CRaCin` your Java application

Start so fast I want to CRIU

CRaCin` your Java application

Speaker

Dirk Fauth
Research Engineer
Eclipse Committer

ETAS GmbH
Borsigstraße 24
70469 Stuttgart

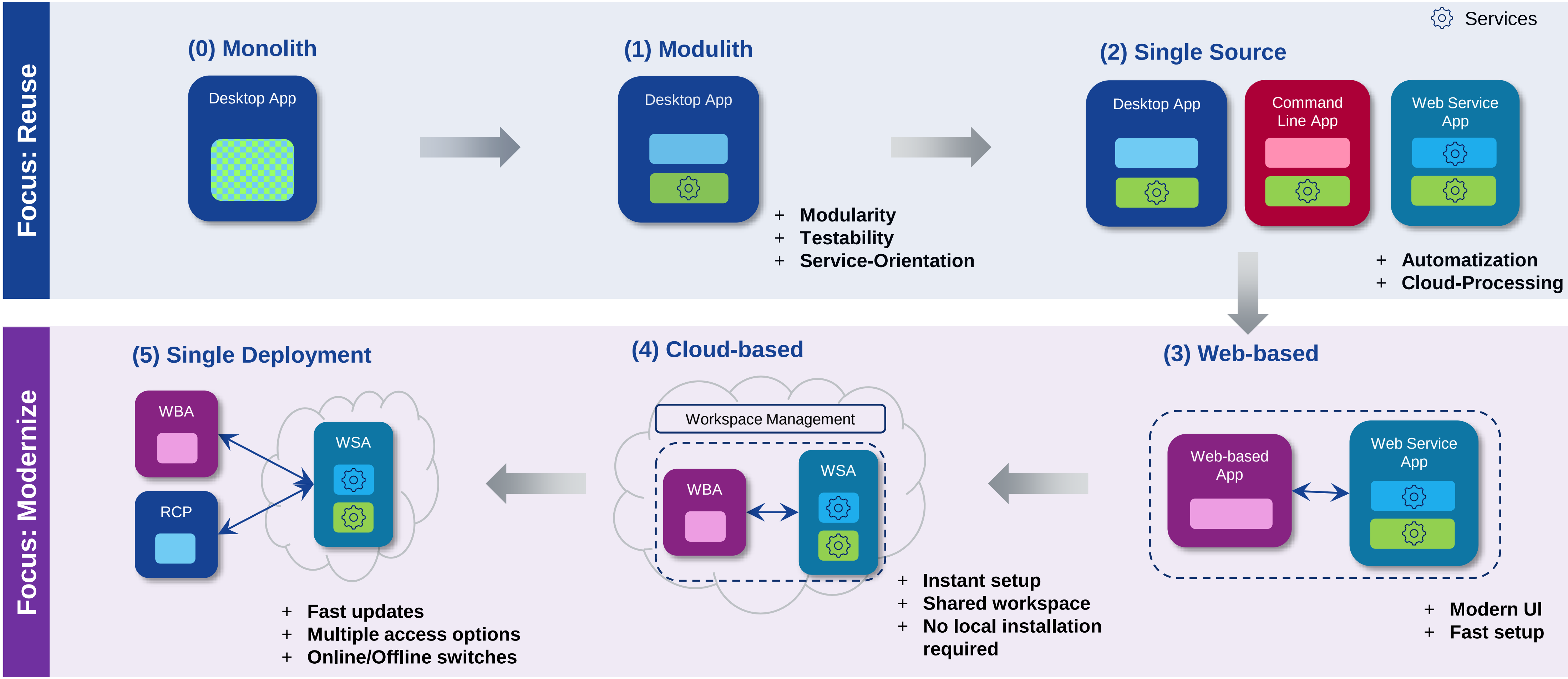
dirk.fauth@etas.com
www.etas.com

https://vogella.com/blog/
Twitter: fipro78

CRaCin` your Java application

Evolution Path

- 🟩 Business Logic
- 🟦 User Interface
- ⚙ Services



(1) Modulith

- + Modularity
- + Testability
- + Service-Orientation

(2) Single Source

- + Automatization
- + Cloud-Processing

(3) Web-based

- + Modern UI
- + Fast setup

(4) Cloud-based

- + Instant setup
- + Shared workspace
- + No local installation required

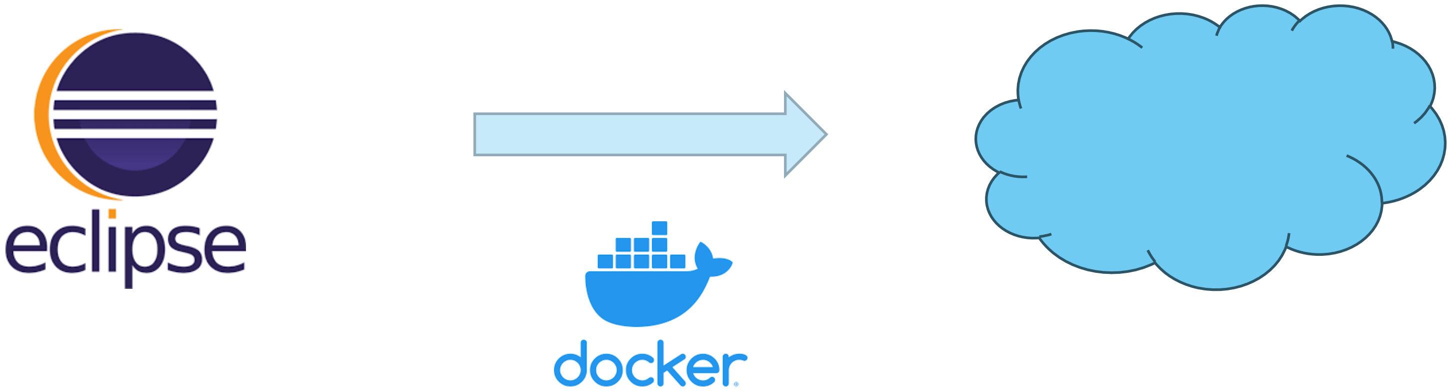
(5) Single Deployment

- + Fast updates
- + Multiple access options
- + Online/Offline switches

CRaCin` your Java application

Motivation

Shift existing Eclipse applications (partly) to the cloud



„Size of a container for a Java application to big!“

„Startup of a Java application to slow for cloud applications!“

CRaCin` your Java application

Overview

1. **Java in container**
   - Container awareness
   - jlink
   - CDS / AppCDS
   - AOT

2. **CRaC / CRIU**
   - Overview
   - OpenJDK CRaC vs OpenJ9 CRIU Support
   - Container creation process

3. **Conclusion**

Java in container

Container awareness / jlink / CDS / AppCDS / AOT

CRaCin` your Java application

Container awareness / jlink / CDS / AppCDS / AOT

- **Container awareness**
  - JVM detects when it is running inside a container, can be tuned via JVM options
  - Goal: **Avoid unexpected termination of Java process**

JVM option	Description	Default
`-XX:±UseContainerSupport`	Enable detection and runtime container configuration support	true
`-XX:ActiveProcessorCount`	CPU count that the VM should use and report as active	n/a normally determined by the number of available processors from the operating system
`-XX:InitialRAMPercentage`	Percentage of real memory used for initial heap size	1.5625
`-XX:MaxRAMPercentage`	Maximum percentage of real memory used for maximum heap size	25
`-XX:MinRAMPercentage`	Minimum percentage of real memory used for maximum heap size on systems with small physical memory	50

https://docs.oracle.com/en/java/javase/21/docs/specs/man/java.html
https://developers.redhat.com/articles/2022/04/19/java-17-whats-new-openjdks-container-awareness
https://developers.redhat.com/articles/2024/03/14/how-use-java-container-awareness-openshift-4

CRaCin` your Java application

Container awareness / jlink / CDS / AppCDS / AOT

- **jlink**
  - Create a custom JRE for your application
  - Assemble and optimize a set of modules and their dependencies into a custom runtime image
  - Goal: **Reduce JRE size**
  - JPMS required
  - Only *possible* to reduce the size of the runtime image

https://docs.oracle.com/en/java/javase/21/docs/specs/man/jlink.html

CRaCin` your Java application

Container awareness / jlink / CDS / AppCDS / AOT

- **Class Data Sharing (CDS)**
  - Create shared memory of a set of core library classes
  - Dump internal class representation into a file
  - Goal: **Reduce the startup time and memory footprint between multiple Java Virtual Machines**

- **Application Class Data Sharing (AppCDS)**
  - Extend CDS to include selected classes from the application class path

PRO	CON
Store common classes in shared drive	No optimization or hotspot detection
Reduce loading time of classes	Only reduces loading time of classes

https://docs.oracle.com/en/java/javase/21/vm/class-data-sharing.html
https://eclipse.dev/openj9/docs/shrc/

CRaCin` your Java application

Container awareness / jlink / CDS / AppCDS / AOT

- Ahead-of-time Compilation (AOT)
  - Compile Java methods into native code
  - Goal: **Reduce the startup time and memory footprint**

PRO	CON
Super-fast startup	No optimization or hotspot detection
Peak performance from first request	Closed-world assumptions
Small memory footprint	Limited use of method inlining
Lower CPU and memory usage	Reflection possible but complicated
Smaller attack surface	Development env != Deployment env

https://www.graalvm.org/latest/reference-manual/native-image/
https://eclipse.dev/openj9/docs/aot/

CRaC / CRIU

CRaCin` your Java application

CRIU

- Checkpoint/Restore In Userspace
- Linux software
- Freeze a running container/application
- Checkpoint its state to disk
- Use saved data to restore the container/application
- Run the container/application exactly as it was during the time of the freeze

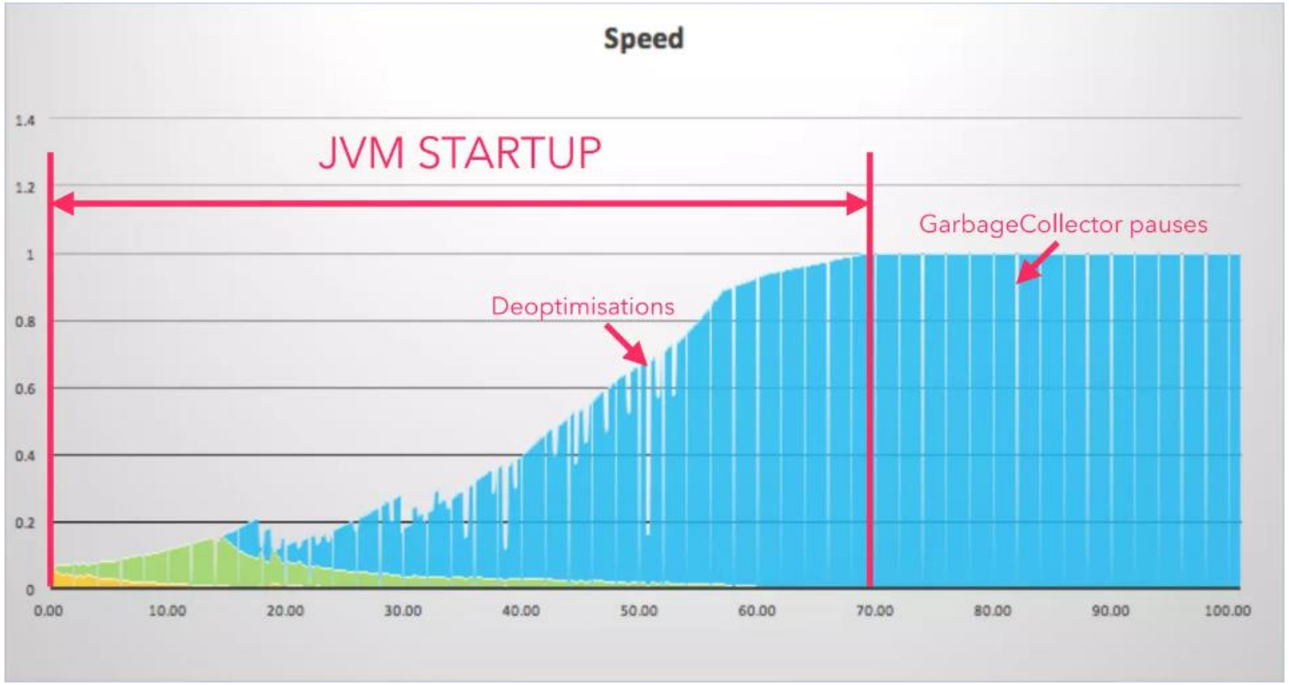
https://criu.org/Main_Page

CRaCin` your Java application

Java Application Startup

Performed everytime from scratch when starting the program

- Consists of several consecutive processes:
  - **JVM startup**
  - **Application startup**
  - **JVM warmup = Compile and Optimize Code**



2022 - Gerrit Grunwald – What the CRaC?

CRaCin` your Java application

OpenJDK CRaC vs OpenJ9 CRIU Support

	Azul Zulu OpenJDK CRaC (Coordinated Restore at Checkpoint)	**OpenJ9 CRIU Support**
Base images	Azul Zulu Ubuntu JDK CRaC `FROM azul/zulu-openjdk:21-jdk-crac-latest` *(for JRE only - build custom image)*	IBM Semeru Runtime (JDK/JRE) `FROM icr.io/appcafe/ibm-semeru-runtimes:open-21-jre-ubi-minimal`
Alpine / musl support	✓	✗
`jcmd`	✓ *(JDK only)*	✗ ✓ *(CRaC Support + JDK only)*

API	`jdk.crac`
	`org.crac`
		`org.eclipse.openj9.criu`
		`org.crac (jdk.crac)`

JVM options	`-XX:CRaCCheckpointTo={PATH}`	`-XX:+EnableCRIUSupport`
		`-XX:CRaCCheckpointTo={PATH}`

Capabilities	`CAP_CHECKPOINT_RESTORE`
	`CAP_SYS_PTRACE`
		`CAP_CHECKPOINT_RESTORE`
		`CAP_SYS_PTRACE`
		`CAP_SETPCAP`

Disable default seccomp profile	-	`--security-opt seccomp=unconfined` Necessary to grant criu access to required system calls.
PID handling	Automatic if Java process has PID 1 or via `-XX:CRaCMinPid={value}`	Manually e.g. via executing a dummy command
Restore	`-XX:CRaCRestoreFrom={PATH}`	`criu restore --unprivileged -D {PATH} --shell-job -v4`

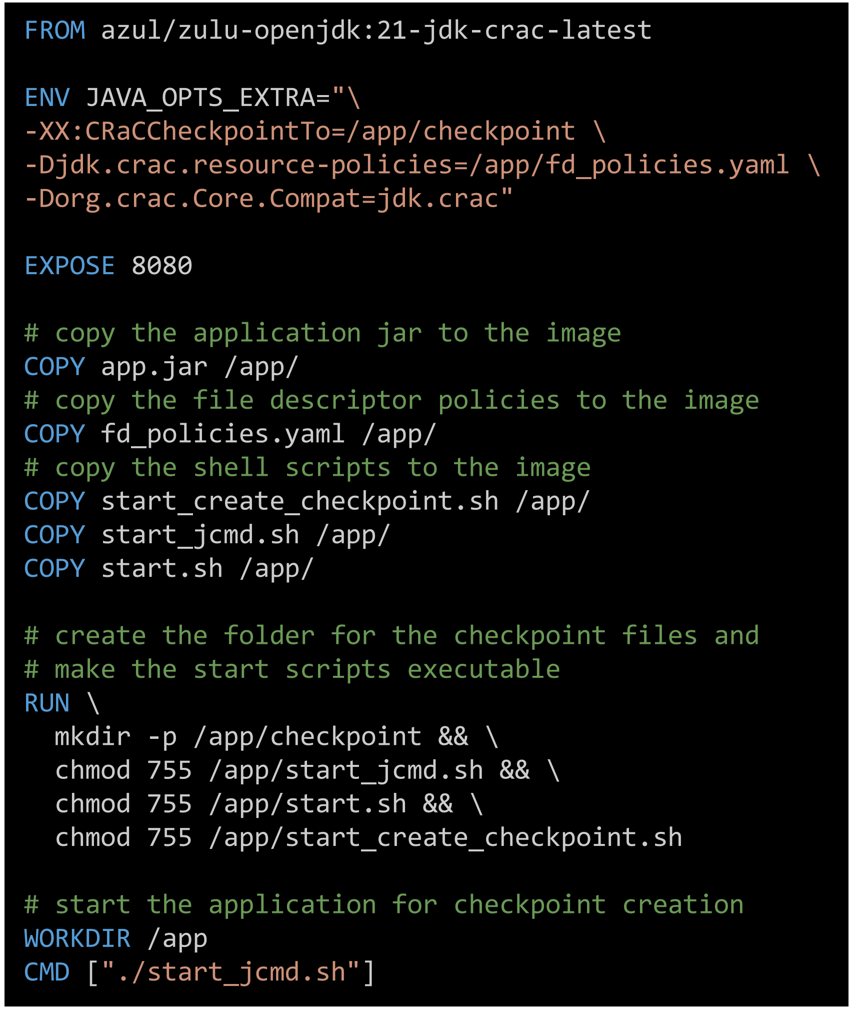
CRaC / CRIU

Container Creation Process

CRaCin` your Java application

Container Image Creation Process

1. Build the container image with the Java application



```
FROM azul/zulu-openjdk:21-jdk-crac-latest

ENV JAVA_OPTS_EXTRA="\
-XX:CRaCCheckpointTo=/app/checkpoint \
-Djdk.crac.resource-policies=/app/fd_policies.yaml \
-Dorg.crac.Core.Compat=jdk.crac"

EXPOSE 8080

# copy the application jar to the image
COPY app.jar /app/
# copy the file descriptor policies to the image
COPY fd_policies.yaml /app/
# copy the shell scripts to the image
COPY start_create_checkpoint.sh /app/
COPY start_jcmd.sh /app/
COPY start.sh /app/

# create the folder for the checkpoint files and
# make the start scripts executable
RUN \
  mkdir -p /app/checkpoint && \
  chmod 755 /app/start_jcmd.sh && \
  chmod 755 /app/start.sh && \
  chmod 755 /app/start_create_checkpoint.sh

# start the application for checkpoint creation
WORKDIR /app
CMD ["./start_jcmd.sh"]
```

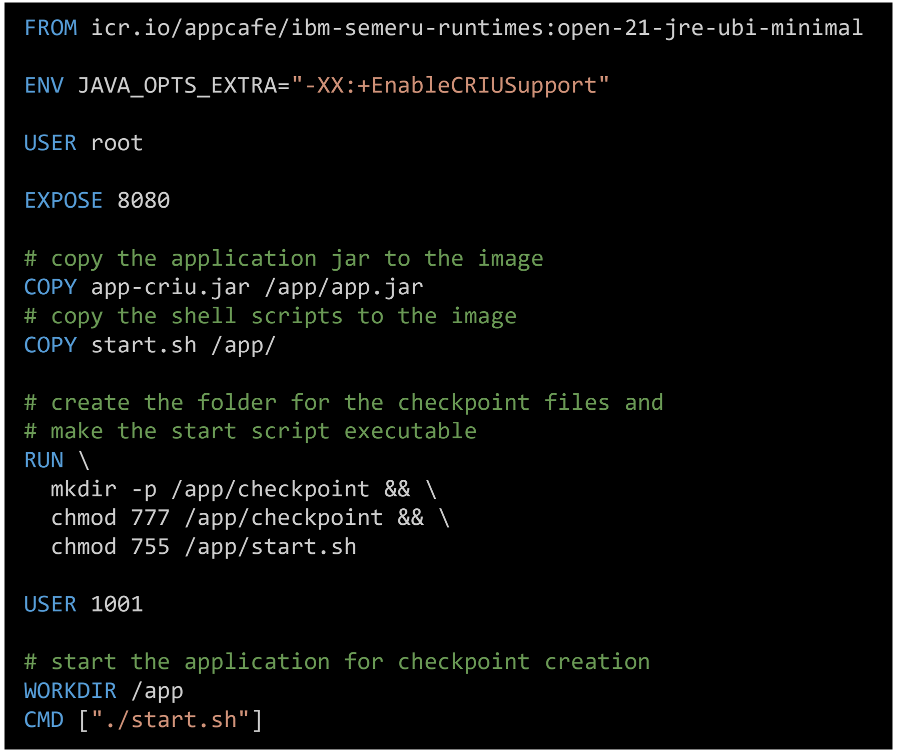
CRaCin` your Java application

Container Image Creation Process

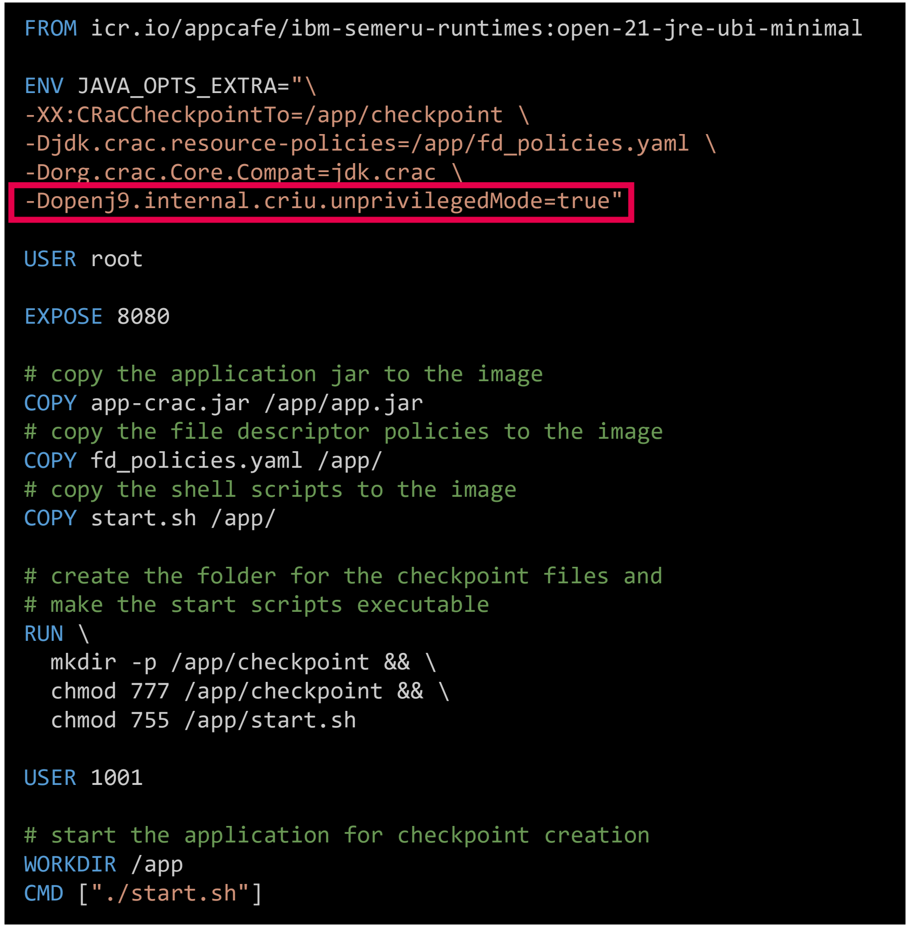
OpenJ9

org.crac API

1. Build the container image with the Java application



```
FROM icr.io/appcafe/ibm-semeru-runtimes:open-21-jre-ubi-minimal

ENV JAVA_OPTS_EXTRA="-XX:+EnableCRIUSupport"

USER root

EXPOSE 8080

# copy the application jar to the image
COPY app-criu.jar /app/app.jar
# copy the shell scripts to the image
COPY start.sh /app/

# create the folder for the checkpoint files and
# make the start script executable
RUN \
  mkdir -p /app/checkpoint && \
  chmod 777 /app/checkpoint && \
  chmod 755 /app/start.sh

USER 1001

# start the application for checkpoint creation
WORKDIR /app
CMD ["./start.sh"]
```



```
FROM icr.io/appcafe/ibm-semeru-runtimes:open-21-jre-ubi-minimal

ENV JAVA_OPTS_EXTRA="\
-XX:CRaCCheckpointTo=/app/checkpoint \
-Djdk.crac.resource-policies=/app/fd_policies.yaml \
-Dorg.crac.Core.Compat=jdk.crac \
-Dopenj9.internal.criu.unprivilegedMode=true"

USER root

EXPOSE 8080

# copy the application jar to the image
COPY app-crac.jar /app/app.jar
# copy the file descriptor policies to the image
COPY fd_policies.yaml /app/
# copy the shell scripts to the image
COPY start.sh /app/

# create the folder for the checkpoint files and
# make the start scripts executable
RUN \
  mkdir -p /app/checkpoint && \
  chmod 777 /app/checkpoint && \
  chmod 755 /app/start.sh

USER 1001

# start the application for checkpoint creation
WORKDIR /app
CMD ["./start.sh"]
```

CRaCin` your Java application

Container Image Creation Process

2. Run a container with the necessary capabilities

```
docker run \
-it \
--cap-add=CHECKPOINT_RESTORE --cap-add=SYS_PTRACE \
--name application_checkpoint \
application_checkpoint
```

```
docker run \
-it \
--cap-drop=ALL \
--cap-add=CHECKPOINT_RESTORE --cap-add=SYS_PTRACE --cap-add=SETPCAP \
--security-opt seccomp=unconfined \
--name application_checkpoint \
application_checkpoint
```

CRaCin` your Java application

Container Image Creation Process

3. Create a checkpoint

```
#!/bin/sh
. ./start_create_checkpoint.sh &
. ./start.sh
```

```
# sleep to ensure everything is ready
sleep 15
# create the checkpoint
jcmd app.jar JDK.checkpoint
```

```
#!/bin/sh

for i in $(seq 1000)
do
    /bin/true
done

java $JAVA_OPTS_EXTRA -jar app.jar
```

Alternative:
Connect to container and execute `jcmd` *manually*

org.crac API

```
Executors.newSingleThreadScheduledExecutor().schedule(() -> {
    try {
        Core.checkpointRestore();
    } catch (Exception e) {
        e.printStackTrace();
    }
},
10, TimeUnit.SECONDS);
```

OpenJ9

```
Executors.newSingleThreadScheduledExecutor().schedule(() -> {
    if (CRIUSupport.isCRIUSupportEnabled()) {
        new CRIUSupport(Paths.get("checkpoint"))
            .setLeaveRunning(false)
            .setFileLocks(true)
            .setTCPEstablished(true)
            .setLogLevel(4)
            .setUnprivileged(true)
            .checkpointJVM();
    } else {
        logger.warning("CRIU is not enabled: „
            + CRIUSupport.getErrorMessage());
    }
}, 10, TimeUnit.SECONDS);
```

CRaCin` your Java application

Container Image Creation Process

4. Create a new image from the previous one that adds the checkpoint files

OpenJDK azul

```
docker container commit \
--change='CMD ["java", "-XX:CRaCRestoreFrom=/app/checkpoint"]' \
$CONTAINER_ID \
application_restore
```

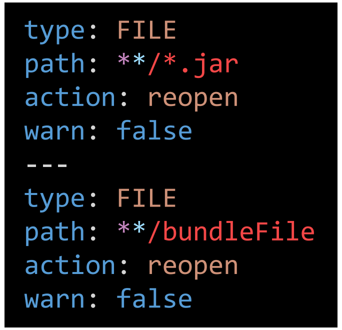
OpenJ9

```
docker container commit \
--change='CMD ["criu", "restore", "--unprivileged", "-D", "/app/checkpoint", "--shell-job", "-v4", "--log-file=restore.log"]' \
$CONTAINER_ID \
application_restore
```

No support for `–XX:CRaCRestoreFrom` *(yet)*

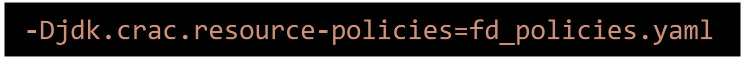
CRaCin` your Java application

Handling Open Resources



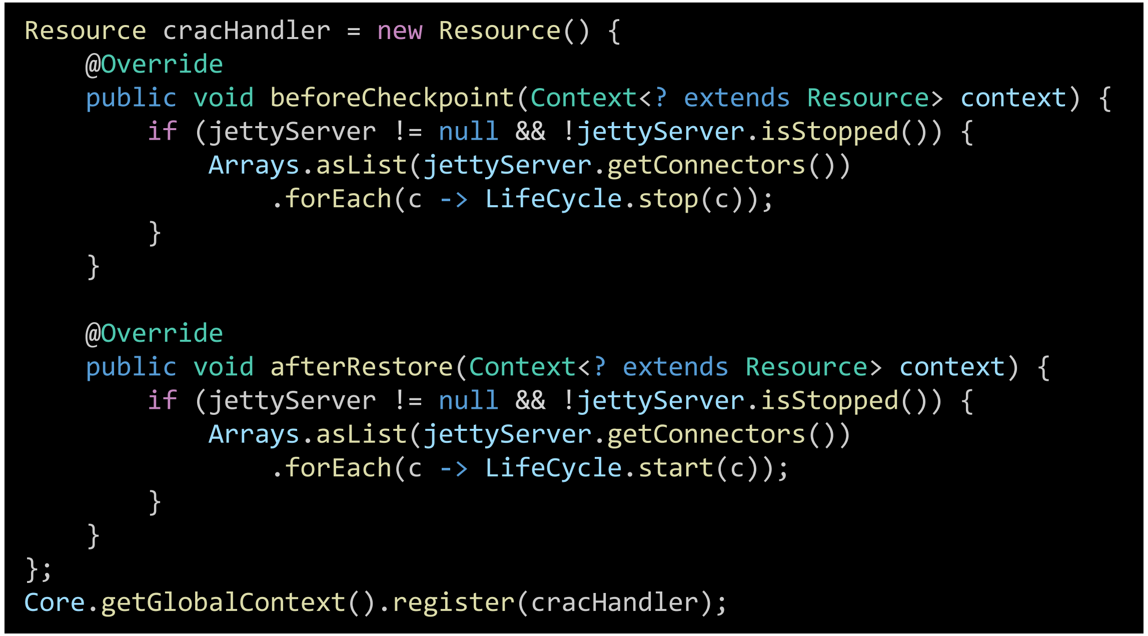
```yaml
type: FILE
path: **/*.jar
action: reopen
warn: false
---
type: FILE
path: **/bundleFile
action: reopen
warn: false
```

OpenJDK
azul



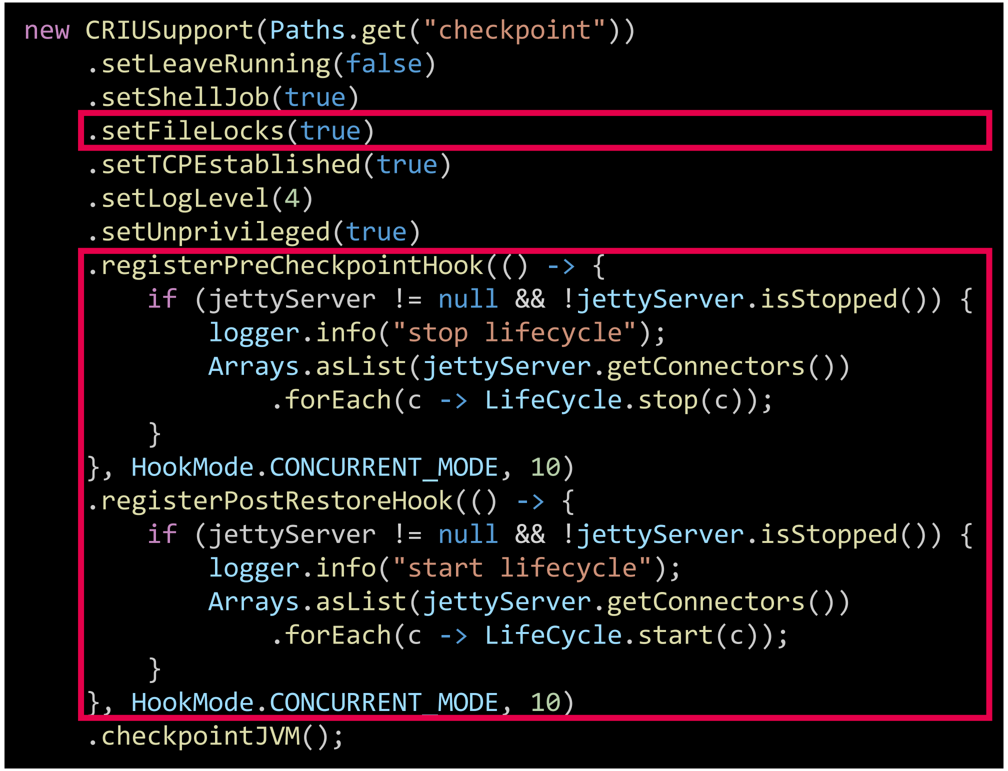
```
-Djdk.crac.resource-policies=fd_policies.yaml
```

`org.crac` API



```java
Resource cracHandler = new Resource() {
    @Override
    public void beforeCheckpoint(Context<? extends Resource> context) {
        if (jettyServer != null && !jettyServer.isStopped()) {
            Arrays.asList(jettyServer.getConnectors())
                .forEach(c -> LifeCycle.stop(c));
        }
    }

    @Override
    public void afterRestore(Context<? extends Resource> context) {
        if (jettyServer != null && !jettyServer.isStopped()) {
            Arrays.asList(jettyServer.getConnectors())
                .forEach(c -> LifeCycle.start(c));
        }
    }
};
Core.getGlobalContext().register(cracHandler);
```

OpenJ9

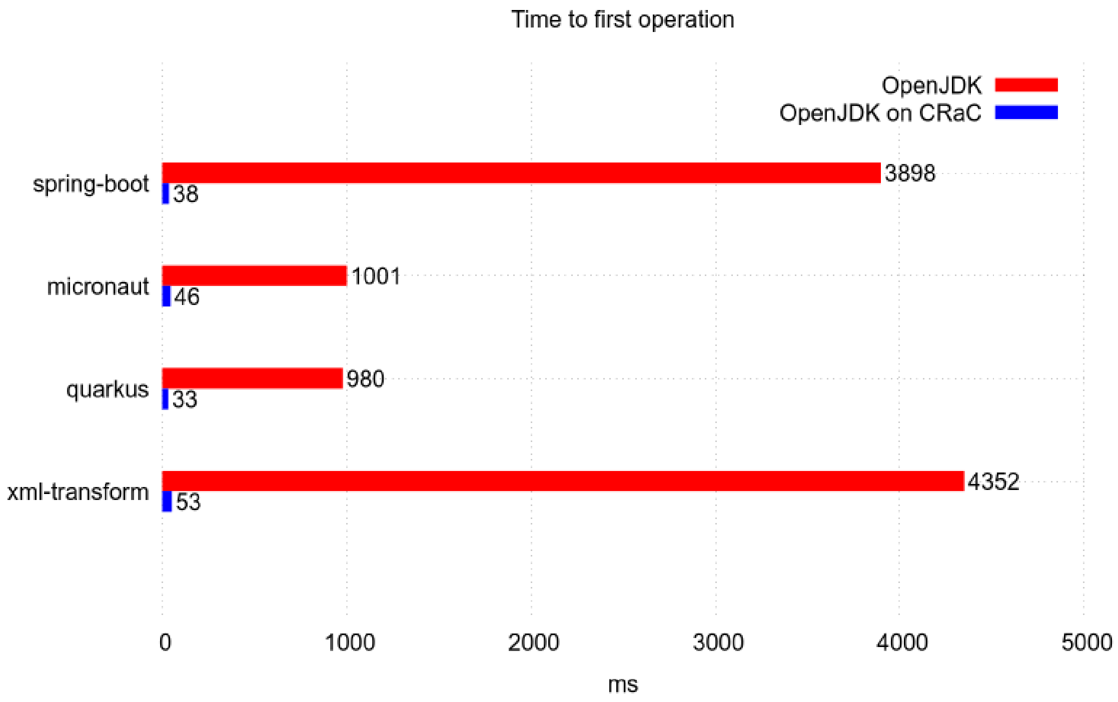


```java
new CRIUSupport(Paths.get("checkpoint"))
    .setLeaveRunning(false)
    .setShellJob(true)
    .setFileLocks(true)
    .setTCPEstablished(true)
    .setLogLevel(4)
    .setUnprivileged(true)
    .registerPreCheckpointHook(() -> {
        if (jettyServer != null && !jettyServer.isStopped()) {
            logger.info("stop lifecycle");
            Arrays.asList(jettyServer.getConnectors())
                .forEach(c -> LifeCycle.stop(c));
        }
    }, HookMode.CONCURRENT_MODE, 10)
    .registerPostRestoreHook(() -> {
        if (jettyServer != null && !jettyServer.isStopped()) {
            logger.info("start lifecycle");
            Arrays.asList(jettyServer.getConnectors())
                .forEach(c -> LifeCycle.start(c));
        }
    }, HookMode.CONCURRENT_MODE, 10)
    .checkpointJVM();
```

Conclusion

CRaCin` your Java application

Checkpoint Startup Measurement



docs.azul – What is CRaC?

`docker run` → app ready

~ 4 sec OpenJDK
~ 500 ms OpenJDK on CRaC

(for the example OSGi Jetty Jakarta-REST Whiteboard application)

CRaCin` your Java application

Checkpoint Costs

- Checkpoint image = Files on disk
- Dependent on the heap that is used by the application
- Bigger heap means bigger files

Base Image	Size w/o checkpoint	Size with checkpoint
Azul Zulu / Ubuntu / JDK	495 MB	645 MB
Azul Zulu / Ubuntu / JRE	375 MB	519 MB
Azul Zulu / Alpine / JRE	257 MB	399 MB
OpenJ9 / UBI / JDK	572 MB	661 MB
OpenJ9 / UBI / JRE	356 MB	444 MB

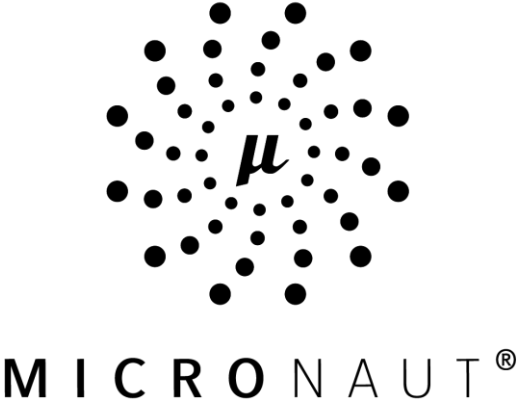
~ **130 - 150 MB** checkpoint files CRaC
~ **90 MB** checkpoint files OpenJ9

(for the example OSGi Jetty Jakarta-REST Whiteboard application)

- Save infrastructure costs with regards to processing time
- But increase costs with regards to space

CRaCin` your Java application

Frameworks & Libraries with CRaC / CRIU Support



CRaCin` your Java application

CRaC / CRIU

- Coordinated restore at checkpoint (CRaC)
  Checkpoint/Restore In Userspace (CRIU)
  - Create checkpoint from warmed up JVM and restore it from there
  - Goal: **Reduce the startup time**

PRO	CON
Super-fast startup	Larger memory footprint (runtime + checkpoint)
Peak performance from first request	Must close and reopen files, connections, sockets
Runtime optimization available	Sensitive data potentially leaked in snapshots
Lower CPU and memory usage	
Development env == Deployment env	

https://docs.azul.com/core/crac/crac-introduction
https://eclipse.dev/openj9/docs/criusupport/

CRaCin` your Java application

Further information

- CRaC Introduction @Azul
  https://docs.azul.com/core/crac/crac-introduction
- OpenJ9 CRIU Support
  https://eclipse.dev/openj9/docs/criusupport/

- What the CRaC - Superfast JVM startup (Video)
  https://www.youtube.com/watch?v=Vy1EbB2kBBs
- What the CRaC - Superfast JVM startup (Slides)
  https://de.slideshare.net/slideshow/what-the-crac-superfast-jvm-startup-252967592/252967592

- Sustainable Java Applications With Quick Warmup
  https://dzone.com/articles/sustainable-java-applications-with-quick-warmup
- Spring I/O - The Future of Java Performance in Serverless: Native Java, CRaC and Project Leyden
  https://2024.springio.net/slides/the-future-of-java-performance-in-serverless-native-java-crac-and-project-leyden-springio24.pdf

CRaCin` your Java application

Further information

https://vogella.com/blog/cracin-your-osgi-application/

https://github.com/fipro78/osgi-jakartars

Thank you

Dirk Fauth

ETAS/ENA

dirk.fauth@etas.com