1
Copyright © 2023, Oracle and/or its affiliates
Public

Table of contents

Context: Cloud-Native Microservices Applications	4
Microservices Architecture	4
Microservices Frameworks	5
Eclipse MicroProfile	5
Cloud Native Computing	7
Oracle Hatches Helidon	9
Inception and Initial Iterations	9
The Foundation is Formed	9
The Bird Flies the Nest	10
Helidon Features and Benefits	10
Open Source with Support	10
Two API Flavors for Two Programming Styles	11
Feature Richness	11
Enterprise Features	12
Integrations	13
Packaging, Footprint, and Startup Time	14
Security	15
Observability	15
Ecosystem	16
Backed by Oracle	17
Example Helidon Application Architecture with Kubernetes	17
Adoption and Innovation	18
Global Cross-Industry Adoption	18
Widespread Usage Within Oracle	18
Innovation In Step with Java	19
Fly with Helidon	19

List of images

Image 1. Helidon is Greek (Χελιδόνι) for swallow: a light, fast, agile bird.	4
Image 2. Eclipse MicroProfile 5.0 and standalone API specifications.	6
Image 3: The evolution of enterprise application development.	8
Image 4: Helidon high-level architecture.	9
Image 5: Helidon major releases, compatibilities, and contents.	10
Image 6: Helidon API flavors and programming styles.	11
Image 7: Helidon SE feature set.	12
Image 8: Helidon MP feature set.	12
Image 9: Helidon footprints and startup times by packaging option.	15

Image 10: Helidon CLI usage output.	16
Image 11: Example Helidon Application Architecture with Kubernetes	18
Image 12: Global cross-industry Helidon adoption.	18

List of tables

Table 1: Eclipse MicroProfile 5.0 API Specifications and Purposes	6
Table 2: Eclipse MicroProfile Standalone API Specifications	7



Image 1. Helidon is Greek (Χελιδόνι) for swallow: a light, fast, agile bird.

Purpose

This document provides an overview of the context, evolution, advantages, and uptake associated with Helidon, a cloud-native, open-source framework for writing Java microservices. It is intended to educate you about Helidon and encourage you to select Helidon for your Java microservices projects.

Helidon will be covered completely in upcoming sections of this document. But first, it is informative to consider the context behind Helidon's birth, for a sharper sense of the environment it engages in.

Context: Cloud-Native Microservices Applications

Powerful trends have significantly altered the trajectory of enterprise application development in the last decade. The rise of microservices architecture, microservices frameworks and standards, and cloud native computing set the context for Oracle's launch of Helidon. As important background for Helidon, these trends are reviewed below.

Microservices Architecture

The term "microservices architecture" refers to a style of enterprise application architecture in which an application is designed as an ensemble of small services, each typically running in its own process, and remotely invocable usually over HTTP¹. The processes running microservices are independently deployable, and their deployment is typically automated.

Microservices architecture emerged as an alternative to monolithic architecture, in which all server-side logic of an application is deployed as a single executable unit. The driving force for this evolution was independence: different parts of applications change at different rates, so being able to deploy them separately, instead of redeploying an entire monolith when any one part changes, is more efficient, less complex, and less disruptive. Independent deployment also lends to independent scaling of services according as the load on them, and independent implementation technology and data sources as appropriate in the organization owning the application.

¹ <https://martinfowler.com/articles/microservices.html>

User Testimonial

“When we started the project around 2019, we were looking at using Java, and we wanted to adopt a modern approach and microservices. Obviously, Helidon was designed for that. It does what we need, and it's been very good.”

Martin Hall, Helidon User

Senior Manager
Oracle Hospitality GBU

Major Trends in Enterprise Application Development

- **Microservices Architecture:** designing applications as an ensemble of small, independently deployed, remotely invocable services
- **Cloud Native Computing:** designing applications to take advantage of the cloud delivery model, and services offered by cloud providers

In short, microservices architecture has the potential to make enterprise application development and operation more agile, less brittle, and more productive.

Defining microservices architecture as an ensemble of “small” services of course begs the question of what “small” means. Microservices frameworks, including Helidon, generally support services ranging from tiny, like “Hello World,” to logic-rich functions that access data sources and advance workflows, for example.

Microservices Frameworks

As the microservices architectural style emerged, so did frameworks to help practitioners implement it, naturally. The landscape of Java microservices frameworks is crowded with many examples, dating from 2008 for a simple RESTful services framework, to 2020 for a common implementation of Eclipse MicroProfile specifications (discussed in the next section).

Though the landscape is crowded, it can be split into groupings of frameworks with similar characteristics. The group at the lightest end of the spectrum, which includes Helidon SE, supports implementing RESTful microservices without depending on any enterprise Java standards. The group at the other, heaviest, end of the spectrum, is made of open-source Java EE application servers that have been enhanced to support specifications intended specifically for Java microservices. In between these extremes there are two groups: one consisting of alternative open-source web application frameworks, and one comprising open-source implementations of standards, primarily the Eclipse MicroProfile specifications. This latter group includes Helidon MP.

Eclipse MicroProfile

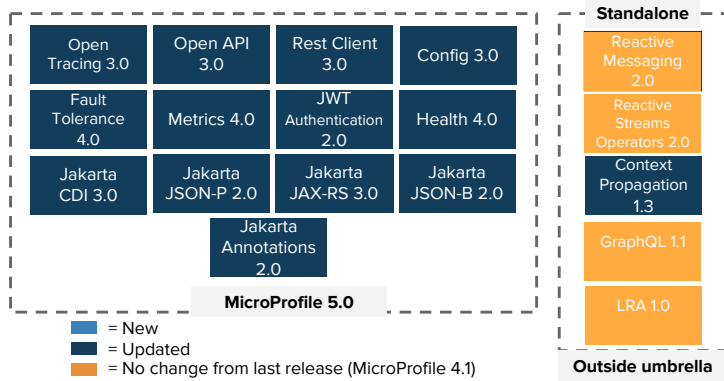
In June 2016 the MicroProfile initiative was announced, jointly by representatives of vendors and user communities in the Enterprise Java space, as a collaboration to deliver lightweight runtimes with a core set of open standard APIs enabling development of enterprise microservices. MicroProfile 1.0 was released a few months later, based on the [CDI 1.1](#), [JAX-RS 2.0](#), and [JSON-P 1.0](#) specifications as its core, familiar to Java EE developers. The MicroProfile initiative became an Eclipse project in December 2016, thereafter known as Eclipse MicroProfile, with the stated mission of “optimizing Enterprise Java for a microservices architecture.”²

Over the next six years, Eclipse MicroProfile rapidly developed and released major versions 2.0, 3.0, 4.0, 5.0, and 6.0 of its umbrella specification, with intervening minor versions, consistent with its founding goal of rapidly evolving its platform.

As diagrammed in Image 2, [Eclipse MicroProfile 5.0](#) is comprised of 13 core API specifications which are included in the umbrella specification. In addition, there are five standalone API specifications not yet included in the umbrella specification.

² https://www.eclipse.org/community/eclipse_newsletter/2017/september/article1.php

MicroProfile 5.0 (Dec 7th 2021)



HTTPS://MICROPROFILE.IO/ | HTTPS://PROJECTS.ECLIPSE.ORG/PROJECTS/TECHNOLOGY/MICROPROFILE

Image 2. Eclipse MicroProfile 5.0 and standalone API specifications.

Each MicroProfile 5.0 API specification’s purpose is summarized in Table 1, ordered by the order in which they were added to the umbrella specification.

Table 1: Eclipse MicroProfile 5.0 API Specifications and Purposes

API SPECIFICATION	PURPOSE
<u>Jakarta CDI 3.0</u>	Specifies a dependency injection mechanism with lifecycle contexts, decorations, interceptors, and event notifications.
<u>Jakarta JAX-RS 3.0</u>	Specifies a set of APIs to develop web services according to the Representational State Transfer (REST) style.
<u>Jakarta JSON-P 2.0</u>	Defines a framework for parsing, generating, transforming, and querying JSON documents.
<u>Config 3.0</u>	Defines a system for application configuration from different sources, with defaults and overrides from the environment.
<u>Fault Tolerance 4.0</u>	Defines a standard API and approach for applications to follow to achieve fault tolerance.
<u>JWT Authentication 2.0</u>	Outlines a proposal for using OpenID Connect (OIDC) based JSON Web Tokens (JWT) for Role Based Access Control (RBAC) of microservice endpoints.
<u>Metrics 4.0</u>	Proposes the addition of well-known monitoring endpoints and metrics for each process adhering to the Eclipse MicroProfile standard.
<u>Health 4.0</u>	Defines a single container runtime mechanism for validating the availability and status of a MicroProfile implementation.
<u>OpenTracing 3.0</u>	Defines an API and behaviors that allow services to participate in an environment where distributed tracing is enabled.
<u>OpenAPI 3.0</u>	Provides a set of Java interfaces and programming models which allow Java developers to natively produce OpenAPI v3 documents from their JAX-RS applications.
<u>REST Client 3.0</u>	Provides a type-safe mechanism for invoking RESTful services.

Jakarta Annotations 2.0	Defines a collection of annotations representing common semantics enabling a declarative style of programming.
Jakarta JSON-B 2.0	Defines a binding framework for converting Java objects to and from JSON documents.

Each standalone specification's purpose is summarized in Table 2, ordered by the order in which they appear in Image 2 above.

Table 2: Eclipse MicroProfile Standalone API Specifications

API SPECIFICATION	PURPOSE
Reactive Messaging 2.0	Delivers a way to build microservice systems promoting location transparency and temporal decoupling, enforcing asynchronous communication between different parts of a system.
Reactive Streams Operators 2.0	Defines an API for manipulating Reactive Streams, providing operators such as map, filter, flatMap, in a similar fashion to the java.util.stream API introduced in Java 8.
Context Propagation 1.3	Introduces APIs for propagating contexts across units of work that are thread-agnostic.
GraphQL 1.1	Provides a "code-first" set of APIs that enable users to quickly develop portable GraphQL-based applications in Java.
LRA 1.0	Introduces annotations and APIs for services to coordinate long running activities while maintaining loose coupling and guaranteeing a globally consistent outcome without needing to lock data.

In addition to the mission, rapid specification development, umbrella specification, and API specifications, Eclipse MicroProfile provides a [process](#) for certifying an implementation's compatibility with a specification, using a Technology Compatibility Kit.

Cloud Native Computing

After microservices architecture, frameworks, and standards, the rise of cloud native computing within the last decade is the other major trend setting the context for Oracle's launch of Helidon.

Of course, cloud **native** computing assumes cloud computing – a megatrend in enterprise application development not separately summarized in this document. Suffice it to say that over the last two decades, cloud computing has emerged as a compelling model of information technology ownership: essentially renting computing capacity, and infrastructure and software services, from cloud providers instead of owning the same infrastructure and software in an organization's own data center.

Cloud **native** computing refers to the concept of building and running applications to take advantage of the capabilities offered by the cloud computing model – and the services offered by cloud providers. Cloud native applications are designed and built to exploit the scale, elasticity, resiliency, and flexibility the

cloud provides, and the services available. The [Cloud Native Computing Foundation](#), founded in 2015, defines the term this way:

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

Cloud provider services empower modern application development using technologies such as Docker, Kubernetes, microservices, serverless functions, streaming, and a wide variety of ancillary application services (like authorization and storage, to name just two). DevOps services covering the entire software development lifecycle from planning to coding to production operations monitoring are part of cloud providers' services suites.

Cloud native applications are programs designed for a cloud computing architecture. They have many benefits including independence, resilience, agility, automation, observability, and availability. Cloud native applications are built with independent services packaged in self-contained lightweight containers that are portable, scalable, isolated from infrastructure, and deployable into container runtime engines based on Kubernetes. Cloud native applications are often delivered using a DevOps pipeline that includes continuous integration and continuous delivery (CI/CD) toolchains, which are important for automating building, testing, and deployment.

Cloud native architecture concerns the design of applications or services that were made specifically to exist in the cloud, rather than in a more traditional on-premises infrastructure. Microservices are the core of cloud native application architecture.













	Development Process	Application Architecture	Deployment & Packaging	Application Infrastructure
~ 1980	Waterfall 	Monolithic 	Physical Server 	Datacenter 
~ 1990				
~ 2000	Agile 	N-Tier 	Virtual Servers 	Hosted 
~ 2010	DevOps 	Microservices 	Containers 	Cloud 

Image 3: The evolution of enterprise application development.

Image 3 above depicts these broad evolutionary trends in enterprise application development that set the context for Oracle launching Helidon.

Oracle Hatches Helidon

Inception and Initial Iterations

Recognizing the rising trends of microservices architecture and cloud-native computing, a team at Oracle inceptioned a new project in 2014 for Java microservices support, from scratch, which would eventually become Helidon. Over three early iterations the concept simplified from a platform to a runtime with a micro-kernel and modules, to a framework for writing cloud native Java microservices. In fact, the name for the latter concept, before Helidon, was J4C – Java for Cloud – reflecting a focus on cloud-based deployments of microservice applications built with a simple, small, and light set of supporting components that could be used in Java SE applications.

The Foundation is Formed

In 2017 the J4C team concluded the core of its cloud-native microservices project should consist of a fast reactive web server, a configuration capability, and a security system. Netty was chosen as the core server for its speed. This set of features formed the foundation of J4C, which would become Helidon SE.

Meanwhile, the Eclipse MicroProfile project had recently been proposed and progressing. The J4C team recognized that adding MicroProfile support would provide a path for Java EE developers familiar with specifications like CDI, JAX-RS, and JPA to venture into microservices architecture with an Oracle-backed project. Adding MicroProfile support to J4C amounted to a simple step of layering MicroProfile API support on top of the J4C foundation. The initial implementation of this enhancement was accomplished in 2018, and Helidon MP was born. The resulting high-level architecture depicted in Image 4, of Helidon MP layered on Helidon SE, has remained stable ever since.

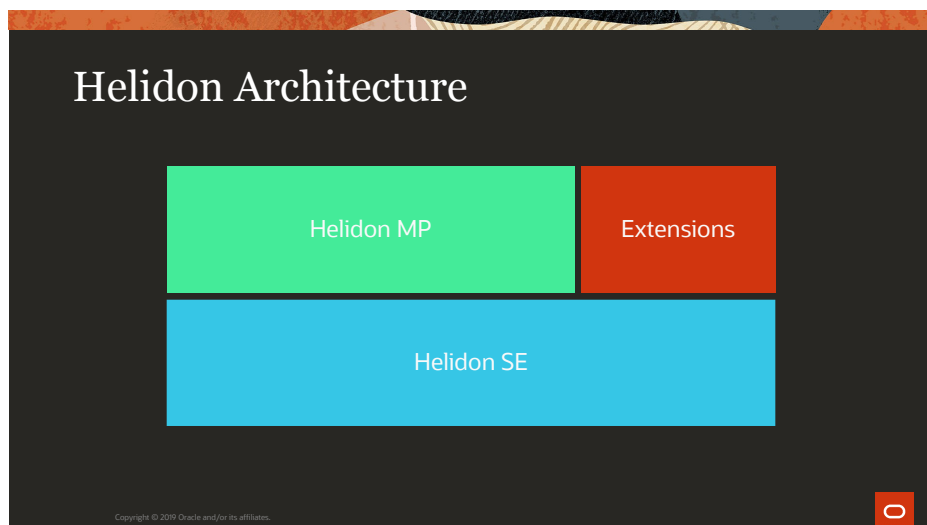


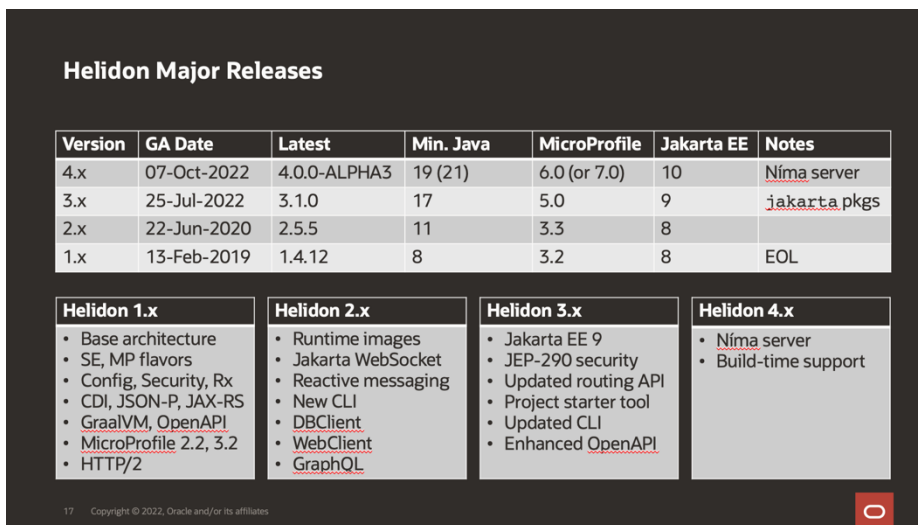
Image 4: Helidon high-level architecture.

The Bird Flies the Nest

After a carefully considered project naming process, J4C was renamed Helidon, and the project was released to the world in September 2018, as open-source software hosted on GitHub with a dedicated project [website](#) and presence on many web media properties.³

By this time, several Oracle product teams had incorporated Helidon for implementing microservices in their products, providing valuable vetting and feedback. A rich roadmap was defined, determining the direction of the project including publicizing it on the conference circuit. Principals from the Helidon team became contributors to the Eclipse MicroProfile project, and Oracle joined the Eclipse MicroProfile [Working Group](#).

Since then, Helidon has seen a series of releases, reflecting specification revisions, integrating cloud services and other software, and generally adding enhancements to the project and its ecosystem. Image 5 below tabulates those releases and their compatibilities and contents.



The image shows a slide titled "Helidon Major Releases" with a table of release details and four summary boxes for Helidon 1.x, 2.x, 3.x, and 4.x. The table lists version, GA date, latest version, minimum Java version, MicroProfile version, Jakarta EE version, and notes. The summary boxes list key features and updates for each version.

Version	GA Date	Latest	Min. Java	MicroProfile	Jakarta EE	Notes
4.x	07-Oct-2022	4.0.0-ALPHA3	19 (21)	6.0 (or 7.0)	10	Nima server
3.x	25-Jul-2022	3.1.0	17	5.0	9	jakarta pkgs
2.x	22-Jun-2020	2.5.5	11	3.3	8	
1.x	13-Feb-2019	1.4.12	8	3.2	8	EOL

Helidon 1.x	Helidon 2.x	Helidon 3.x	Helidon 4.x
<ul style="list-style-type: none">• Base architecture• SE, MP flavors• Config, Security, Rx• CDI, JSON-P, JAX-RS• GraalVM, OpenAPI• MicroProfile 2.2, 3.2• HTTP/2	<ul style="list-style-type: none">• Runtime images• Jakarta WebSocket• Reactive messaging• New CLI• DBClient• WebClient• GraphQL	<ul style="list-style-type: none">• Jakarta EE 9• JEP-290 security• Updated routing API• Project starter tool• Updated CLI• Enhanced OpenAPI	<ul style="list-style-type: none">• Nima server• Build-time support

Image 5: Helidon major releases, compatibilities, and contents.

The next section's coverage of Helidon features and benefits is based on the most recent release in the series, Helidon 3.x, while the following section's coverage of innovation by Helidon is based largely on Helidon 4.x.

Helidon Features and Benefits

Open Source with Support

Helidon is open-source software, licensed with [Apache License, Version 2.0](#). Its codebase is kept in [GitHub](#). Its artifacts are published to [Maven Central](#). This makes it easy for users to inspect, modify, and contribute to its source code. The Apache license makes it easy for organizations to adopt Helidon from a licensing perspective. Publishing artifacts to Maven Central makes it easy and natural for developers and operators to pull Helidon binaries into development environments and CI/CD pipelines. In short, Helidon is intentionally aligned with modern mainstream development practices to make it as easy as possible to adopt and use.

And yet, enterprise-grade support is also available for Helidon. Oracle offers cost-competitive commercial support for Helidon, for customers serious about support SLAs for their production operations. So, customers can get the best of both worlds: seamless incorporation of Helidon into DevOps practices and third-party product approvals, and award-winning customer support for high-scale mission-critical production applications.

Two API Flavors for Two Programming Styles

Helidon offers two API flavors: Helidon SE, and Helidon MP. Both are fun to program in, but each caters to a different style of programming:

- Helidon SE is pure reactive Java: no annotations, no dependency injection, no blocking code. It is as small and light and fast as possible.
- Helidon MP implements Eclipse MicroProfile, using CDI (including extensions) for inversion of control, JAX-RS annotations for RESTful services, and imperative coding. Those characteristics are comfortable and familiar to experienced enterprise Java programmers.

The “Hello World” code samples in Image 6 illustrate the differences between the programming styles supported by Helidon SE and Helidon MP. By offering two distinct API flavors, Helidon accommodates different preferences in programming styles within the same microservices framework.

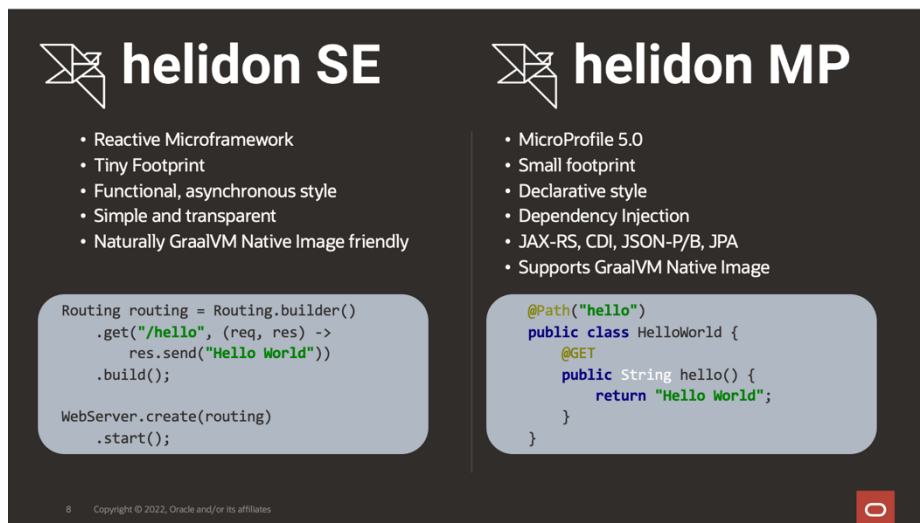


Image 6: Helidon API flavors and programming styles.

Feature Richness

Both API flavors, Helidon SE and Helidon MP, offer a rich and similar set of features, like configuration and metrics and security, as examples. In Helidon MP, the APIs for the features are specified by a standards body, whereas in Helidon SE they are not. Image 7 depicts the feature set available in Helidon SE, while Image 8 depicts the feature set available in Helidon MP. In both cases, the set of features available is complete enough to cover every aspect of the needs of modern microservices applications.

³ <https://medium.com/helidon/helidon-takes-flight-fb7e9e390e9c>

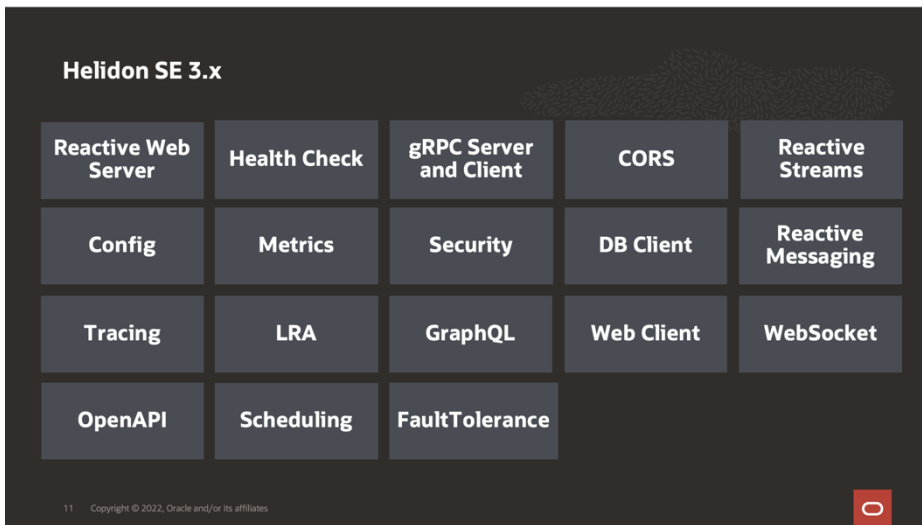


Image 7: Helidon SE feature set.

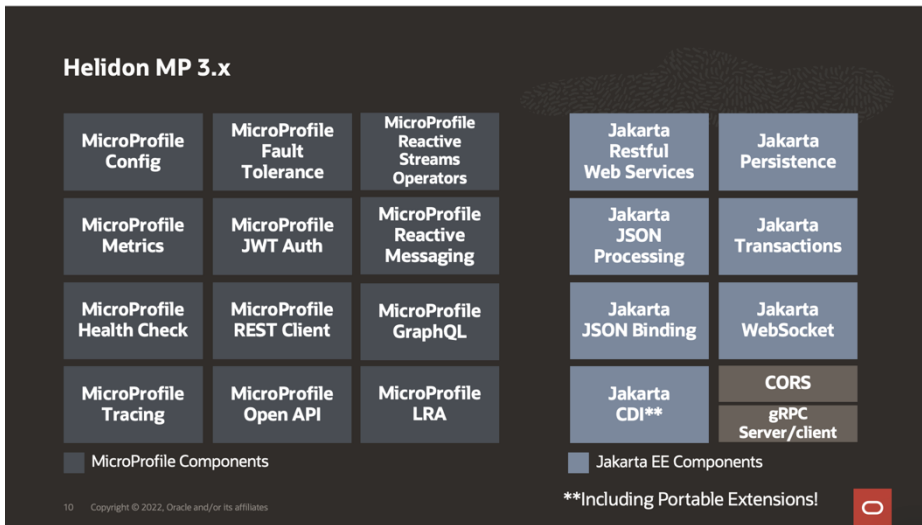


Image 8: Helidon MP feature set.

Enterprise Features

Helidon intentionally includes many features required by industrial-strength enterprise applications – even when they are now architected with microservices. Among these features are support for data access, messaging, and transactions, with integrations to existing Oracle products in each category.

For data access, Helidon SE includes a DBClient, and Helidon MP includes persistence features. Helidon SE’s DBClient provides a unified, reactive API for working with databases in a non-blocking way, supporting both JDBC-accessible databases and MongoDB. Helidon MP’s persistence features include DataSources with JDBC connection pools, Java Persistence API (JPA) integration, and Java Transaction API (JTA) integration. Oracle Autonomous Database is supported by the data access features in both Helidon API flavors.

For messaging, both Helidon SE and Helidon MP support reactive messaging, with connectors for JMS (including Oracle WebLogic JMS), Kafka, and Oracle AQ.

Transaction support in Helidon MP includes support for MicroProfile [Long-Running Actions \(LRA\)](#) and Oracle MicroTx Free, in addition to support for the Java Transaction API. LRA facilitates implementations of the Saga pattern amongst microservices: sequences of local transactions, with compensating transactions in case of local transaction failure. [Oracle MicroTx Free](#) is a free transaction manager for microservices that supports multiple transaction protocols including LRA.

Integrations

Helidon integrates with many other technologies that are useful in the implementation of microservices applications, for example:

- [Oracle Coherence](#) and [Coherence Community Edition](#), the leading in-memory data grid, which can serve as a distributed cache or system of record for stateful microservices
- [The Oracle Cloud Infrastructure \(OCI\) SDK for Java](#), for using a wide variety of OCI services from within Helidon applications
- [Oracle WebLogic Server \(WLS\)](#), including
 - Bi-directional REST service invocations
 - Helidon-to-WLS SOAP web service invocations
 - Helidon consumption and production of messages on WLS-hosted JMS destinations
 - Single sign-on between Helidon and WLS -hosted services using Oracle Identity Cloud Service
 - Distributed transaction coordination between Helidon and WLS -hosted resources using Oracle MicroTx Free
- [Messaging Connectors](#) for JMS, Kafka, and Oracle AQ, to allow Helidon applications to consume and produce messages with those providers
- [HashiCorp Vault](#) for accessing securely stored tokens, passwords, API keys, PKI certificates, and other secrets
- [Micrometer Metrics](#), for monitoring Helidon applications using Micrometer
- [Neo4j](#), for using a graph database from within Helidon applications

The Coherence and OCI SDK integrations merit further information. When using Coherence with Helidon MP, the following integration points are available:

- Coherence can be bootstrapped into Helidon MP applications via CDI, so that Helidon-based REST and gRPC services can access Coherence data when they're ready to take requests
- Coherence resources like NamedMaps or NamedTopics can be injected into Helidon MP microservices via CDI; CDI-managed objects like EventInterceptors or CacheStores can be injected into Coherence; and CDI observers can handle Coherence-generated events
- All Coherence metrics are available via standard Helidon metrics endpoints

- Coherence can be configured using MicroProfile Config, and Coherence can act as a MicroProfile ConfigSource
- Coherence tracing spans are automatically included in Helidon spans

When using the OCI SDK with Helidon, more than 100 OCI service APIs are supported, in both blocking and reactive interaction styles. The Helidon OCI support includes integration with OCI observability features. Each OCI service has its own client provided by the OCI SDK, and using the client in a Helidon application only requires adding a dependency on it to the application's pom file, then injecting the client into a Helidon service. Configuration and authorization of the OCI client can be done with Java properties files.

These integrations allow Helidon applications to leverage and benefit from the functionality of a wide variety of software typically needed in high-scale mission-critical enterprise applications.

Packaging, Footprint, and Startup Time

It has been a Helidon design goal from the beginning to be small, light, and fast – as measured by disk and memory footprint, and by Helidon JVM startup time. In relation to this goal, Helidon supports three different packaging methods for Helidon-based applications:

1. Helidon applications can be packaged as an executable jar file, with dependencies in a contained `/libs` directory, launched with a JVM in the container or environment.
2. Helidon applications can be packaged as a JLink runtime image, resulting in a customized Java runtime that is smaller and faster to start than an executable jar file; or
3. Helidon applications can be packaged as a [GraalVM native image](#) – a native executable with the smallest possible footprint and fastest start time.

In all three options, the Helidon application is a self-contained Java SE application running in its own dedicated JVM (or native executable), which can be containerized in a Docker image. Helidon provides tooling for all three options to make them easy to use.

Due to Helidon's design goal of being small, light, and fast, all three packaging options yield excellent footprints and startup times in comparison to heavyweight application servers. But the choice of packaging option further differentiates those measures for the same application, as graphed in Image 9.

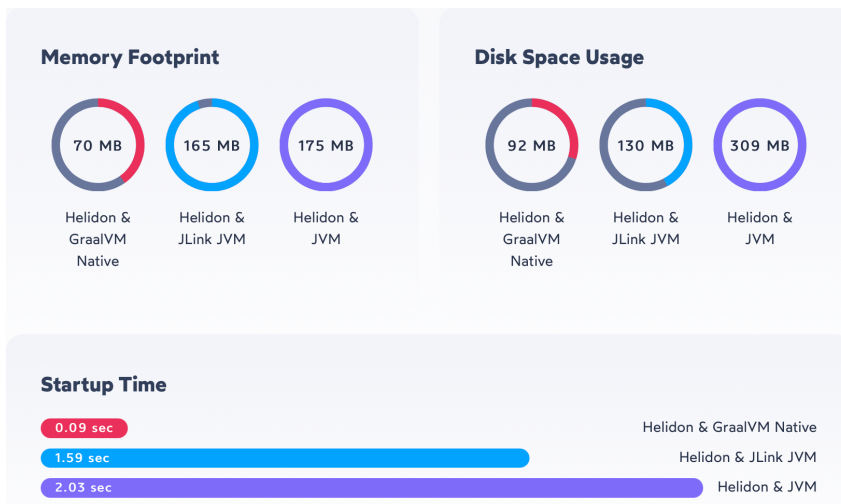


Image 9: Helidon footprints and startup times by packaging option.

Security

Helidon has built-in support in both API flavors for multiple security features, including authentication, authorization, outbound identity propagation, and auditing. Seven different security providers are supported, including Open ID Connect, and Oracle Identity Cloud Service. Helidon's OCI integration automatically picks up OCI credentials from the environment for authenticating to OCI. Encrypting configuration secrets is part of Helidon's security feature set, as is JEP-290 deserialization filtering.

In addition to the security features built into Helidon's feature set, Helidon is also developed under the umbrella of Oracle's security policies. The Helidon development team routinely applies Oracle's security policies and procedures to ensure that published Helidon artifacts are safe and up to date.

Observability

Monitoring and diagnosing the production operation of modern distributed enterprise applications is challenging. Helidon meets that challenge with a set of features specifically intended to provide thorough observability of a Helidon-based microservices application. Included in that feature set are the following capabilities:

- **Logging** – Helidon provides support for multiple logging frameworks: Java Util Logging (JUL), SLF4J, and Log4j, including Mapped Diagnostic Contexts (MDC). Helidon even provides an MDC implementation for use with JUL, since it does not contain one out of the box.
- **Metrics** - the Helidon metrics subsystem provides a unified way for Helidon servers to export monitoring data to management agents, and a unified Java API to register and update metrics to expose telemetry from their services.
- **Tracing** - Helidon includes support for distributed tracing through its own API, backed by either the OpenTelemetry API, or by OpenTracing API. Tracing is integrated with WebServer, gRPC Server, and Security.

- **Health Checks** – both Helidon SE and Helidon MP provide functionality for implementing application health checks, including liveness and readiness probes used by Kubernetes. Helidon ships with built-in health checks for common inquiries such as heap utilization and deadlock detection.

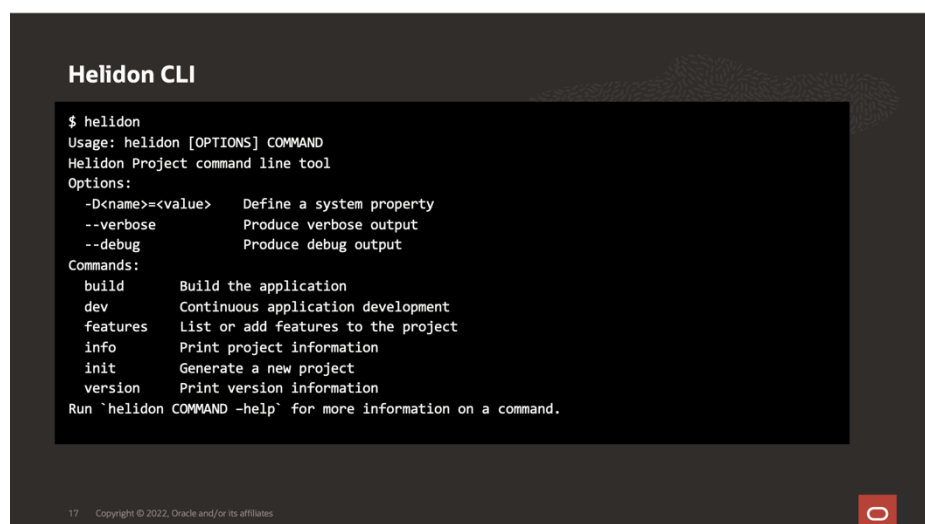
Ecosystem

Finally, in terms of features and benefits, Helidon has an ecosystem around its Java APIs and artifacts, comprised of useful tools, IDE support, rich [documentation](#), training courses and certification, very active [blog](#) publication, ubiquitous social media presence, and public communication [channels](#) to its development team.

One of the useful tools in the ecosystem is the Helidon CLI. The CLI runs in a shell and allows users to generate and build Helidon applications, add features to them, and print information about them.

The CLI includes a project starter command, `helidon init`, that walks users through a selection interview to bootstrap an application, pulling from a rich set of application building blocks or archetypes. Upon completing the interview, source code for the application, with components selected in the interview, is generated into a project directory structure. Then another simple CLI command builds the application specified. The set of archetypes is available via direct command line arguments if additional customization or options are needed.

The CLI also features a “development loop” command, wherein it monitors for changed project source files, and rebuilds and restarts the application upon detecting one, enabling a tight edit-compile-run-test loop while developing Helidon-based services. Image 10 shows the usage output for the CLI when run with no parameters.

A screenshot of a terminal window showing the usage output for the Helidon CLI. The title of the window is "Helidon CLI". The output text is as follows:

```
$ helidon
Usage: helidon [OPTIONS] COMMAND
Helidon Project command line tool
Options:
  -D<name>=<value>  Define a system property
  --verbose         Produce verbose output
  --debug          Produce debug output
Commands:
  build      Build the application
  dev       Continuous application development
  features   List or add features to the project
  info      Print project information
  init      Generate a new project
  version   Print version information
Run 'helidon COMMAND -help' for more information on a command.
```

At the bottom left of the terminal window, there is a small text: "17 Copyright © 2022, Oracle and/or its affiliates". At the bottom right, there is a small red square icon with a white circle inside.

Image 10: Helidon CLI usage output.

Another useful tool in the ecosystem is the [project starter UI](#) on the Helidon project [website](#). This UI generates a Helidon project directory structure and initial set of files to jump-start a Helidon application development effort. It allows users to select a Helidon API flavor, application archetype (from the same

rich set utilized by the CLI), and JSON library, and allows users to customize generated Maven coordinates and package names. It then downloads an archive containing a generated starter project, all from an easy-to-use web UI.

The ecosystem also includes IDE support for Helidon. There is a Helidon [plugin](#) for IntelliJ IDEA, and a Visual Studio Code [extension](#) for Helidon.

Oracle University offers training and certification for Helidon, with a [course](#) called Helidon Microservices Developer, and a corresponding [certification](#) called Oracle Certified Professional Helidon Microservices Developer.

Backed by Oracle

Though it is an open-source project, Helidon is backed by the strength of Oracle. Oracle personnel make up the core Helidon project team, which has been stable since Helidon's beginning.

Oracle offers cost-competitive commercial support for Helidon, for customers serious about support SLAs for their production operations.

Helidon is strategic technology at Oracle – one of the products within Oracle considered critical to success in delivering solutions. It is widely adopted within Oracle by applications and industry vertical solutions, as highlighted next.

Because of Helidon, Oracle is a member of the [Eclipse MicroProfile Working Group](#), and senior Helidon development managers sit on the Eclipse MicroProfile Steering Committee.

In short, Helidon is here to stay at Oracle, and in the microservices framework landscape.

Taken together, all these Helidon features and benefits have led to eager and growing adoption of Helidon as presented next.

Example Helidon Application Architecture with Kubernetes

Image 11 below depicts a generic typical Helidon application architecture with containerized Helidon services orchestrated with Kubernetes. This example is taken from an actual Helidon-based system at a customer outside Oracle, who use [Oracle Verrazzano Enterprise Container Platform](#) to run Helidon-based microservices in a Kubernetes environment.

This system has web applications running on mobile devices or laptop computers acting as clients of Helidon-based microservices running in Kubernetes and accessing data sources in the enterprise. An Istio service mesh is used for traffic ingress and service discovery. Kubernetes ecosystem tooling – Prometheus and Grafana – are used to monitor the metrics exposed by the Helidon application, and Jaeger is used as a distributed tracing UI. The Helidon-based services in the system integrate with a variety of external systems in the customer's environment for security, communication, and other functions. And in the devops activity, popular tooling is used to create containers with the application, and continuously integrate and deploy the application.

This example can likely serve as a template, or reference architecture, for many a Helidon-based microservices application.

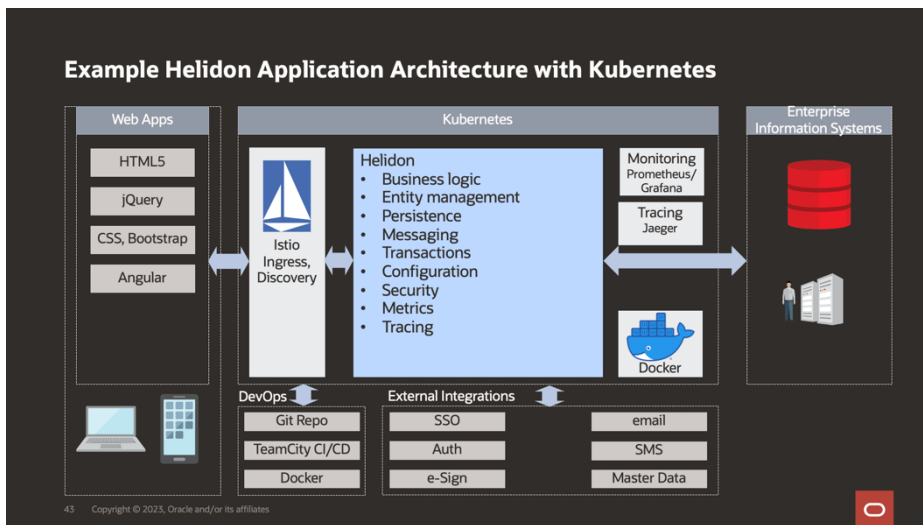


Image 11: Example Helidon Application Architecture with Kubernetes

Adoption and Innovation

In the few years since Helidon became available, it has earned extensive adoption both within Oracle amongst product teams, and externally to Oracle amongst organizations throughout the world.

Global Cross-Industry Adoption

It is always fascinating to observe how a technology product is used across industries and geographies in the world, and Helidon is no exception. Image 12 below provides a sampling of Helidon usage globally, from governments to telecommunications companies to financial services firms and more.

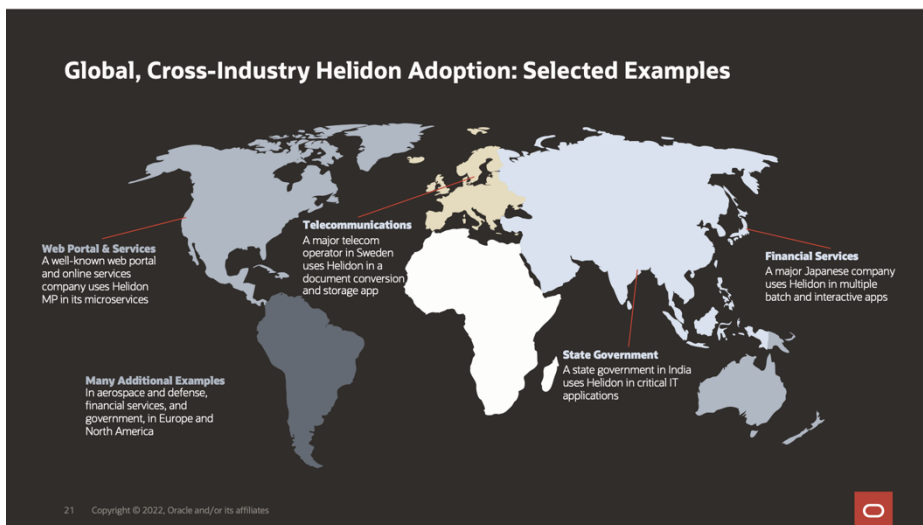


Image 12: Global cross-industry Helidon adoption.

Widespread Usage Within Oracle

Oracle is where Helidon was born, and Oracle product teams were first to vet Helidon for industrial-strength usage. Helidon held up to these teams' requirements for utility and quality in a microservices framework, and consequently has earned widespread usage within Oracle. Many of these

examples have been written up and published as Helidon success stories, available for the public to read and review:

- [A Helidon Flight: Oracle Hospitality Integration Platform](#)
- [Oracle CX Industry Framework: A Helidon Flight \(with Aerobatic Stunts!\)](#)
- [Flying in Formation: Helidon and WebLogic Integration in Oracle Communications Order and Service Management](#)

Innovation In Step with Java

Starting with two API flavors for two programming styles, Helidon has been an innovator in the microservices framework space. And Helidon is taking that habit to the next level in its 4.0 release, by leveraging virtual threads in Java 19 (part of Project Loom). Developed in close collaboration with the Java team at Oracle, a new web server named Helidon Níma will debut in Helidon 4.0, replacing Netty with an HTTP server based on virtual threads.

Níma is more than a web server. It's a complete service framework, on which additional servers like gRPC will be built, that is optimized for virtual threads. All the bits in previous Helidon servers that were optimized to presume that threads were heavyweight and should only be created sparingly, are removed. The Níma API is optimized and streamlined to take advantage of this new reality. This allows users to write code looking like it might block, without worrying that it will block: the platform thread underlying the virtual thread running the user code will continue executing other virtual threads. The overhead of the virtual threads is orders of magnitude lower than that of platform threads.

Virtual threads allow for game-changing breakthrough performance with imperative-style code. Performance testing of Helidon Níma has shown that simple imperative code can achieve the throughput of reactive code when using virtual threads. This will deliver tremendous productivity benefits to teams developing microservices applications, as they will no longer have to content with the complexity, unreadability, and difficulty of debugging reactive code. Instead, they can write simple imperative-style code, and still get the scalability of reactive code.

Fly with Helidon

Thank you for investing the time and attention to read about the context, evolution, advantages, and uptake of Helidon. It is an innovative and proven cloud-native, open-source framework for writing Java microservices, that is sure to see increasing adoption and functionality as time goes on.

With its support for Eclipse MicroProfile, Helidon is an excellent choice for migrating Java EE applications to a microservices architecture. Helidon is also an excellent choice for greenfield development, with the option of familiar Jakarta EE APIs, or lightweight reactive APIs. And Helidon supports services of all sizes, ranging from the tiniest that just return a constant, to logic-rich functions that access data sources and advance workflows.

Helidon is a great framework that you can begin using today. Your team can quickly be productive – you can begin your applications immediately using the

starter archetypes. You can develop them locally and deploy them into on-premises or cloud environments. When you are ready to move your application into production and you need the full assurance of Oracle backing, you can obtain cost-competitive commercial support, training, and expert services.

Please give Helidon serious consideration for your next microservices project. If history is any indication, it's possible yours might become the next Helidon success story!

In the meantime, here are the resources you can use to get started, and to follow the evolution of Helidon. Its development team would be happy to hear from you any time!

- Project website: <https://helidon.io>
- Medium publication: <https://medium.com/helidon>
- GitHub repository: <https://github.com/helidon-io/helidon>
- Public Slack workspace: <https://slack.helidon.io/>
- Twitter account: https://twitter.com/helidon_project
- Mastodon account: <https://mastodon.social/@helidon>
- [YouTube channel](#)
- Stack Overflow tag: <https://stackoverflow.com/tags/helidon>

Connect with us

Call **+1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

 blogs.oracle.com

 facebook.com/oracle

 twitter.com/oracle

Copyright © 2023, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0120

Author: Randy Stafford