Going Serverless

with Spring Boot 3

Timo Salm, Senior Lead Tanzu DevX Solution Engineer

July 2024

vmware by Broadcom

Agenda

What is Serverless?

What Is Serverless?

Serverless doesn’t mean there are no servers it means you don’t care about them.

Serverless can be grouped into two areas:

- Backend as a Service (BaaS): Replacing server-side, self-managed components with off-the-shelf services
- Functions as a Service (FaaS): A new way of building and deploying server-side software oriented around deploying individual functions

The key is that with both, you don't have to manage your own server hosts or server processes and can focus on business value!

Why Serverless?

Shorter lead time

Reduced packaging and deployment complexity

Increased flexibility of scaling

Reduced labor and resource costs

Reduced risk

Drawbacks / Limitations of Serverless

- Unpredictable costs
- Spinning up machines takes time - from a few seconds to minutes
- Most Serverless applications are stateless, management of the state can be tricky
- Vendor lock-in unless you are using OSS projects like e.g. Knative
- Loss of control over
    - absolute configuration
    - the performance of Serverless components
    - issue resolution
    - security
- Higher latency due to inter-component communication over HTTP APIs and "cold starts"
- And more …

Going Serverless

with your Spring Boot applications

Demo

Running a Spring Boot application on a Serverless runtime

Unleash the Full Potential of Serverless for Our Application

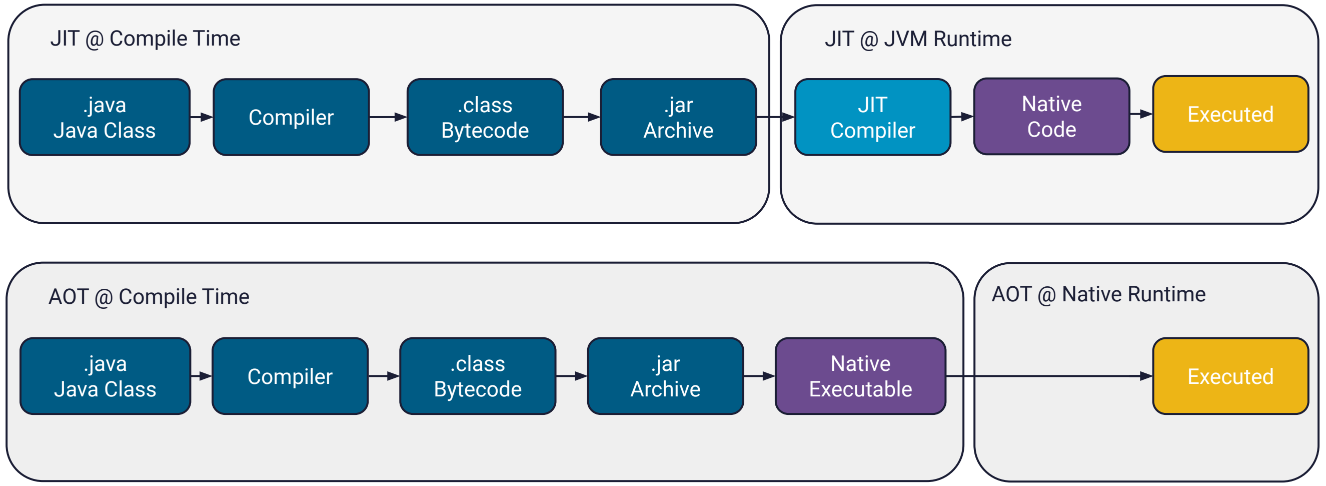
Faster startup time + Lower resource consumption (memory, CPU) → More cost savings

GraalVM Native Images

Option 1

What Are Native Images?

- Standalone executable of ahead-of-time compiled Java code
- Includes the application classes, classes from its dependencies, runtime library classes, and statically linked native code from JDK
- Runs without the need for a JVM, necessary components are included in a runtime system called “Substrate VM”
- Specific to the OS and machine architecture for which it was compiled
- Requires fewer resources than regular Java applications running on a JVM
- GraalVM is an advanced JDK with support for ahead-of-time Native Image compilation



Demo

Building and running our Spring Boot application as native image

GraalVM Native Image Tradeoffs

Developer Productivity

Compilation takes much longer and consumes more resources

Dynamic Java features may require special ”treatment”

Additional metadata required for reflection, proxies, resources, …

“Closed World” Assumptions to retain static analysis benefits

Classpath and bean conditions are fixed at build time, and manipulation of bytecode and Java agents are not supported

Providing Custom Hints With Spring Boot 3

Custom hints can be registered programmatically by implementing the `RuntimeHintsRegistrar` interface. Activate those hints with `@ImportRuntimeHints` on any Spring bean or @Bean factory method.

Hints are automatically inferred for classes that need binding (e.g., for JSON serialization). But if you use WebClient or RestTemplate directly, you might need to use `@RegisterReflectionForBinding`.

Providing Custom Hints With Spring Boot 3

Example

JAVA

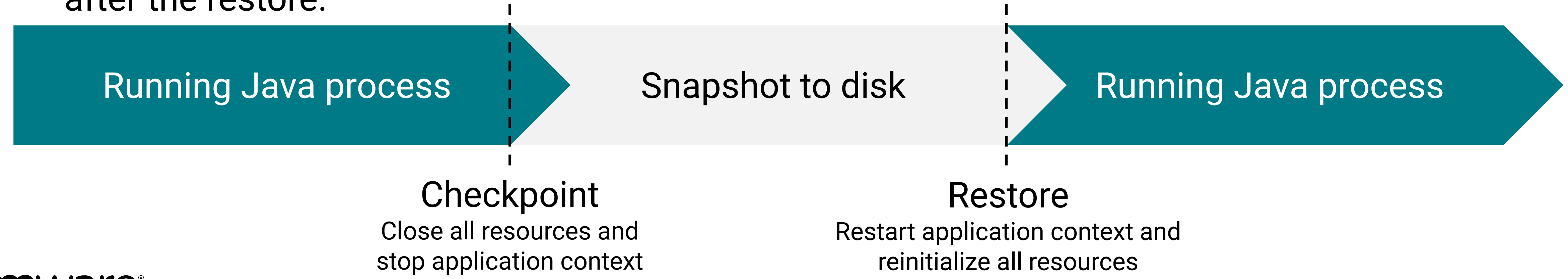
```java
public class MyRuntimeHints implements RuntimeHintsRegistrar {
    @Override
    public void registerHints (RuntimeHints hints, ClassLoader classLoader) {
        // Register method for reflection
        Method method = ReflectionUtils.findMethod(MyClass.class, "sayHello", String.class);
        hints.reflection().registerMethod(method, ExecutableMode.INVOKE) ;
        // Register resources
        hints.resources().registerPattern("my-resource.txt");
        // Register serialization
        hints.serialization().registerType(MySerializableClass.class);
        // Register proxy
        hints.proxies().registerJdkProxy(MyInterface.class);
    }
}
```

JVM Checkpoint Restore with Project CraC

Option 2

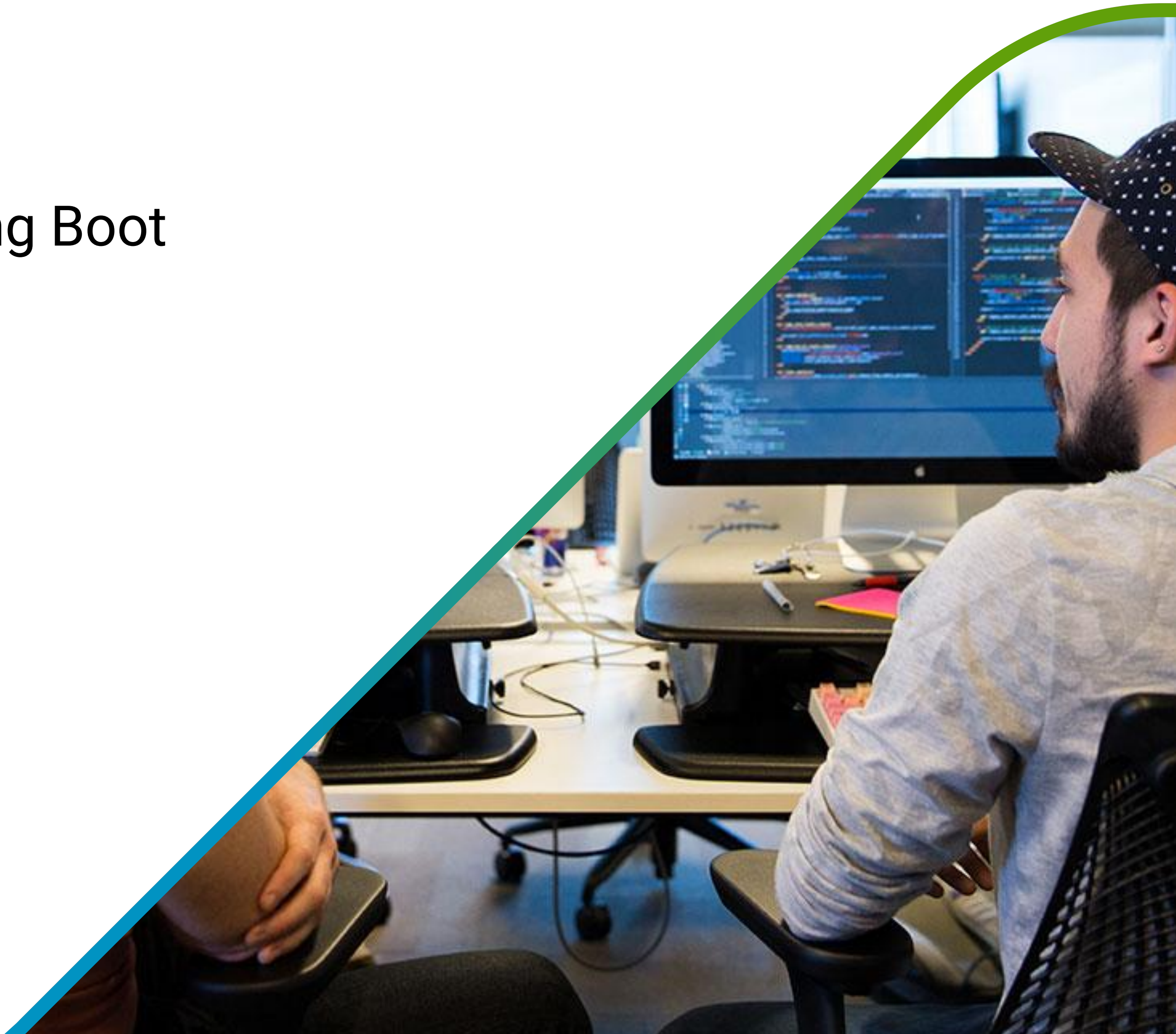
What is Project CRaC?

- Coordinated Restore at Checkpoint (CRaC) is an OpenJDK project
- Provides a Java API to take a snapshot of a Java process (checkpoint) when it is fully warmed up and restore it on any number of HotSpot JVMs
- The restored process retains all the capabilities of the HotSpot JVM, including further JIT optimizations at runtime
- Not all existing Java programs can run without modification, as all resources need to be explicitly closed before you can create a checkpoint, and these resources must be reinitialized after the restore.


Running Java process
Snapshot to disk
Running Java process
Checkpoint
Close all resources and stop application context
Restore
Restart application context and reinitialize all resources

Demo

Taking a snapshot of our running Spring Boot application and restoring it



Project CRaC Tradeoffs

Checkpoint requires fully warmed-up Java process

Additional lifecycle management

Requires graceful stopping and starting of resources and pools

Snapshot files may contain secrets and other sensitive data

Linux specific and requires some Linux capabilities

Class Data Sharing (CDS)

Option 3

What is Class Data Sharing?

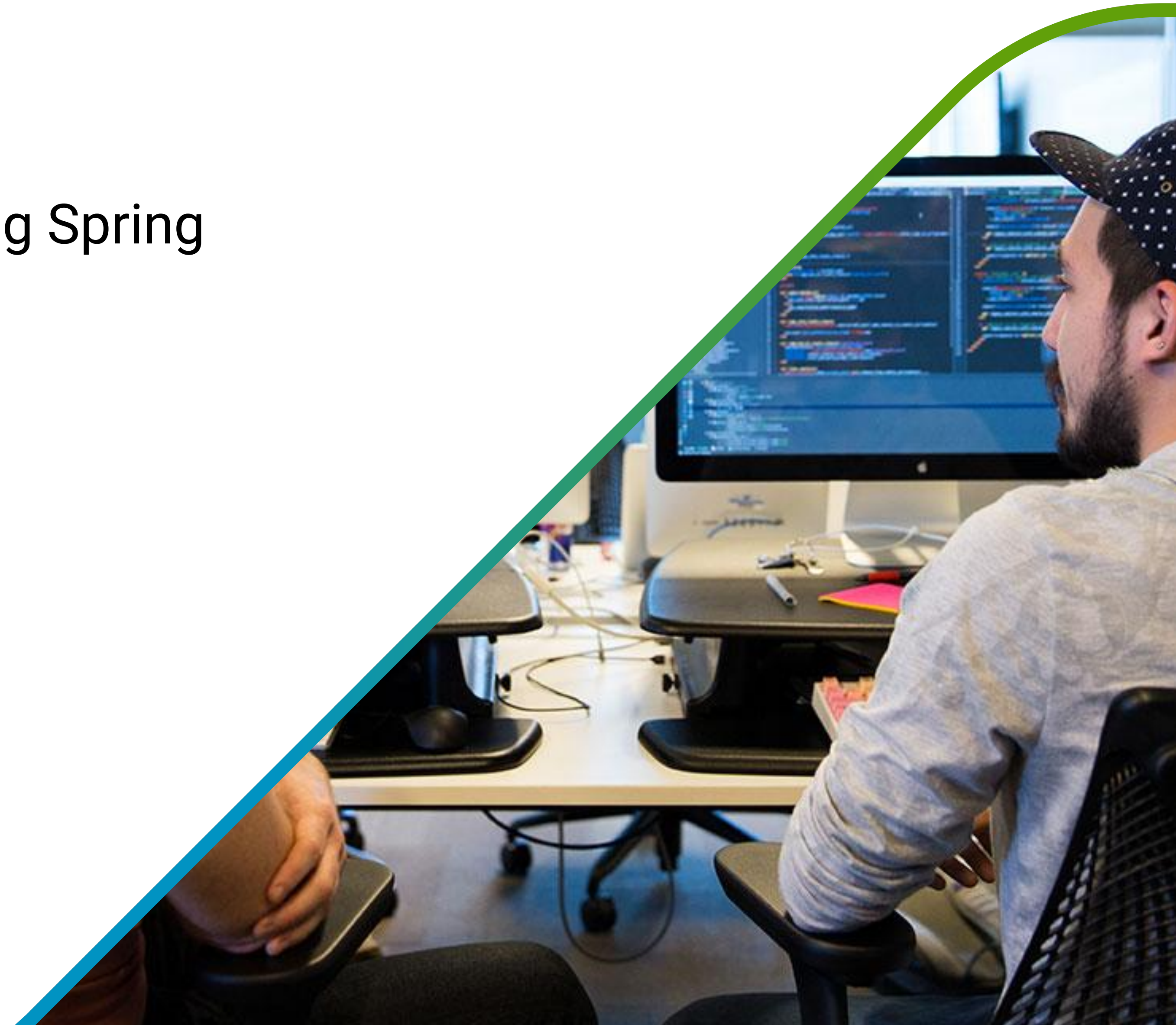
- Class Data Sharing (CDS) is a JVM feature that reduces memory footprint and improves startup time
- Mature and production-ready technology built into the JVM that continuously improves with future enhancements through Project Leyden
- Initial CDS support introduced in Spring Framework 6.1
- Full support introduced in Spring Boot 3.3
- Less restrictive than GraalVM and Project CRaC

How CDS works internally

- Class loading: Find class through classloader, verify the bytecode and link it
- Run the application, then generate the shared archive file
- Archive contains classes in a JVM-specific layout (already located, linked and verified)
- Archive is memory-mapped directly into the process, which is faster than loading the classes
- Archive can be shared between multiple JVM processes, reducing memory footprint
- If classpath has changed, the application still starts (but slower)

Demo

Creating the CDS archive of our running Spring Boot application and using it



Class Data Sharing Tradeoffs

Improvement is not as dramatic as with GraalVM or Project CRaC

Spring Boot uber JARs do not allow optimal CDS performance - additional steps necessary

JDK and classpath used for archive creation and starting the application must be identical

Future of CDS: Project Leyden’s “premain”

- JDK team in collaboration with the Spring team and others are working on CDS improvements
- “The Leyden "premain" prototype includes many optimizations that **shift work from run time to earlier** [...] **executions of the application**. [...] Importantly, we **pre-compile bytecode to native code** [...]”
- Early access builds are already available
- “We are seeing 2~4x start-up improvements with preliminary testing of popular application frameworks.”

Summary

Summary

Faster startup times and lower memory overhead reduce costs.

GraalVM Native Images

- Provide the most improvements in startup time and memory consumption
- Option with the most constraints, like significantly longer build times, additional metadata that has to be provided for dynamic language features, and no support for Spring Profiles

JVM Checkpoint Restore with Project CraC

- Similar improvements in startup time but not in memory consumption
- The solution also has several constraints. One of the most tricky ones is where to create the snapshot (build- or runtime) and how to provide it

Class Data Sharing (CDS)

- The improvements with Class Data Sharing (CDS) are not as dramatic as with the other options, and therefore, probably not a solution for scale to zero, but with hardly any constraints https://github.com/timosalm/going-serverless

Thank You