ORACLE

Virtual threads in Helidon

The key to performance and productivity

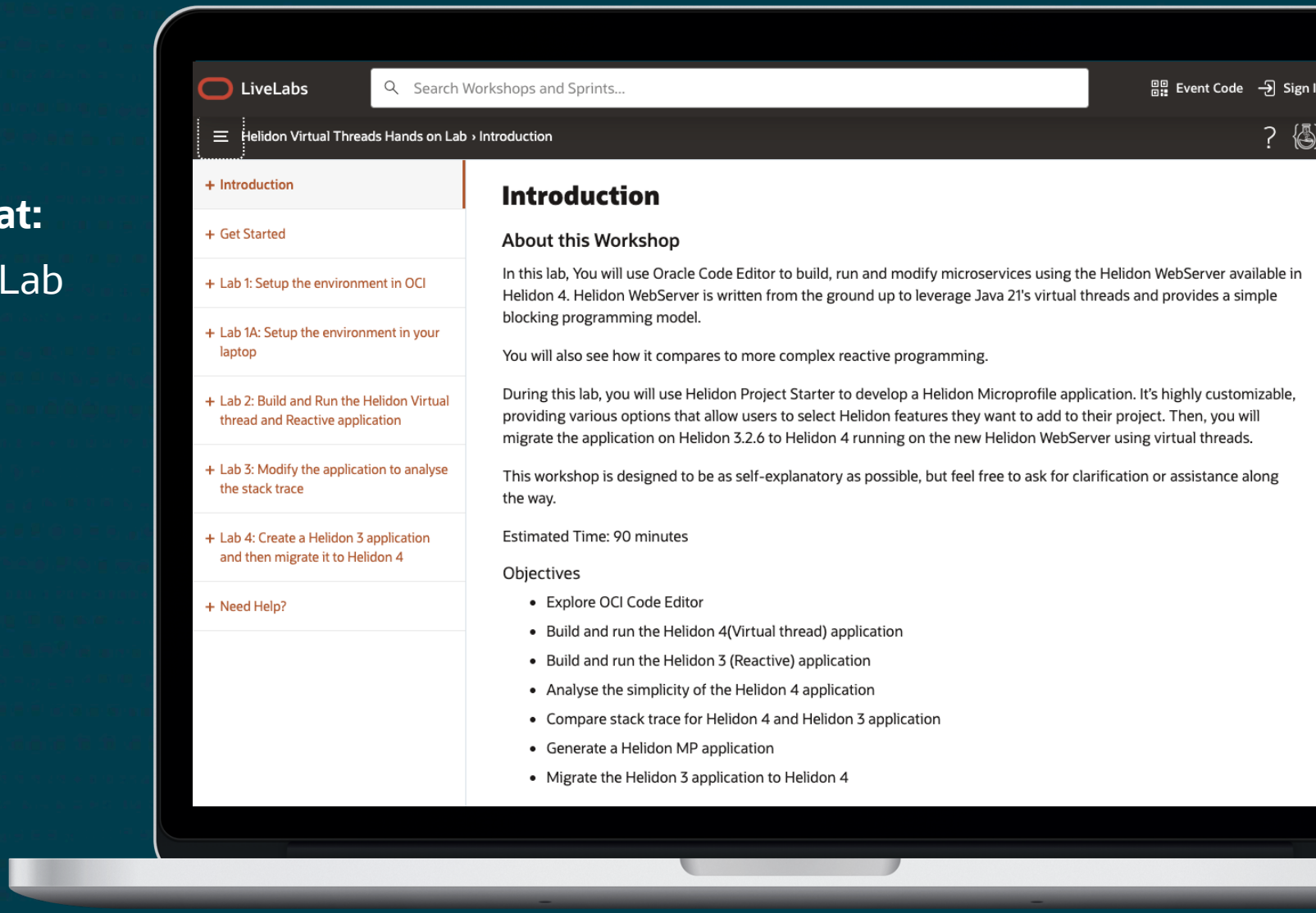
Dave Cabelus
Senior Principal Product Manager
Project Helidon, Oracle
April 11, 2024

bitly

Before we start

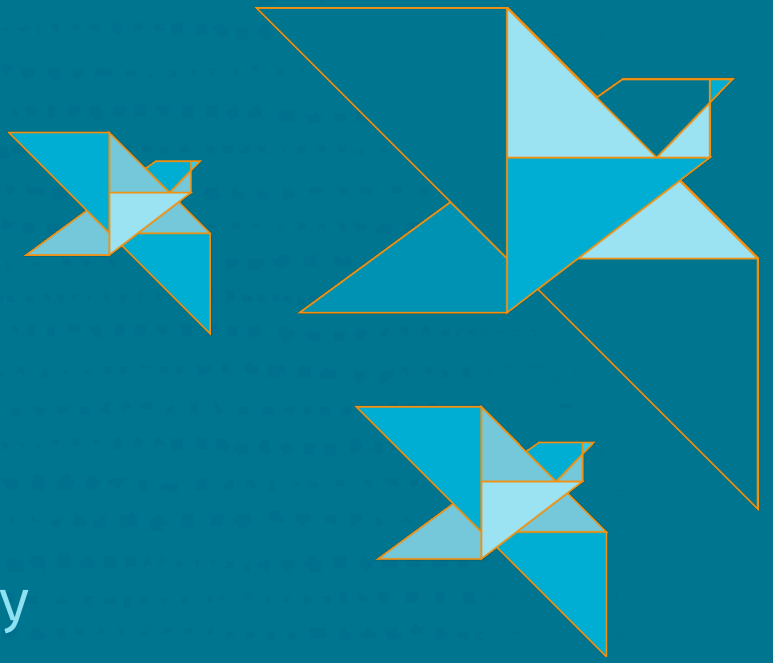
Virtual lab materials can be found at:

https://bit.ly/HelidonVirtualThreadsLab



What is **Helidon**

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



- **Framework** for developing Java (micro)services
- **Cloud native** and K8s friendly
- Your application is “just” a Java SE application
- 100% **Open Source**, on GitHub
- Multiple flavors for different use cases – SE and MP

Philosophy:

- Simplicity
- Transparency
- Fun to use
- Judicious use of external dependencies
- Adopt the latest Java JDK features

https://helidon.io ⟶ Get Started

Expensive Concurrency in Java

- Java platform-threads are mapped one-to-one to the kernel threads
- Each kernel thread created by JVM needs megabytes of memory
- Kernel threads are scheduled by OS
- **Starting new kernel thread is expensive!**
- **Context switching is expensive!**

What can we do about it?

- Reusing threads - **thread pools**
- **“Don’t block the thread!”** – Keep one thread busy, rather than multiple threads waiting – e.g., Reactive programming

A Better Solution?

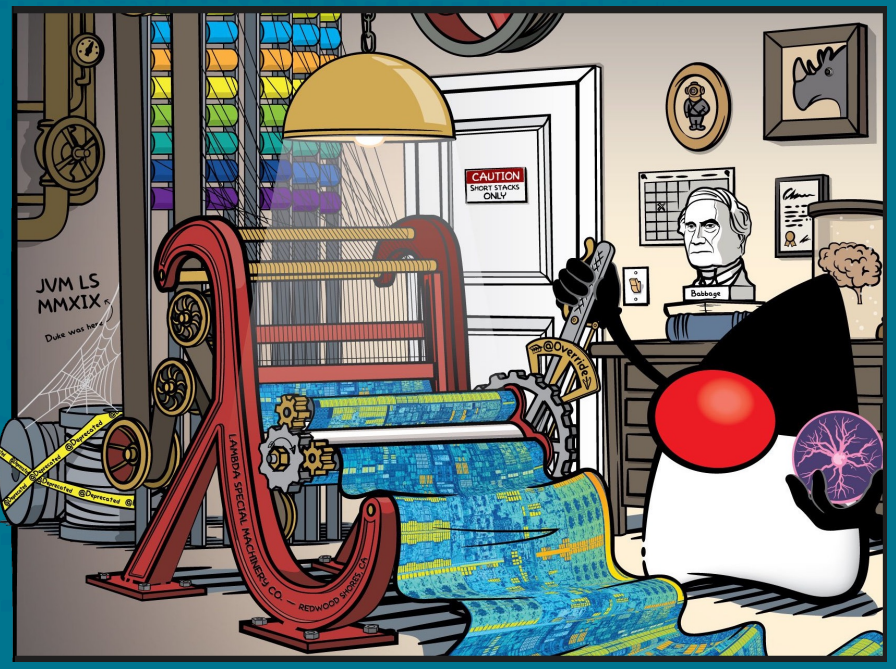
- Virtual Threads (Part of project Loom)
  - JEP-425 Preview feature since Java 19
  - JEP-444 Delivered in Java 21 (September 2023)
- Threads can now be either **Platform** or **Virtual**
- Blocking operations do not block a platform/carrier thread
- Can have a huge number of virtual threads
- Useful stack traces
- “Naive” approach to coding Java is back (and safe)

Helidon 4

Blocking is cool again

Helidon 4: No more trade offs

- First full featured framework **built for virtual threads**
- Rewritten from “socket up” for virtual threads in collaboration with the Java team
- Scalability of reactive with the **simplicity** of blocking
- Other frameworks retrofit virtual thread support
- Helidon 4 WebServer (Níma) replaces Netty
- **Excellent performance with predictable latency**



The high throughput of a reactive server with the simplicity of thread-per-request style programming

Helidon 4 WebServer: Threading Model

Virtual Threads from the Socket Up

1. Server socket listener: platform thread, one per server socket (port)
2. All connections handled by virtual threads
3. For HTTP/1: one virtual thread per connection
   - Requests are sequential on connection
   - Virtual thread reads request off of socket, calls handler, writes response, then reads next request
4. For HTTP/2: one virtual thread per connection + one virtual thread per HTTP/2 stream
   - Multiple streams are multiplexed on one connection
   - One virtual thread per connection
   - One virtual thread per stream
5. Virtual threads are very light. Millions of them!

Virtual Threads scheduling onto Platform Threads

1. Virtual thread mounts to platform thread (carrier) to run
2. Virtual thread unmounts when blocking
3. Therefore you do not block the platform thread (only the virtual thread)

```
// this code runs in virtual thread
// it is mounted to a platform thread (our carrier)
int randomInt = random.nextInt();
CompletableFuture<String> responseFuture = callHttpRequest(randomInt);

// here we unmount until the completable future returns the response
String response = responseFuture.get();
// and we continue with processing, mounted on a carrier thread again
```

Helidon 4 SE

- Switch to imperative code for improved developer productivity
  - Easier to write
  - Easier to understand
  - Easier to debug
  - Easier to maintain

Helidon 3 SE (reactive)

```
.get("/callOtherService", (req, res) -> {
    seClient.get() WebClientRequestBuilder
            .request(JsonObject.class) Single<JsonObject>
            .map(jo -> jo.getString("status")) Single<String>
            .map(String::toUpperCase)
            .onError(res::send)
            .forSingle(s -> {
                res.send(s);
            });
})
```

Helidon 4 SE (imperative)

```
.get("/callOtherService", (req, res) -> {
    String status = nimaClient.get() Http1ClientRequest
            .request(JsonObject.class) JsonObject
            .getString("status");

    String upperCaseStatus = status.toUpperCase();

    res.send(upperCaseStatus);
})
```

Helidon 4 MP

Helidon 4 MP is powered by Níma based Helidon 4 SE

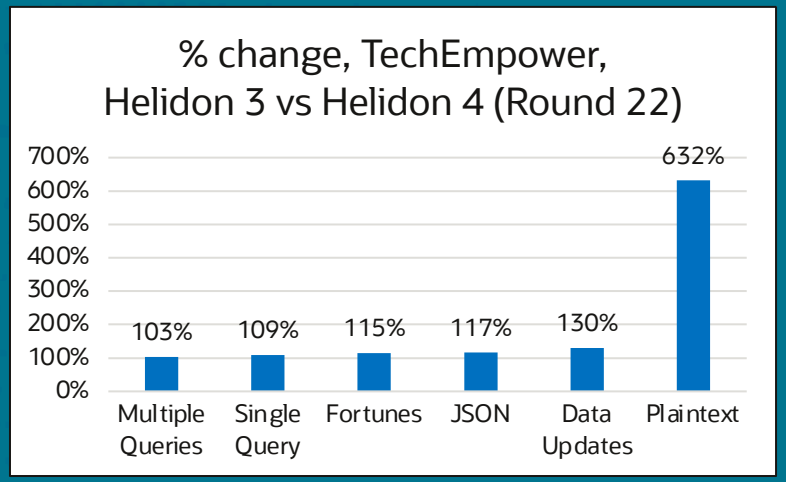
Helidon 4 + Java 22 Performance

- Helidon 4 has the leading TechEmpower benchmarks among competitors…before adding Java 22

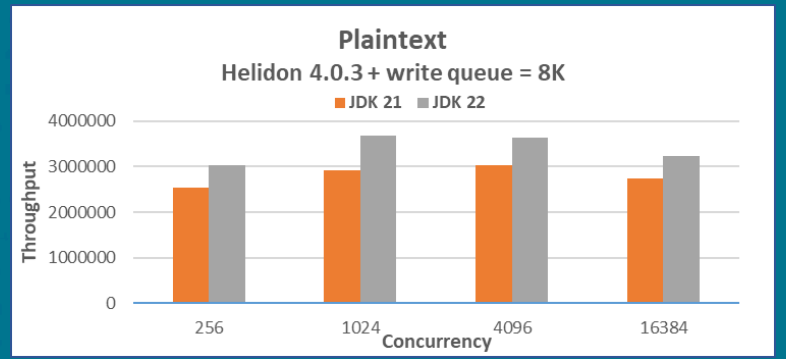
Composite Framework Scores

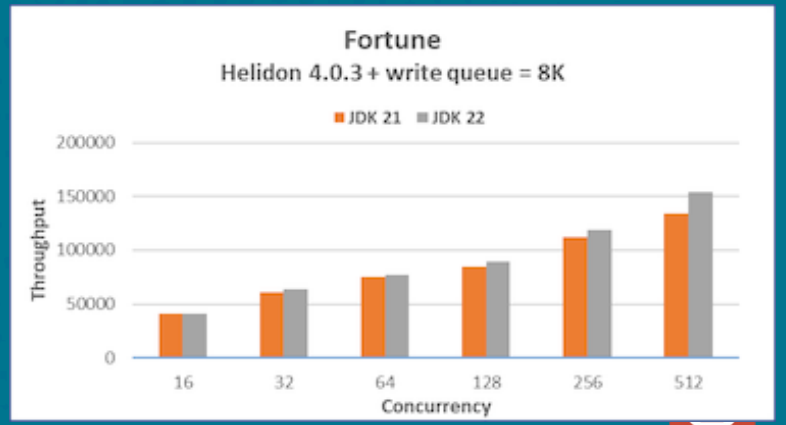
Each framework's peak performance in each test type (shown in the colored columns below) is multiplied by the weights shown above. The results are then summed to yield a weighted score. Only frameworks that implement all test types are included. 159 total frameworks ranked, 5 visible, 154 hidden by filters. See filter panel above.

Rnk	Framework	JSON	1-query	20-query	Fortunes	Updates	Plaintext	Weighted score	
37	helidon	429,240	268,833	30,291	238,545	9,390	3,035,006	**3,664**	45.3%
38	quarkus	903,185	318,897	17,610	214,275	6,697	2,861,479	**3,637**	45.0%
40	micronaut	568,955	221,741	28,171	179,741	15,209	1,327,013	**3,555**	44.0%
81	dropwizard	170,910	75,821	17,933	54,065	9,674	208,744	**1,608**	19.9%
88	spring	236,259	147,907	15,932	24,082	7,131	506,087	**1,507**	18.6%

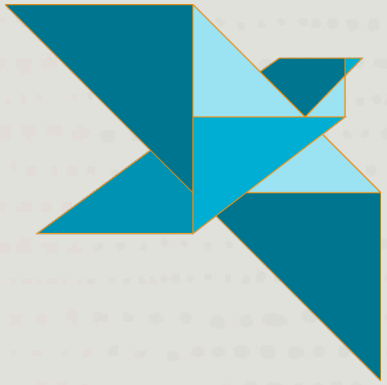


- Use of Virtual Threads continues to show benefits
  - Helidon 4 vs 3 shows a significant bump, up to 6X
  - Adding Java 22 shows even more improvements





- No code changes needed in Helidon or Helidon apps to get the benefits of the Java 22 update



= Simplicity + Productivity + Performance

Try it yourself!

LiveLabs | Search Workshops and Sprints... | Event Code | Sign In

Helidon Virtual Threads Hands on Lab › Modify the Helidon Virtual thread and Reactive application and analyse the stack trace

Task 2: Analyse stack trace for the Helidon 4 application

1. Copy and paste the following command to force an exception(count must be an integer!)

Copy

```
curl "http://localhost:<port>/parallel?count=foo"
```

You will output similar to the following:

```
$ curl "http://localhost:45043/parallel?count=foo"
For input string: &amp;quot;foo&amp;quot;$
```

2. Check the server log, you will have output similar to the below:

```
WARNING: Internal server error
io.helidon.http.RequestException: For input string: "foo"
        at io.helidon.http.RequestException$Builder.build(RequestException.java:139)
        at io.helidon.webserver.http.ErrorHandlers.unhandledError(ErrorHandlers.java:202)
        at
io.helidon.webserver.http.ErrorHandlers.lambda$handleError$1(ErrorHandlers.java:182)
        at java.base/java.util.Optional.ifPresentOrElse(Optional.java:198)
        at io.helidon.webserver.http.ErrorHandlers.handleError(ErrorHandlers.java:181)
        at
io.helidon.webserver.http.ErrorHandlers.runWithErrorHandling(ErrorHandlers.java:118)
        at io.helidon.webserver.http.Filters$FilterChainImpl.proceed(Filters.java:121)
        at io.examples.helidon.blocking.BlockingMain.lambda$routing$0(BlockingMain.java:62)
        at io.helidon.webserver.http.Filters$FilterChainImpl.proceed(Filters.java:119)
        at io.helidon.webserver.http.Filters.executeFilters(Filters.java:87)
        at io.helidon.webserver.http.Filters.lambda$filter$0(Filters.java:83)
        at
```

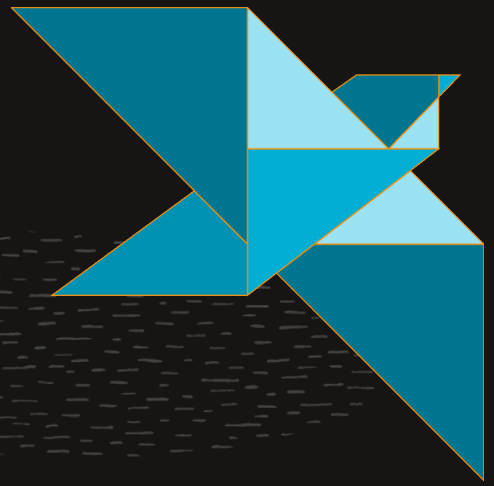
Thank You!

https://helidon.io → Get Started

https://medium.com/helidon

https://www.youtube.com/@Helidon_Project

https://github.com/helidon-io/helidon




ORACLE

Speaker

Dave Cabelus

Senior Principal Product Manager
Oracle