Policy Team Experiences in switching to Alpine Linux and upgrading to Java 11

Yehui Wang

June 13 , 2019

Switching from Ubuntu Based to Alpine

- Why Alpine?
  - When size and time is matter .
- For comparison, Ubuntu Vs Alpine in Policy:

	With Ubuntu	With Alpine
policy-api	998 MB	376 MB
policy-apex-pdp	1.3 GB	823 MB
policy-distribution	1.1 GB	446 MB
policy-drools	1.11 GB	434 MB
policy-pap	998 MB	349 MB
policy-pe	1.6 GB	927 MB

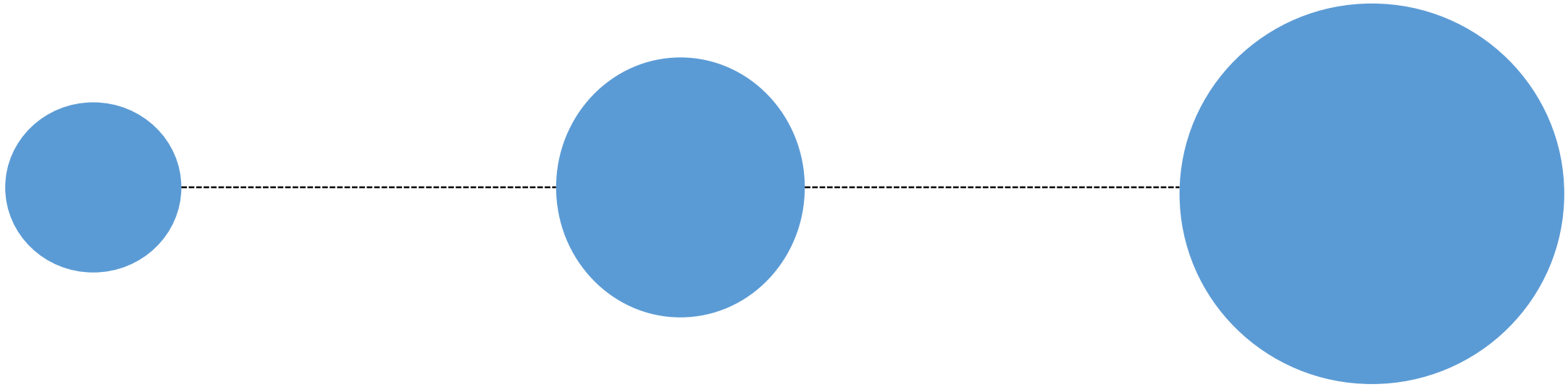
- https://docs.docker.com/samples/library/alpine/

New Release Cadence Since Java 10

Update
Every Quarter

Time-based
Feature Release
Every 6 Months

Long-term Support
Release (LTS)
Every 3 Years



Java 10 (18.3) | Java 11 (18.9) | Java 12 (19.3) | Java 13 (19.9) | Java 14 (20.3) | Java 15 (20.9) | Java 16 (21.3) | Java 17 (21.9)

New Version-String Scheme Since Java10

- The format of the new version-string is:
  - $FEATURE.$INTERIM.$UPDATE.$PATCH

```
$ java --version
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.18.04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
$
```

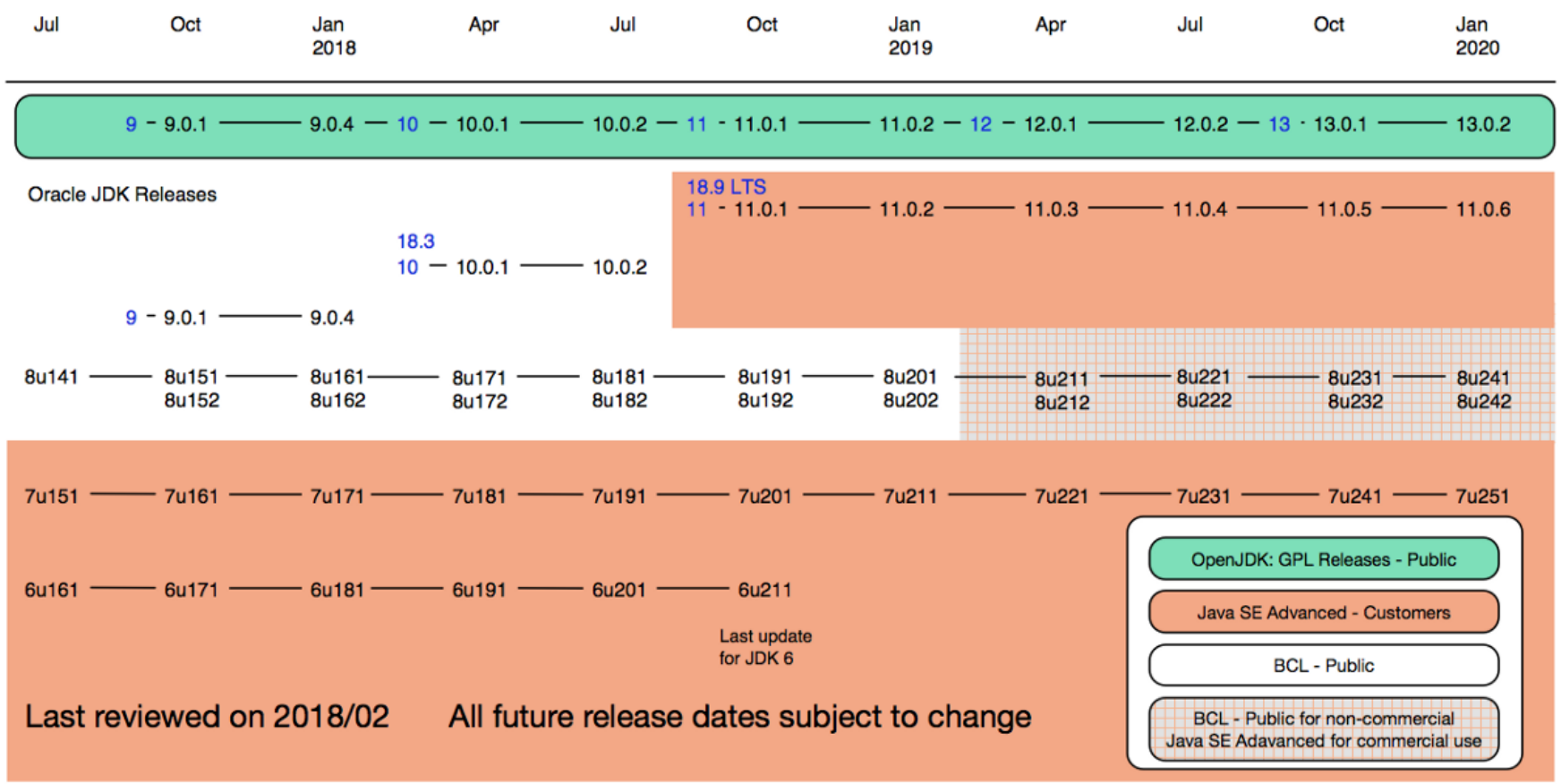
```
$ java --version
openjdk 11.0.1 2018-09-20 LTS
OpenJDK Runtime Environment (build 11+42-LTS)
OpenJDK 64-Bit Server VM (build 11+42-LTS, mixed mode)
$
```

```
$ java --version
openjdk 10.0.1 2018-04-19
OpenJDK Runtime Environment (build 10.0.1+13)
OpenJDK 64-Bit Server VM (build 10.0.1+13, mixed mode)
$
```

OpenJDK is the New Default, Oracle JDK is fully commercial

- Sun/Oracle JDK used to:
  - was richer in features
  - perceived to be more stable
  - perceived to be more performant
- As of Java 11:
  - Oracle JDK 11 and OpenJDK 11 are almost identical from a technical point of view
  - You can't use Oracle JDK 11 in production without paying Oracle from day one after its release (you can use it for development and testing)
    - From the **BCL** to the **GPL2+CPE+Commercail License**

Oracle Feature and Long Term Support Release RoadMap



Long-Term Support

- What is Long Term Support
  - Merge fixes into old JDK versions.
- What does Oracle support:
  - Free update for current OpenJDK version for 6 months.
  - Commercial support for Oracle JDK for 5+ years for Java 11, 17, 23 etc.
  - No Free LTS by Oracle
- What happens after six months if you want to stay on a specific major version while still receiving updates with security and bug fixes?
  - Rely on Operating System updates
    - On *nix platforms, you may well obtain your JDK via the operating system
  - Pay for commercial support
  - Free LTS by community for 4+years, built and shipped by Adopt OpenJDK.
  - Amazon Corretto, a GPL+CE-licensed OpenJDK build with free long-term support

Prepare for Migration

- At the beginning, you need to guarantee that your project works on Java 8 as well as on Java 11
  - Create separate Jenkins server to support Java 11 configurations
  - Using profiles in Maven for configuration specific to individual Java versions.
- Update Tools:
  - IntelliJ IDEA: 2018.2
  - Eclipse: 2018-12 (4.10)
  - Maven: 3.5.0
    - compiler plugin 3.8.0
    - surefire and failsafe: 2.22.0
  - Anything that operates on bytecode, like
    - like ASM (7.0), Byte Buddy (1.9.0), cglib (3.2.8), or Javassist (3.23.1-GA)
  - Anything that uses something that operates on bytecode like
    - Spring (5.1), Hibernate (5.4), Mockito (2.20.0)

Don’t need modules to run on Java 11

- You are not required to create modules(JPMS) to have your code run on Java 9 or later

Migrating From Java 8 To Java 11 – part 1

- Removal Of Java EE Modules, deprecated in Java 9 and removed from Java 11
  - the JavaBeans Activation Framework (JAF) in javax.activation
  - CORBA in the packages javax.activity, javax.rmi, javax.rmi.CORBA, and org.omg.*
  - the Java Transaction API (JTA) in the package javax.transaction
  - JAXB in the packages javax.xml.bind.*
  - JAX-WS in the packages javax.jws, javax.jws.soap, javax.xml.soap, and javax.xml.ws.*
  - Commons Annotation in the package javax.annotation

Migrating From Java 8 To Java 11 – part 2

- Add third-party dependencies that contain the classes you need
  - JAF: with *com.sun.activation:javax.activation*
  - CORBA: there is currently no artifact for this
  - JTA: *javax.transaction:javax.transaction-api*
  - JAXB: *com.sun.xml.bind:jaxb-impl*
  - JAX-WS: *com.sun.xml.ws:jaxws-ri*
  - Commons Annotation: *javax.annotation:javax.annotation-api*

Migrating From Java 8 To Java 11 – part 3

- Illegal Access To Internal APIs
  - One of the module system's biggest selling points is strong encapsulation. It makes sure non-public classes as well as classes from non-exported packages are inaccessible from outside the module.
  - Most com.sun.* and sun.* packages, on the other hand, are internal and hence inaccessible by default.
- What to Do?
  - Run jdeps on Your Code: jdeps -jdkinternals Sample.class
  - Fix the code by getting rid of internal API invocation.
  - Consider command line flags:
    - --add-exports
    - --add-opens

Migrating From Java 8 To Java 11 – part 4

- Removal Of Deprecated APIs and JavaFX
  - Since Java 9, the @Deprecated annotation got a Boolean attribute: forRemoval. If true, the deprecated element is going to be removed as soon as the next major release.
  - Here are some of the more common classes and methods that were removed between Java 8 and 11:
    - sun.misc.Base64 (use java.util.Base64)
    - com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel
    - (use javax.swing.plaf.nimbus.NimbusLookAndFeel)
    - on java.util.LogManager, java.util.jar.Pack200.Packer/ Unpacker:
    - methods addPropertyChangeListener and removePropertyChangeListener
    - on java.lang.Runtime: methods getLocalizedInputStream and getLocalizedOutputStream
    - various methods on SecurityManager
  - JDK 9 Release Notes – Removed APIs, Features, and Options, JDK 10 Release Notes – Removed Features and Options, JDK 11 Release Notes – Removed Features and Options
- Removed Java VisualVM

Migrating From Java 8 To Java 11 – part 5

- New Class Loader Implementations
  - (URLClassLoader) getClass().getClassLoader() or (URLClassLoader) ClassLoader.getSystemClassLoader() sequences will no longer execute
    - don't cast the application class loader to URLClassLoader
    - If you want to access the class path content, check the system property java.class.path and parse it:

```
String pathSeparator = System
    .getProperty("path.separator");
String[] classPathEntries = System
    .getProperty("java.class.path")
    .split(pathSeparator);
```

References

- Oracle JDK Migration Guide
- Oracle Java SE Support Roadmap
- Oracle JDK Releases for Java 11 and Later