FASTER STARTUP WITH SPRING BOOT AND CRaC
(Coordinated Restore at Checkpoint)

Also works with Quarkus and Micronaut!

MAGNUS LARSSON

CADEC 2025.01.23 & 2025.01.29 | CALLISTAENTERPRISE.SE

CALLISTA

https://callistaenterprise.se/blogg/

Blogg | Callista

CALLISTA

OM OSS | UPPDRAG | ERBJUDANDEN | EVENT | BLOGG | NYHETER | JOBBA HOS OSS | ENGLISH

Faster startup with Spring Boot 3.2 and CRaC, part 1 - Automatic checkpoint

01 JULY 2024 // MAGNUS LARSSON

Faster startup with Spring Boot 3.2 and CRaC, part 2 - Warmup and configuration

16 OCTOBER 2024 // MAGNUS LARSSON

Faster startup with Spring Boot 3.2 and CRaC,

part 3 - Automated build process, TBD...

AGENDA

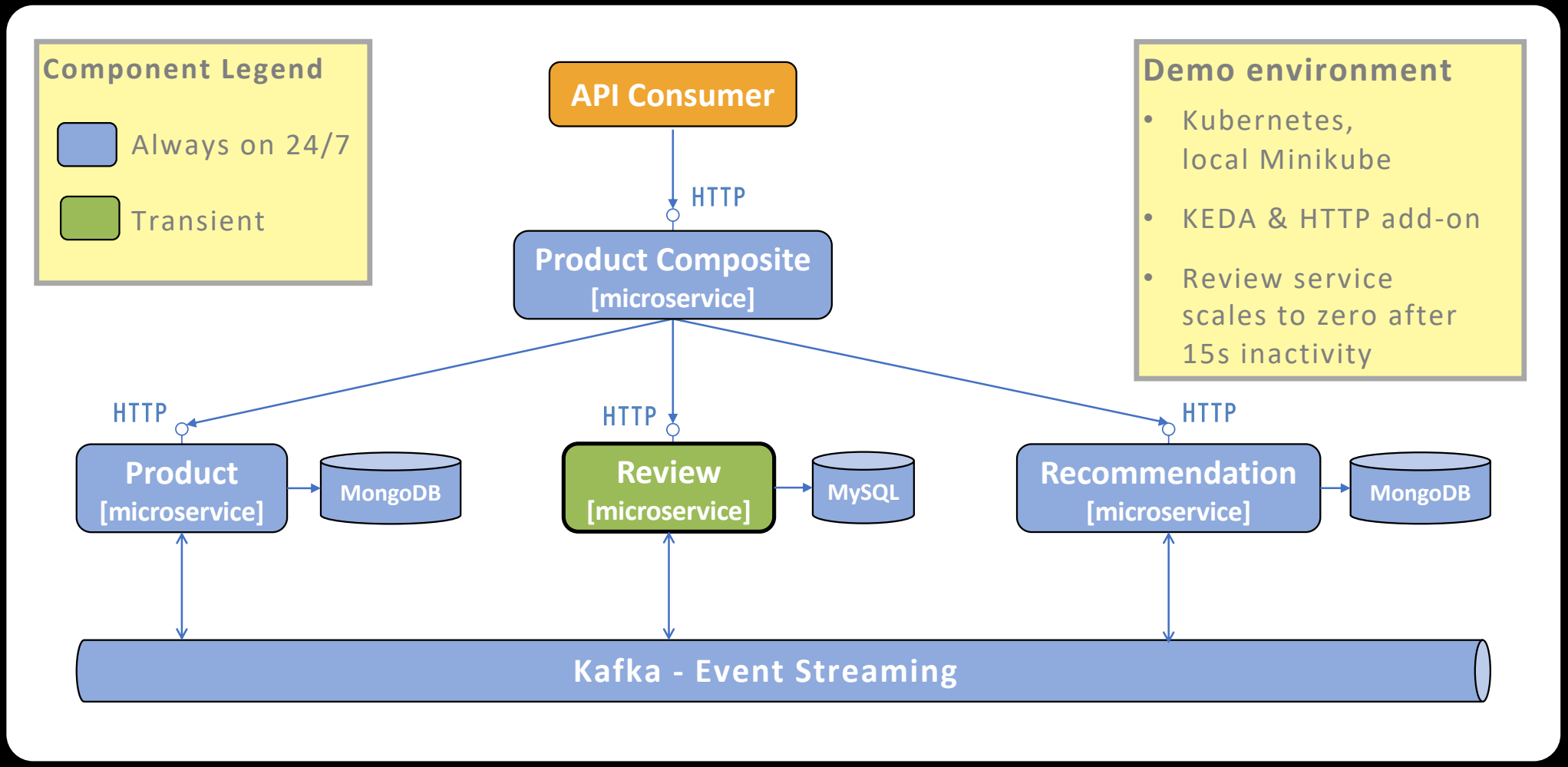
- What's the problem?
- The CRaC concept
- From concept to production
- Alternatives
- Summary

WHAT'S THE PROBLEM?

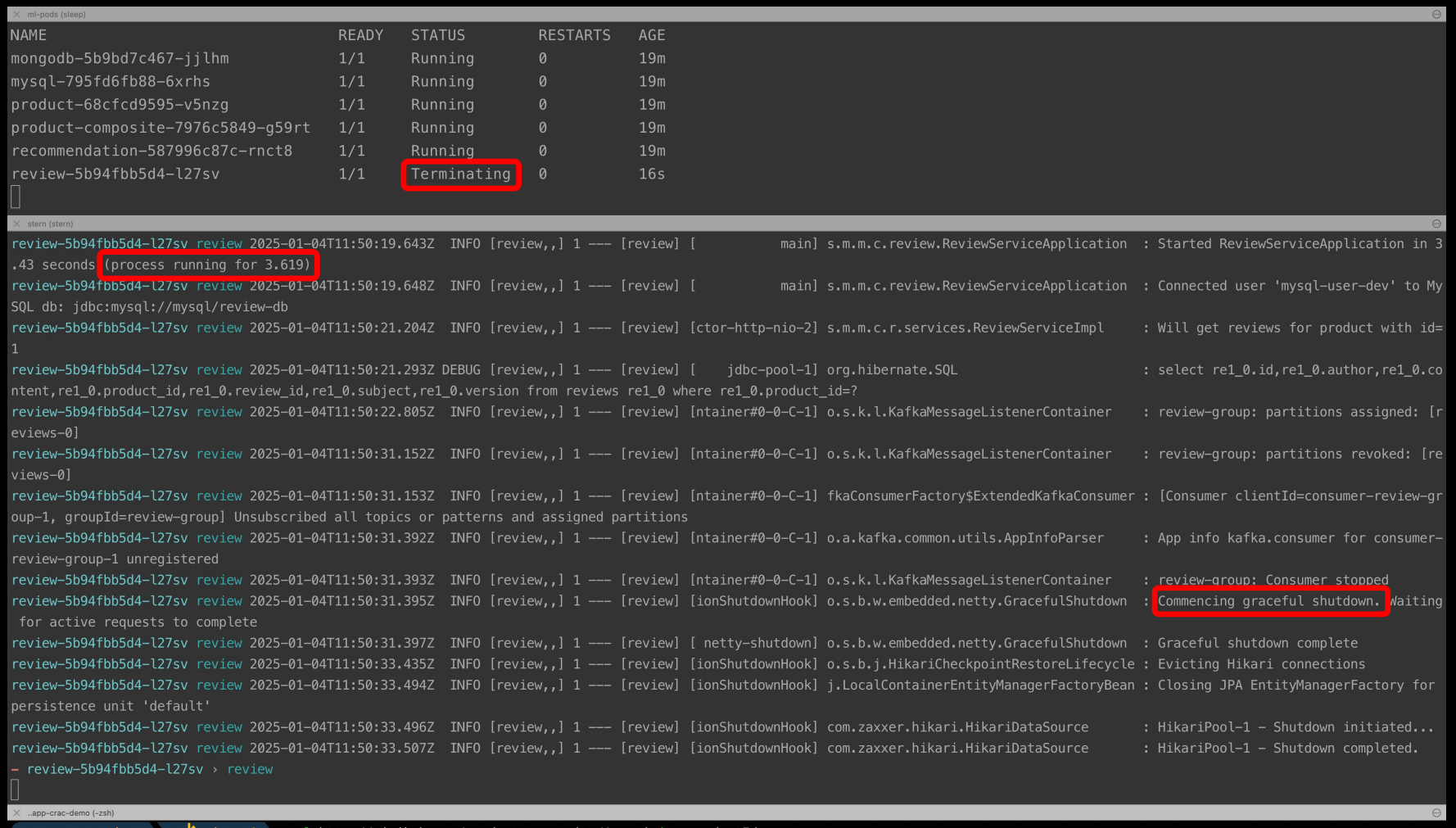
- Startup times...
  - Restarts
    - E.g. after an update or crash
  - Scaling
    - E.g. autoscaling, including scale to zero

- Demo
  - Startup a Spring Boot application
    - In cloud
    - Pay as you go
    - Transient application
    - Scale to zero

DEMO ENVIRONMENT



DEMO #1, STARTUP OF A TRADITIONAL JAVA VM APPLICATION



app-crac-demo main 1

THE CRaC CONCEPT

- CRaC is an OpenJDK project
- CRaC = Coordinated Restore at Checkpoint
- **Checkpoint**
  - Dump memory of a running (warmed up) Java app to file
- **Restore**
  - Restart the Java app by loading the file into the memory

- Demo #2
  1. Jar application startup time
  2. Make a checkpoint
  3. Restore time from checkpoint

DEMO #2, BASIC CHECKPOINT AND RESTORE

```
# Normal start of application
java -jar build/libs/hello-crac-0.0.1-SNAPSHOT.jar
   Started DemoApplication in 1.702 seconds

# Create checkpoint (automatically)
java -Dspring.context.checkpoint=onRefresh \
     -XX:CRaCCheckpointTo=checkpoint \
     -jar build/libs/hello-crac-0.0.1-SNAPSHOT.jar

# Restore application from checkpoint
java -XX:CRaCRestoreFrom=checkpoint
   Restored DemoApplication in 0.221 seconds
```

FROM CONCEPT TO PRODUCTION

1. …
2. …
3. …
4. …
5. …
6. …
7. Demo

FROM CONCEPT TO PRODUCTION

1. Only works on Linux
2. …
3. …
4. …
5. …
6. …
7. Demo

ONLY WORKS ON LINUX

- CRaC depends on a Linux feature
  - CRIU - *Checkpoint/Restore In Userspace*

- Approaches
  1. **Reusable:** Create Docker images of **CRaC-enabled applications**
  2. **Dedicated:** Infrastructure-specific solutions, e.g., AWS Lambda SnapStart

- This presentation is about creating **reusable Docker images**

- CRIU requires extra Linux capabilities
  - Checkpoint: `CHECKPOINT_RESTORE` and `SYS_PTRACE`
  - Restore: `CHECKPOINT_RESTORE`
  - Do **not** use `--privileged` in Docker or Kubernetes!

FROM CONCEPT TO PRODUCTION

1. Only works on Linux
2. Warmup
3. …
4. …
5. …
6. …
7. Demo

WARMUP

- Ensure application classes are loaded before the checkpoint
  - Execute relevant test cases
- High-performance applications might require JIT - compilation
  - When are my application classes JIT compiled?
    - Use `java -XX:+PrintCompilation`
    - Log output

```
...com.example.hello_crac.MyRestController::helloRequest (13 bytes)
```

WARMUP

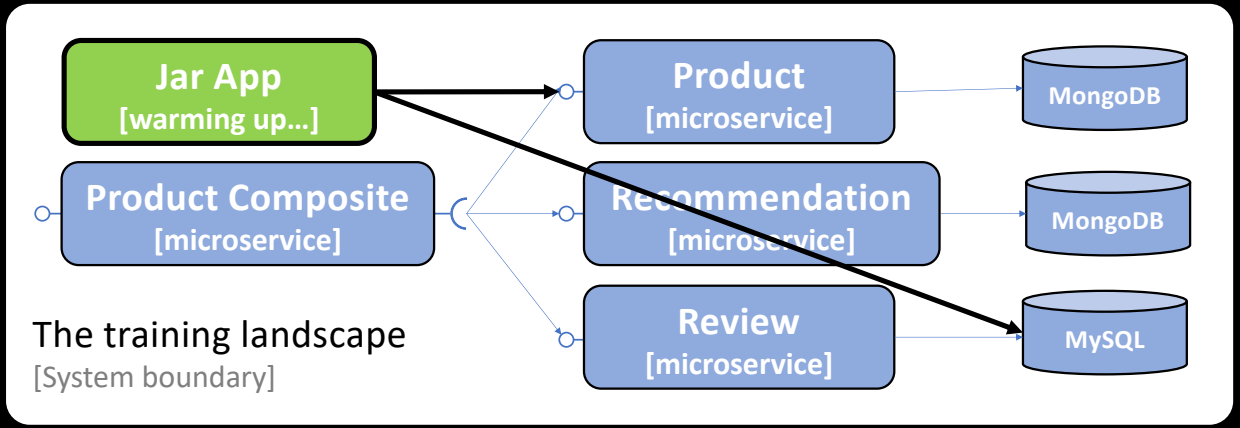
- Training environment
  - Separated from the Production environment
  - Similar in terms of external connections
    - APIs, Databases, message systems…
    - Relevant test data

  - Approaches
    - Permanent environment
    - Temporary environment
      - Docker Compose
        - Locally or in a CI pipeline

```
$ docker compose up -d
$ ./createTestdata.bash
```

Warning

All in-memory configuration will be stored in the Docker image, including sensitive information.



The training landscape
[System boundary]

FROM CONCEPT TO PRODUCTION

1. Only works on Linux
2. Warmup
3. State and connections
4. …
5. …
6. …
7. Demo

STATE AND CONNECTIONS

- Before checkpoint, open connections must be closed
  - Files, HTTP, databases, message brokers, …
- At restore, all connections must be restored
- In-memory state, e.g. caches, must be reset
- CRaC-interface for receiving checkpoint/restore notifications

```
package org.crac;

public interface Resource {
  void beforeCheckpoint(...);
  void afterRestore(...);
}
```

Handled by the 3PP libraries used for the connectivity and state management

See next slides…

FROM CONCEPT TO PRODUCTION

1. Only works on Linux
2. Warmup
3. State and connections
4. Configuration
5. …
6. …
7. Demo

CONFIGURATION

- Runtime-specific configuration must be reloaded at restore
  - Hostnames
  - Connections
  - Credentials
- **Problem:** Spring Boot application loads config at startup
- **Spring Cloud Commons** to the rescue

```
@RefreshScope
@Component
public class ProductCompositeIntegration {

  @Autowired
  public ProductCompositeIntegration(
    @Value("${app.product-service.host}") String productServiceHost,
    @Value("${app.product-service.port}") int  productServicePort,
```

CONFIGURATION

- Provided at build time: `application.yml`

```
spring:
  config.import: file:./runtime-configuration.yml
```

- Provided at restore: `runtime-configuration.yml`

```
app:
  product-service:
    host: product-prod
    port: 8080
```

FROM CONCEPT TO PRODUCTION

1. Only works on Linux
2. Warmup
3. State and connections
4. Configuration
5. 3PP libraries
6. …
7. Demo

3PP LIBRARIES

- 3PP libraries must be CRaC-aware
  - Handle external connections, state, and configuration
  - Implement the CRaC-interface `org.crac.Resource`

- Not all 3PP libraries are CRaC-friendly [yet]
  - To the rescue: spring-lifecycle-smoke-tests

3PP LIBRARIES

- State October 2024:
  - SpringDataJPA and Hibernate, see blog post #1
  - MySQL Connect, see blog post #2
  - RestTemplate and RestClient, see blog post #2
  - Spring Cloud Stream, see blog post #3
  - MongoDB Client, see blog post #2

FROM CONCEPT TO PRODUCTION

1. Only works on Linux
2. Warmup
3. State and connections
4. Configuration
5. 3PP libraries
6. **Building a CRaC image**
7. Demo

BUILDING A CRaC IMAGE

```
# Normal start of application
java -jar build/libs/hello-crac-0.0.1.jar

# Create checkpoint(automatically)
java -Dspring.context.checkpoint=onRefresh \
        -XX:CRaCCheckpointTo=checkpoint \
        -jar build/libs/hello-crac-0.0.1.jar

# Restore application from checkpoint
java -XX:CRaCRestoreFrom=checkpoint
```

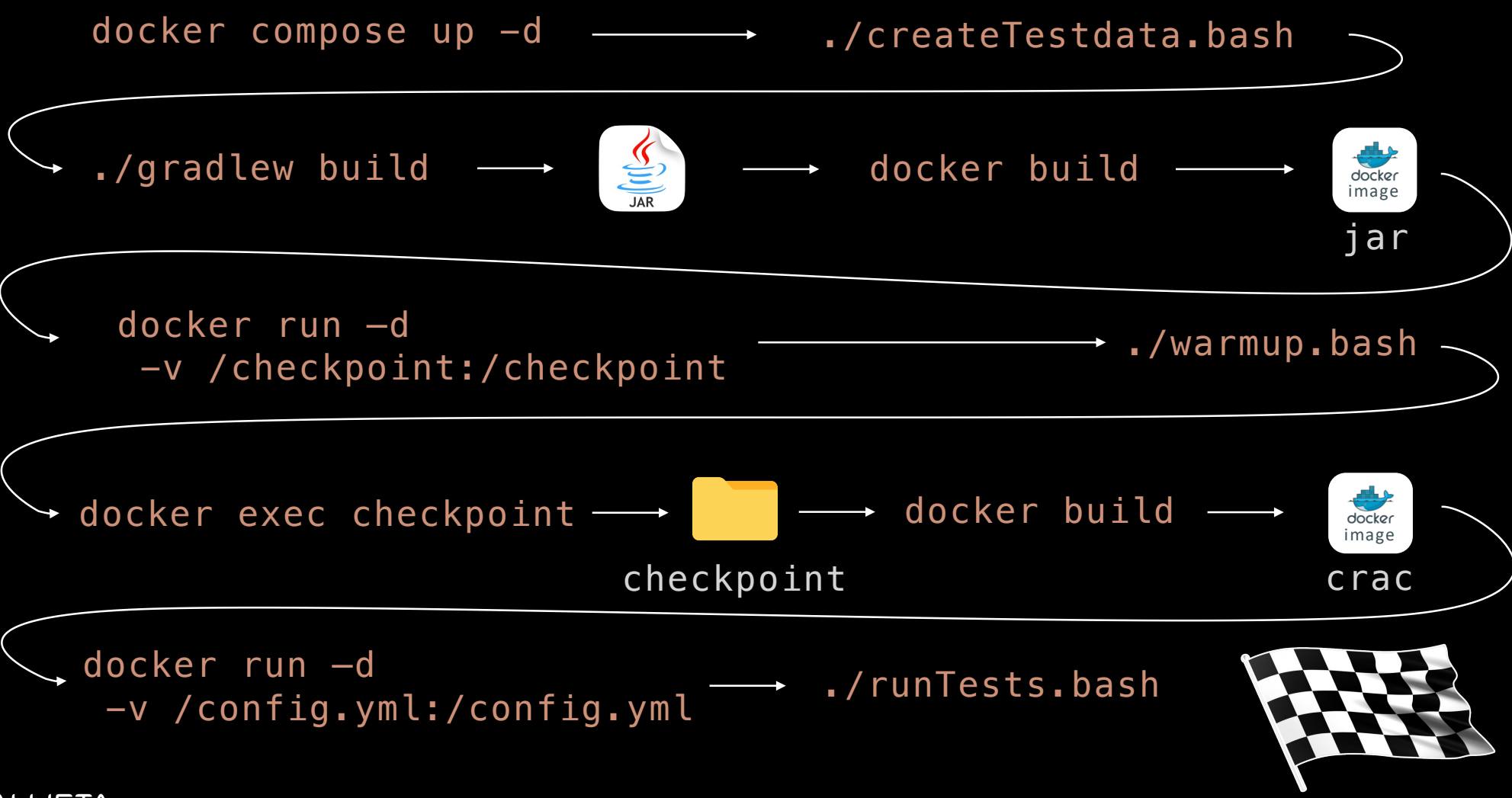
- Automated checkpoint not so useful…
  - No proper warmup
  - Configuration can't be changed at runtime
- Use on-demand checkpoints instead
  - A bit more complex…

BUILDING A CRaC IMAGE

What needs to be done:

1. Optional: Start training landscape and populate test data
2. Build jar-based application
3. Warm up the application
4. Take checkpoint
5. Package CRaC-based application
6. Test CRaC-based application

BUILDING A CRaC IMAGE



BUILDING A CRaC IMAGE - DEMO #3

magnus@MagnusMBP:~ ⌥⌘3

```
$ hello-crac/build-crac-image.bash
...
| #1: 14:18:27 - Building... |
| #2: 14:18:30 - SKIP Start training landscape and prepare testdata... |
| #3: 14:18:30 - Warmup... |
... Started DemoApplication in 1.418 seconds (process running for 1.732)
| #4: 14:18:33 - Checkpoint... |
| #5: 14:18:37 - Building CRaC image... |
| #6: 14:18:38 - Test CRaC image... |
... Spring-managed restart completed (restored JVM running for 88 ms)
| #7: 14:18:40 - CRaC image built and tested successfully! 🏁 |
...
Execution time: 12.36s

$
```

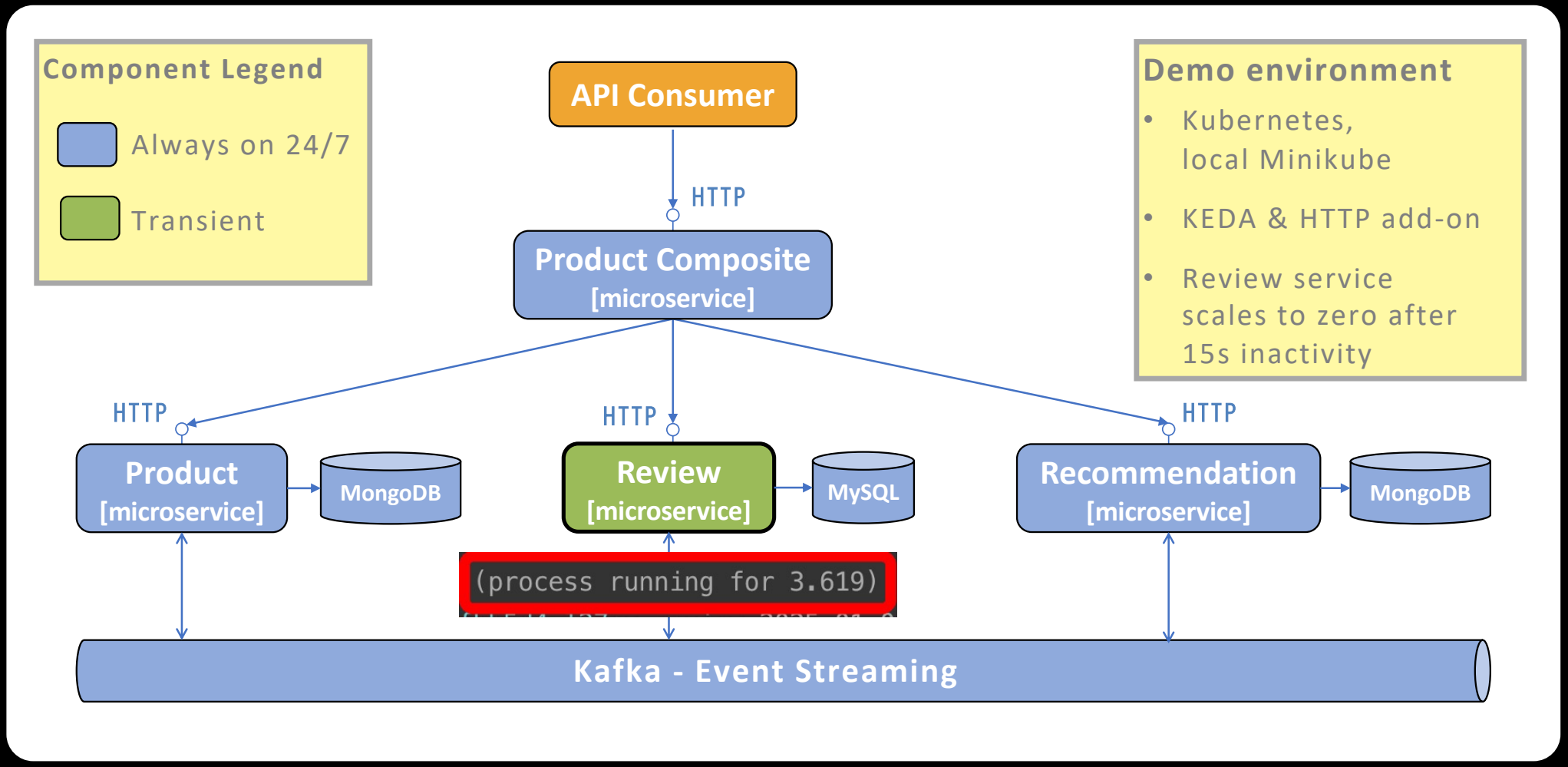
For details and source code, see:

Faster startup with Spring Boot 3.2 and CRaC,
part 3 - Automated build process, TBD...

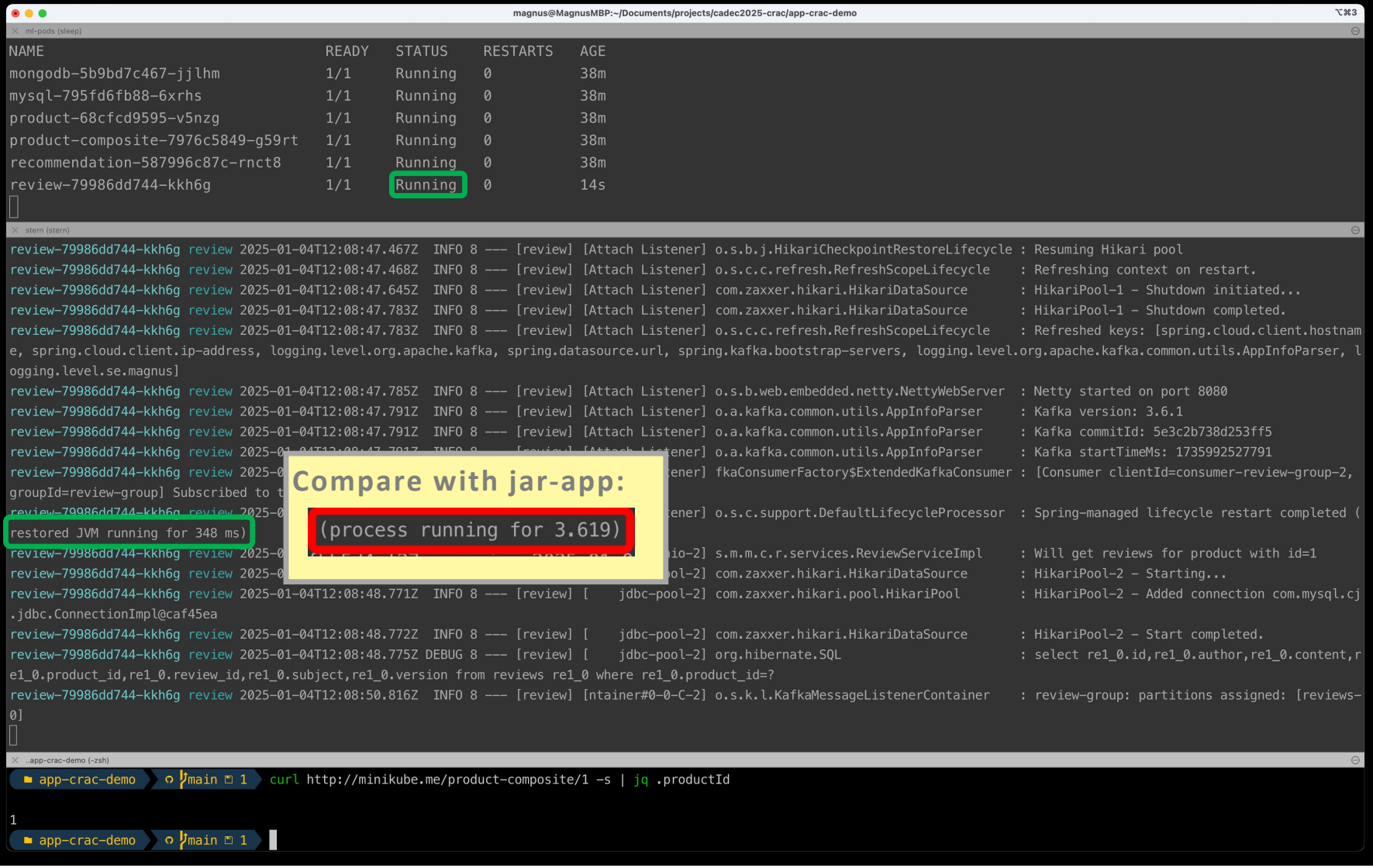
FROM CONCEPT TO PRODUCTION

1. Only works on Linux
2. Warmup
3. State and connections
4. Configuration
5. 3PP libraries
6. Building a CRaC image
7. Demo

RECAP: DEMO ENVIRONMENT



DEMO #4, STARTUP OF A CRaC'ed APPLICATION



ALTERNATIVES

Technology	Spring Boot support	Complexity	Faster Startup	More information
GraalVM native compile - Spring AOT	Spring Boot 3.0 - November 2022	Rather complex constraints on source code Looong compile times	≈20 times faster	https://callistaenterprise.se/assets/presentationer/cadec2023-spring.pdf
CRaC	Spring Boot 3.2 - November 2023	Relatively moderate code and configuration changes	≈10 times faster	https://callistaenterprise.se/blogg/teknik/2024/10/16/SpringBoot-with-CRaC-part2-on-demand-checkpoint/
App CDS Project Layden	Spring Boot 3.3 - May 2024	No impact on source code, only configuration	≈2 times faster	https://bell-sw.com/blog/how-to-use-cds-with-spring-boot-applications/

SUMMARY

- 10 times faster startup!
- Can be very useful in some scenarios
  - E.g. Scale to Zero
- Requires
  - Handling state and connections
  - Reload configuration at restore
  - An automated build process
    - Including warmup before the checkpoint
- Not all 3PP libraries are (yet) CRaC-friendly
- If CRaC is not feasible at this time, try App CDS!

WANT TO KNOW MORE?

- Blog series - Faster startup with Spring Boot and CRaC
  - Part 1 - Automatic checkpoint
  - Part 2 - Warmup and configuration
  - Part 3 - Automated build process, TBD…
    - Follow me on Linkedin to be notified!
      https://www.linkedin.com/in/magnuslarssoncallista/

- Regarding App CDS:
  - How to use CDS with Spring Boot applications

SUMMARY - QUESTIONS?

- 10 times faster startup!
- Can be very useful in some scenarios
  - E.g. Scale to Zero
- Requires
  - Handling state and connections
  - Reload configuration at restore
  - An automated build process
    - Including warmup before the checkpoint
- Not all 3PP libraries are (yet) CRaC-friendly
- If CRaC is not feasible at this time, try App CDS!