Migration Guide JDK8/11 to JDK17/21

Because it's worth it !

By
- Vaibhav Choudhary (JVM Engineer)

https://www.youtube.com/BangaloreJUG

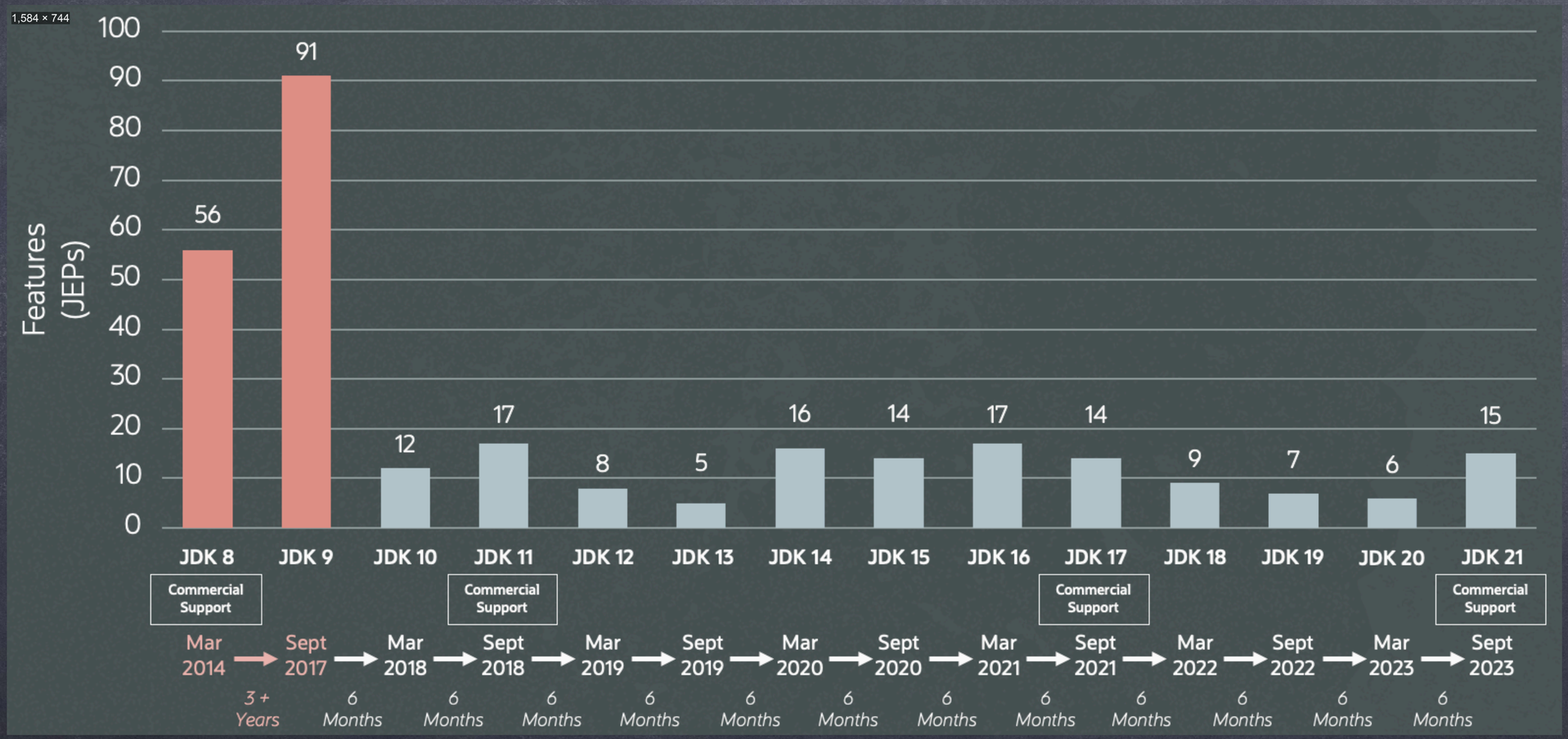
Agenda

- Why JDK21 ?
- JDK 21 vs JDK 11
- Migration issues :
  - Runtime issues
  - Compile time issues
  - 3rd party issues

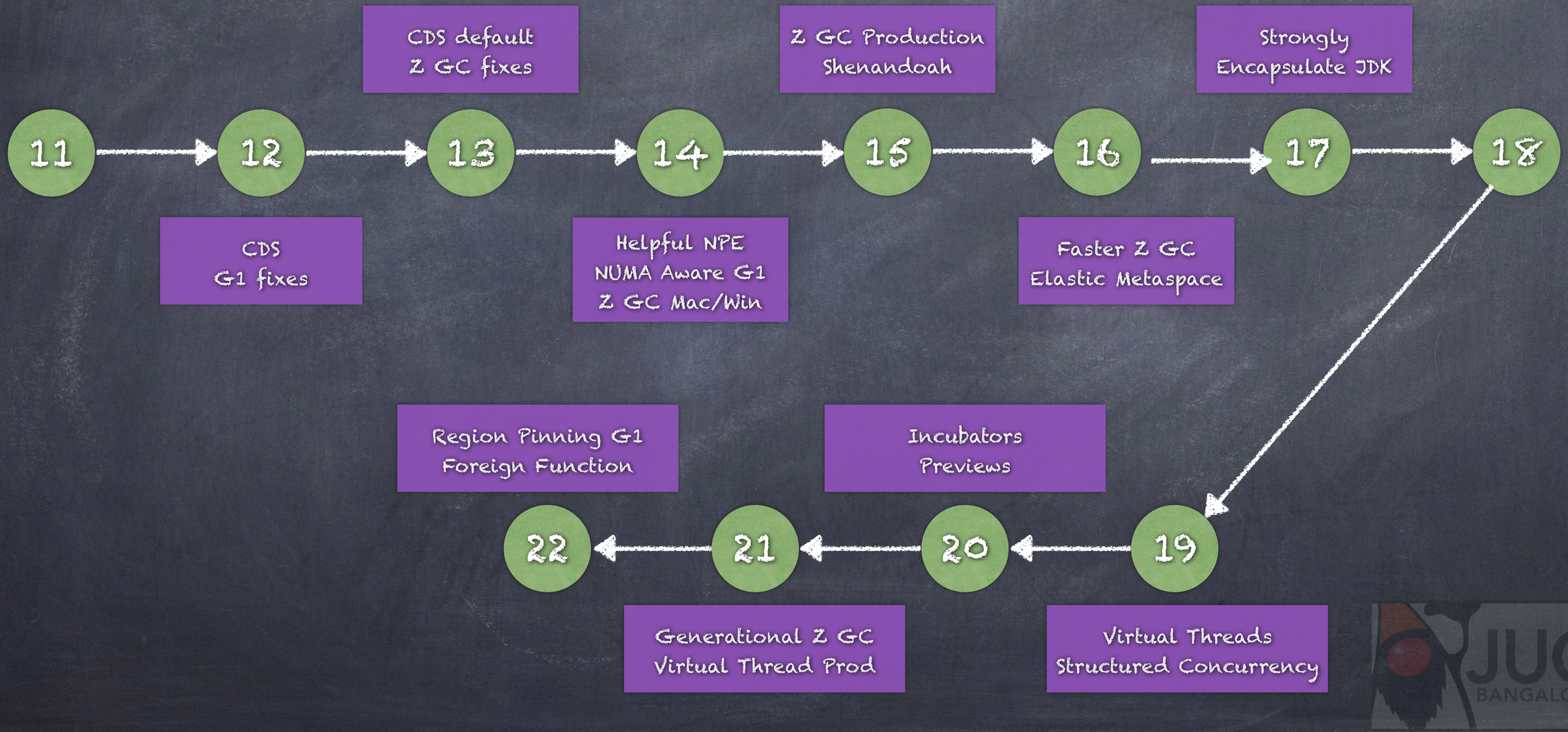
Why JDK21 ?

- JDK21 is an LTS support
- Migrating from JDK11 to JDK21, gives you so many feature and performance in free.
- Opens door for advance development for developers using Virtual threads, Records and many more.

JEP Journey JDK21 ?



Highlighted Features



JDK8 to ...

- Hi, I am on JDK8 and want to move to higher version. Shall I move to JDK11, JDK17, JDK21 or remain on JDK8 ?
- Pros and Cons - X best fit can be Y best fit !
- Depends on application size, team size, allocated time, dependent teams, and many more ...
- Daredevils can go on JDK22 too...
- Retrospection - Why I am still on JDK8 ?

Migration - Steps

Team decided to move on JDK17/21/22 from JDK11/8

Runtime or Compile Time Upgrade ?

Typically runtime first (perf gain) followed by compile time (dev gain) upgrade

Fix all the migration issues for the application including 3rd party dependencies.

Handover to perf team

Green signal from perf team. Move forward

Fix all the test failures and complete all team sign off

Rollout in production. Any last minute issues ?

Lets solve the migration issues ?

- Runtime change :

```
- java.runtime = 11.0.22
+ java.runtime = 21.0.1
```

- Maven / Bazel runs :

```
- ~~bazel build~~ / bazel run
```

```
===== APPLICATION RUN START ======
openjdk version "21.0.1" 2023-10-17
OpenJDK Runtime Environment (build 21.0.1+12-29)
OpenJDK 64-Bit Server VM (build 21.0.1+12-29, mixed mode, sharing)
…
[ERROR] <Failed>
```

Runtime issues 1 - Strongly Encapsulate

```
[Exception in thread "main" java.lang.IllegalAccessException: class
ModuleSystem cannot access class sun.invoke.util.BytecodeName (in module java.base) because
module java.base does not export sun.invoke.util to unnamed module @3196eefd
```

- JEP 396 - Do strong encapsulation for JDK internals by default.
- Till JDK11, someone can live with option `--illegal-access=warn` but not anymore after JDK16.
- Good news is that `sun.misc.Unsafe` is still available.
- Lets get into proper module usages `--add-modules` and `--export-modules`.

```
java --add-opens java.base/sun.invoke.util=ALL-UNNAMED [other args/GC args] application_launcher
```

Runtime issues 2 - Nashorn usages

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"javax.script.ScriptEngine.eval(String)" because "<local1>" is null at
NashornExample.main(NashornExample.java:10)
```

- Nashorn a JavaScript Engine has introduced in JDK8 and gained a good popularity.
- It is very challenging to maintain Nashorn Libraries and hence depreciated in JDK11, removed in JDK15 under JEP 372.
- Replacement of Nashorn in JDK17 ?

```
java -cp asm-7.3.1.jar:asm-commons-7.3.1.jar:asm-tree-7.3.1.jar:asm-util-7.3.1.jar:nashorn-
core-15.4.jar:. [other args/GC args] application_launcher

[In real world, upgrade the dependency graph]
```

Runtime issues 3 - JVM args

```
java -XX:+PrintSafepointStatistics ParallelStreamDemo.java
Unrecognized VM option 'PrintSafepointStatistics'
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.
```

- In continuous process, lot of JVM args get removed/changed from JDK11 till JDK21.
- Legacy that breaks :
  - MaxPermGen - In JDK8, PermGen has been changed by MetaSpace, but the option removed in JDK17... Insane !
  - Few changes in SafePoint Logging
  - Few GC logs / BaisedLocking removed

```
java -Xlog:safepoint+stats [other args/GC args] application_launcher
```

Runtime issues 4 - Helpful NPE

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"javax.script.ScriptEngine.eval(String)" because "<local1>" is null at
NashornExample.main(NashornExample.java:10)
```

- HelpFul NullPointerExceptions under JEP 358
- As per the doc "For example, an NPE from the assignment statement `a.i = 99;` would generate this message:

```
Exception in thread "main" java.lang.NullPointerException:
        Cannot assign field "i" because "a" is null
    at Prog.main(Prog.java:5)
```

- Break test cases (mainly), can break source as well, if code propagated with conditional check.

```
Multiple ways to fix it. Look into the demo.
e.getCause() ? In place of e.getMessage() - One way !
```

Runtime issues 5 - Locale related changes

```
public static void main(String[] args) {
    Locale hebrewOld = new Locale("iw", "IL", "");
    System.out.println(hebrewOld.getLanguage().toString().equals("iw"));
}
```

- Few changes in Locale. Carefully, if your application is heavily i18n based (which is true for most of the modern applications)
- Oracle Doc on Locale changes : https://www.oracle.com/java/technologies/javase/jdk17-suported-locales.html
- A simple demo

```
false // true with JDK11
Fix the code or use “-Djava.locale.useOldISOCodes=true” (support for backward compatibility)
```

Runtime issues ...n - Few mores

- Nanosecond precision in System clock in place of millisecond - JDK-8242504. Careful in your dashboards.
- Few print logs can have different output format.
- HTTP/2, Unicode 13, TLS 1.3 - Java 17+ supports newer technologies.

Runtime issues - 3rd party tools

- Most of the 3rd party tool needs up-gradation.
- Find out the compatible version and upgrade the dependency graph (or toolchain). Best is latest, just like your mobile phones !
- Surefire, maven, OpCodes, SpringBoot, Junit, Powermock - Everything needs higher version. All bleeding edge technologies support JDK17 (in fact JDK21 too).

Handing over to perf team

- Why performance team will see a boost !
- Is it possible that JDK migration can cause regression ? Less likely, but we are here to solve, reach to us ! I am sure, you will not, because you can do it yourself.
- Get the dashboard - Latency, Throughput, GC Pause time, Safe point, GC counts, CPU usages, Peak Performance, Start up time, Memory usages (What not !)

JDK.boost Reason 1

- CDS being default in JDK 13. Major performance gain in startup time.
- Should we see how performant a simple HelloWorld can be.

```
./java -version
openjdk version "21.0.1" 2023-10-17
OpenJDK Runtime Environment (build 21.0.1+12-29)
OpenJDK 64-Bit Server VM (build 21.0.1+12-29, mixed mode, sharing)
```

JDK.boost Reason 2

- Enormous amount of fix in Garbage Collector include G1, Z GC, Parallel GC, ~~CMS~~.
- Try to perf around Garbage collector and see the results. Have you tried Z GC ? Do your application need it ?
- Most of the modern Garbage collectors are concurrent in nature.

JDK.boost Reason 3

- Huge Runtime compiler fixes for modern hardware.
- Count the number of fixes went into C1/C2 from JDK11 till JDK21.
- Sit and enjoy, no code change at all.
- -XX:+CITime, other perf tools to find out.

JDK.boost Reason 3

- Huge Runtime compiler fixes for modern hardware.
- Count the number of fixes went into C1/C2 from JDK11 till JDK21.
- Sit and enjoy, no code change at all.
- -XX:+CITime, other perf tools to find out.

Summary

- Migrating platform languages like Java is MUST to gain and stay on top in market/customers.
- Evolution of language takes care of evolution of underlying architecture. If you have new hardware, cloud, docker, etc, you must leverage the complete benefit of language.
- Take leadership for migration
- So many success story like Netflix recent blog.

Reach to us

https://www.youtube.com/BangaloreJUG

Bangalore JUG meetup page

For any query, mail me at
vaibhav.kumar@gmail.com