Liberica JDK
Using CRaC with Java applications

Liberica JDK
Revision 1.0
April 2024

bellsoft

BellSoft software contains open source software. Additional information about third party code is available at https://bell-sw.com/third_party_licenses. You can also get more information on how to get a copy of source code by contacting info@bell-sw.com.

The use of any software product referenced in this document is governed by the applicable license agreement, which is not modified in any way by the terms of this notice.

Contents

1. Introduction

Liberica JDK with Coordinated Restore at Checkpoint (CRaC) support enables the developers to reduce the startup and warmup times of their Java applications from seconds to milliseconds. This document provides a step-by-step tutorial on using CRaC with Java projects, including Spring Boot applications.

Prerequisites

- Liberica JDK 17 with CRaC support (Although CRaC support is available for Liberica JDK 17 and 21, in this document, we assume JDK 17 with CRaC to be your default JDK). You can download the Java builds with CRaC from the Liberica JDK Download Center.
- Spring Boot 3.2 that natively supports CRaC.

2. Installation

Note that with deb and rpm packages, CRaC works out of the box, because they grant the necessary access rights to the `criu` executable under the hood. In the case of tar.gz packages, we have to grant these rights manually as in the following example.

If you downloaded the tar.gz file, run the following commands for Liberica JDK 17.0.9 with CRaC:

```
sudo chown root:root jdk-17.0.9-crac/lib/criu
sudo chmod u+s jdk-17.0.9-crac/lib/criu
```

Check the owner and permissions of the criu executable:

```
ls -al jdk-17.0.9-crac/lib/criu
-rwsrwxr-x 1 root root 8478552 Oct 30 17:31 jdk-17.0.9-crac/lib/criu
```

3. Simple Java application

Create a simple Java application as follows:

```
public class Example {
    public static void main(String args[]) throws InterruptedException {
        // This is a part of the saved state
        long startTime = System.currentTimeMillis();
        for(int counter: IntStream.range(1, 100).toArray()) {
            Thread.sleep(1000);
            long currentTimeMillis = System.currentTimeMillis();
            System.out.println("Counter: " + counter + "(passed " +
(currentTimeMillis-startTime) + " ms)");
            startTime = currentTimeMillis;
        }
    }
}
```

> **Note:**
>
> Though CRaC preserves the exact state of the running application, the snapshot may contain secrets or other kinds of sensitive information, so you should consider and eliminate possible security risks when working with this feature.

Verify that you are using Liberica JDK with CRaC by checking the Java version, and then compile and start the application with CRaC support:

```
java --version
openjdk version "17.0.9" 2023-10-17 LTS
OpenJDK Runtime Environment (build 17.0.9+14-LTS)
OpenJDK 64-Bit Server VM (build 17.0.9+14-LTS, mixed mode, sharing)

javac Example.java

java -XX:CRaCCheckpointTo=checkpoint-dir Example
```

The `-XX:CRaCCheckpointTo=checkpoint-dir` option points to the directory where the JVM data will be stored upon the checkpoint.

Make a checkpoint with `jcmd`:

```
jcmd Example JDK.checkpoint
86221:
CR: Checkpoint ...
```

The application console output:

```
Counter: 1(passed 1007 ms)
Counter: 2(passed 1010 ms)
Counter: 3(passed 1001 ms)
Counter: 4(passed 1000 ms)
Counter: 5(passed 1000 ms)
Counter: 6(passed 1000 ms)
Nov 01, 2023 5:05:36 PM jdk.internal.crac.LoggerContainer info
INFO: Starting checkpoint
Killed
```

Now, let's try to restore the application. Run the following command (please note that only the `-XX:CRaCRestoreFrom` option is passed as the java argument):

```
java -XX:CRaCRestoreFrom=checkpoint-dir
Output:
java -XX:CRaCRestoreFrom=checkpoint-dir
Output:
Counter: 7(passed 91124 ms)
Counter: 8(passed 1000 ms)
Counter: 9(passed 1001 ms)
Counter: 10(passed 1000 ms)
Counter: 11(passed 1000 ms)
Counter: 12(passed 1001 ms)
Counter: 13(passed 1000 ms)
Counter: 14(passed 1000 ms)
```

The passed time in milliseconds in line `Counter: 7(passed 91124 ms)` shows that the application was interrupted and then restored.

4. Solving possible issues

To demonstrate the potential issues, let's write another application:

```
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class ExampleWithCRaC {
    private ScheduledExecutorService executor;
    private long startTime = System.currentTimeMillis();
    private int counter = 0;

    public static void main(String args[]) throws InterruptedException {
        ExampleWithCRaC exampleWithCRaC = new ExampleWithCRaC();
        exampleWithCRaC.startTask();
    }

    private void startTask() throws InterruptedException {
        executor = Executors.newScheduledThreadPool(1);
        executor.scheduleAtFixedRate(() -> {
            long currentTimeMillis = System.currentTimeMillis();
            System.out.println("Counter: " + counter + "(passed " +
(currentTimeMillis-startTime) + " ms)");
            startTime = currentTimeMillis;
            counter++;
        }, 1, 1, TimeUnit.SECONDS);
        Thread.sleep(1000*30);
        executor.shutdown();
    }
}
```

Start the application and make a checkpoint:

```
java -XX:CRaCCheckpointTo=checkpoint-dir ExampleWithCRaC
jcmd ExampleWithCRaC JDK.checkpoint
96828:
CR: Checkpoint ...
```

The output is the following:

```
Counter: 0(passed 1007 ms)
```

```
Counter: 1(passed 999 ms)
Counter: 2(passed 1000 ms)
Counter: 3(passed 1000 ms)
Counter: 4(passed 1000 ms)
Counter: 5(passed 1000 ms)
Counter: 6(passed 1000 ms)
Counter: 7(passed 1000 ms)
Counter: 8(passed 1000 ms)
Counter: 9(passed 1000 ms)
Counter: 10(passed 1000 ms)
Nov 01, 2023 5:41:54 PM jdk.internal.crac.LoggerContainer info
INFO: Starting checkpoint
Killed
```

Now, let's try to restore the application:

```
java -XX:CRaCRestoreFrom=checkpoint-dir
Counter: 11(passed 64673 ms)
```

The application started and then finished unexpectedly — we expected to get some iterations of the counter, but instead, more than 30 seconds passed, and the app finished.

The application should handle the checkpoint event. This can be achieved with classes from the `jdk.crac` package shipped with Liberica JDK with CRaC.

Let's modify our example and restart the counter threads. The `jdk.crac.Resource` interface helps us handle the checkpoint and restore events by implementing the `beforeCheckpoint` and `afterRestore` methods.

```
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import jdk.crac.Context;
import jdk.crac.Core;
import jdk.crac.Resource;

public class ExampleWithCRaCRestore {
    private ScheduledExecutorService executor;
    private long startTime = System.currentTimeMillis();
    private int counter = 0;

    class ExampleWithCRaCRestoreResource implements Resource {
        @Override
        public void beforeCheckpoint(Context<? extends Resource> context)
throws Exception {
```

```
            executor.shutdown();
            System.out.println("Handle checkpoint");
        }

        @Override
        public void afterRestore(Context<? extends Resource> context) throws
Exception {
            System.out.println(this.getClass().getName() + " restore.");
            ExampleWithCRaCRestore.this.startTask();
        }
    }

    public static void main(String args[]) throws InterruptedException {
        ExampleWithCRaCRestore exampleWithCRaC = new ExampleWithCRaCRestore();
        Core.getGlobalContext().register(exampleWithCRaC.new
ExampleWithCRaCRestoreResource());
        exampleWithCRaC.startTask();
    }

    private void startTask() throws InterruptedException {
        executor = Executors.newScheduledThreadPool(1);
        executor.scheduleAtFixedRate(() -> {
            long currentTimeMillis = System.currentTimeMillis();
            System.out.println("Counter: " + counter + "(passed " +
(currentTimeMillis-startTime) + " ms)");
            startTime = currentTimeMillis;
            counter++;
        }, 1, 1, TimeUnit.SECONDS);
        Thread.sleep(1000*30);
        executor.shutdown();
    }

}
```

Let's try one more time. Compile the application and make a checkpoint:

```
javac ExampleWithCRaCRestore.java
java -XX:CRaCCheckpointTo=checkpoint-dir ExampleWithCRaCRestore
Counter: 0(passed 1007 ms)
Counter: 1(passed 999 ms)
Counter: 2(passed 1000 ms)
Counter: 3(passed 1000 ms)
Counter: 4(passed 1000 ms)
Counter: 5(passed 1000 ms)
Counter: 6(passed 1000 ms)
```

```
Nov 01, 2023 5:54:04 PM jdk.internal.crac.LoggerContainer info
INFO: Starting checkpoint
Handle checkpoint
Killed
```

Now, run

```
jcmd ExampleWithCRaCRestore JDK.checkpoint
100389:
CR: Checkpoint ...
```

Finally, let's restore our application:

```
java -XX:CRaCRestoreFrom=checkpoint-dir
ExampleWithCRaCRestore$ExampleWithCRaCRestoreResource restore.
Counter: 7(passed 61407 ms)
Counter: 8(passed 1000 ms)
Counter: 9(passed 1000 ms)
Counter: 10(passed 1000 ms)
Counter: 11(passed 1000 ms)
```

As you can see, the issue is solved.

5. Reference Spring Boot application

First of all, let's try to use CRaC with the previous Spring Boot version. We will use the Spring Boot Petclinic application as an example. The source code is available here.

After pulling the project, build the `jar` file with `mvn` package and use the following command:

```
java -XX:CRaCCheckpointTo=cr -jar ./target/spring-petclinic-3.2.0-SNAPSHOT.jar
```

```
              |\      _,,,--,,_
             /,`.-'`'   ._  \-;;,_
  _______ __|,4-  ) )_   .;.(__`'-'__     ___ __    _ ___ _______
 |       | '---''(_/._)-'(_\_)   |   |   |   |  |  | |   |       |
 |    _  |    ___|_     _|       |   |   |   |   |_| |   |       | __ _ _
 |   |_| |   |___  |   | |       |   |   |   |       |   |       | \ \ \ \
 |    ___|    ___| |   | |      _|   |___|   |  _    |   |      _|  \ \ \ \
 |   |   |   |___  |   | |     |_|       |   | | |   |   |     |_    ) ) ) )
 |___|   |_______| |___| |_______|_______|___|_|  |__|___|_______|  / / / /
 ==========================================================================/_/_/_/

:: Built with Spring Boot :: 3.2.0


2023-11-24T02:24:08.205-08:00  INFO 17053 --- [           main]
o.s.s.petclinic.PetClinicApplication     : Starting PetClinicApplication
v3.2.0-SNAPSHOT using Java 21.0.1 with PID 17053 (.../target/spring-petclinic-
3.2.0-SNAPSHOT.jar started ...)
...
```

Checkpoint the running application:

```
jcmd spring-petclinic JDK.checkpoint
17053:
An exception during a checkpoint operation:
jdk.crac.CheckpointException
        Suppressed: jdk.crac.impl.CheckpointOpenSocketException:
sun.nio.ch.ServerSocketChannelImpl[/[0:0:0:0:0:0:0:0]:8080]
                at
```

```
java.base/jdk.internal.crac.JDKSocketResourceBase.lambda$beforeCheckpoint$0(JDK
SocketResourceBase.java:44)
                at java.base/jdk.crac.Core.checkpointRestore1(Core.java:174)
                at java.base/jdk.crac.Core.checkpointRestore(Core.java:299)
                at
java.base/jdk.crac.Core.checkpointRestoreInternal(Core.java:312)
```

As you can see, CRaC failed to perform a checkpoint due to `jdk.crac.CheckpointException(jdk.crac.impl.CheckpointOpenSocketException)`.

To use CRaC with the new Spring Boot and Spring Framework we need to add dependency for the `org.crac/crac` package to pom.xml.

```
<dependency>
    <groupId>org.crac</groupId>
    <artifactId>crac</artifactId>
    <version>1.4.0</version>
</dependency>
```

Let's check the difference of our updated pom.xml:

```
git diff
diff --git a/pom.xml b/pom.xml
index 287a08a..f403155 100644
--- a/pom.xml
+++ b/pom.xml
@@ -36,6 +36,11 @@
   </properties>

   <dependencies>
+    <dependency>
+        <groupId>org.crac</groupId>
+        <artifactId>crac</artifactId>
+        <version>1.4.0</version>
+    </dependency>
     <!-- Spring and Spring Boot dependencies -->
     <dependency>
       <groupId>org.springframework.boot</groupId>
```

Now, let's check the dependency and used version for Spring Framework using Maven dependency command:

```
mvn dependency:tree  | grep "spring-boot:jar"
[INFO] |  +- org.springframework.boot:spring-boot:jar:3.2.0:compile

mvn dependency:tree  | grep "spring-core:jar"
```

```
[INFO] |  +- org.springframework:spring-core:jar:6.1.1:compile
```

As a result, Spring Boot 3.2 with Spring Framework 6.1.1 is used for building the Petclinic app. Build it as usual with mvn clean package and start:

```
java -XX:CRaCCheckpointTo=cr -jar ./target/spring-petclinic-3.2.0-SNAPSHOT.jar
```

```
              |\      _,,,--,,_
             /,`.-'`'   ._  \-;;,_
  _______ __|,4-  ) )_   .;.(__`'-'__     ___ __    _ ___ _______
 |       | '---''(_/._)-'(_\_)   |   |   |   |  |  | |   |       |
 |    _  |    ___|_     _|       |   |   |   |   |_| |   |       | __ _ _
 |   |_| |   |___  |   | |       |   |   |   |       |   |       | \ \ \ \
 |    ___|    ___| |   | |      _|   |___|   |  _    |   |      _|  \ \ \ \
 |   |   |   |___  |   | |     |_|       |   | | |   |   |     |_    ) ) ) )
 |___|   |_______| |___| |_______|_______|___|_|  |__|___|_______|  / / / /
 ==========================================================================/_/_/_/

:: Built with Spring Boot :: 3.2.0
```

```
2023-11-24T02:28:52.989-08:00  INFO 17325 --- [           main]
o.s.s.petclinic.PetClinicApplication     : Starting PetClinicApplication
v3.2.0-SNAPSHOT using Java 21.0.1 with PID 17325 (.../target/spring-petclinic-
3.2.0-SNAPSHOT.jar started ..)
2023-11-24T02:28:52.995-08:00  INFO 17325 --- [           main]
o.s.s.petclinic.PetClinicApplication     : No active profile set, falling back
to 1 default profile: "default"
2023-11-24T02:28:53.948-08:00  INFO 17325 --- [           main]
.s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA
repositories in DEFAULT mode.
2023-11-24T02:28:53.992-08:00  INFO 17325 --- [           main]
.s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository
scanning in 37 ms. Found 2 JPA repository interfaces.
2023-11-24T02:28:54.678-08:00  INFO 17325 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080
(http)
2023-11-24T02:28:54.686-08:00  INFO 17325 --- [           main]
o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-11-24T02:28:54.686-08:00  INFO 17325 --- [           main]
o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache
Tomcat/10.1.16]
2023-11-24T02:28:54.718-08:00  INFO 17325 --- [           main]
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded
```

```
WebApplicationContext
2023-11-24T02:28:54.719-08:00  INFO 17325 --- [           main]
w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext:
initialization completed in 1658 ms
2023-11-24T02:28:55.024-08:00  INFO 17325 --- [           main]
com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2023-11-24T02:28:55.184-08:00  INFO 17325 --- [           main]
com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection
conn0: url=jdbc:h2:mem:be9e0d83-4345-497c-84ce-2df90b5740bb user=SA
2023-11-24T02:28:55.185-08:00  INFO 17325 --- [           main]
com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2023-11-24T02:28:55.330-08:00  INFO 17325 --- [           main]
o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing
PersistenceUnitInfo [name: default]
2023-11-24T02:28:55.374-08:00  INFO 17325 --- [           main]
org.hibernate.Version                    : HHH000412: Hibernate ORM core
version 6.3.1.Final
2023-11-24T02:28:55.400-08:00  INFO 17325 --- [           main]
o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache
disabled
2023-11-24T02:28:55.550-08:00  INFO 17325 --- [           main]
o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring
JPA class transformer
2023-11-24T02:28:56.297-08:00  INFO 17325 --- [           main]
o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available
(set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2023-11-24T02:28:56.299-08:00  INFO 17325 --- [           main]
j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory
for persistence unit 'default'
2023-11-24T02:28:56.544-08:00  INFO 17325 --- [           main]
o.s.d.j.r.query.QueryEnhancerFactory     : Hibernate is in classpath; If
applicable, HQL parser will be used.
2023-11-24T02:28:57.751-08:00  INFO 17325 --- [           main]
o.s.b.a.e.web.EndpointLinksResolver      : Exposing 13 endpoint(s) beneath base
path '/actuator'
2023-11-24T02:28:57.837-08:00  INFO 17325 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http)
with context path ''
2023-11-24T02:28:57.850-08:00  INFO 17325 --- [           main]
o.s.s.petclinic.PetClinicApplication     : Started PetClinicApplication in
5.204 seconds (process running for 5.568)
```

Now, we can perform the checkpoint:

```
jcmd spring-petclinic JDK.checkpoint
17325:
CR: Checkpoint ...
```

The application output is as follows:

```
2023-11-24T02:29:39.847-08:00  INFO 17325 --- [Attach Listener] jdk.crac
: Starting checkpoint
2023-11-24T02:29:39.876-08:00  INFO 17325 --- [Attach Listener]
o.s.b.j.HikariCheckpointRestoreLifecycle : Evicting Hikari connections
Killed
```

We can check that the cr directory is created and contains a lot of files:

```
ls cr
...  core-17415.img  core-17419.img  files.img      mm-17325.img
pstree.img    tty-info.img
...  core-17416.img  cppath          fs-17325.img   pagemap-17325.img
seccomp.img
...  core-17417.img  dump4.log       ids-17325.img  pages-1.img        stats-
dump
...  core-17418.img  fdinfo-2.img    inventory.img  perfdata           timens-
0.img
```

These files represent the dumped HotSpot JVM memory with all the necessary information about restoring the application.

Finally, let's restore the Petclinic app:

```
java -XX:CRaCRestoreFrom=cr

2023-11-24T02:30:59.012-08:00  WARN 17325 --- [l-1 housekeeper]
com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Thread starvation or
clock leap detected (housekeeper delta=1m33s725ms577µs950ns).
2023-11-24T02:30:59.015-08:00  INFO 17325 --- [Attach Listener]
o.s.c.support.DefaultLifecycleProcessor  : Restarting Spring-managed lifecycle
beans after JVM restore
2023-11-24T02:30:59.019-08:00  INFO 17325 --- [Attach Listener]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http)
with context path ''
2023-11-24T02:30:59.020-08:00  INFO 17325 --- [Attach Listener]
o.s.c.support.DefaultLifecycleProcessor  : Spring-managed lifecycle restart
completed (restored JVM running for 50 ms)
```

You can verify that the application is running successfully on *localhost:8080*.

As we can see in the log file, we now have extremely low startup (restoring) time — only 50 ms.

Liberica JDK
Using CRaC with Java applications

bell-soft