Migrating from Java 8 to Java 17, especially with Spring Boot APIs, can present several challenges. Here are some key areas to watch out for:

1. **Language and API Changes**:
   * **Removed APIs**: Some APIs available in Java 8 have been removed or deprecated in Java 17. [Ensure that your code does not rely on these outdated APIs1](https://dev.to/hieuit96bk/spring-boot-3-and-java-17-migration-guide-b8f).
   * **New Language Features**: Java 17 introduces new features like records, sealed classes, and pattern matching. [While these are beneficial, they may require code refactoring2](https://www.baeldung.com/java-migrate-8-to-17).
2. **Dependency Compatibility**:
   * **Spring Boot Version**: Ensure you are using a compatible version of Spring Boot. [It’s recommended to upgrade to Spring Boot 2.7 before moving to Spring Boot 3.0, which requires Java 171](https://dev.to/hieuit96bk/spring-boot-3-and-java-17-migration-guide-b8f)[3](https://dev.to/nichetti/migrating-to-spring-boot-30-and-java-17-a-comprehensive-guide-2pbn).
   * **Third-Party Libraries**: Verify that all third-party libraries and dependencies are compatible with Java 17. [Some libraries may need to be updated to their latest versions](https://dev.to/hieuit96bk/spring-boot-3-and-java-17-migration-guide-b8f)[3](https://dev.to/nichetti/migrating-to-spring-boot-30-and-java-17-a-comprehensive-guide-2pbn).
3. **Configuration Changes**:
   * **Jakarta EE Migration**: Spring Boot 3.0 has migrated from Java EE to Jakarta EE APIs. [This means you will need to replace javax imports with jakarta1](https://dev.to/hieuit96bk/spring-boot-3-and-java-17-migration-guide-b8f).
   * [**Configuration Properties**: Use the Spring Boot Properties Migrator to help update configuration properties that may have changed between versions3](https://dev.to/nichetti/migrating-to-spring-boot-30-and-java-17-a-comprehensive-guide-2pbn).
4. **3.2. Migration Steps**
5. **3.2.1. Configuration Properties Migration**
6. Let's add the migrator by adding the following to your Maven **pom.xml**:
7. <dependency>
8. <groupId>org.springframework.boot</groupId>
9. <artifactId>spring-boot-properties-migrator</artifactId>
10. <scope>runtime</scope>
11. </dependency>
12. This will analyze your application’s environment and print diagnostics at startup console logs. Then you can based on that update your properties accordingly.
13. **Testing and Debugging**:
    * **Extensive Testing**: Migration impacts all application flows, so comprehensive testing is crucial. [Ensure that your test coverage is thorough to catch any new bugs or issues](https://dev.to/hieuit96bk/spring-boot-3-and-java-17-migration-guide-b8f)[3](https://dev.to/nichetti/migrating-to-spring-boot-30-and-java-17-a-comprehensive-guide-2pbn).
    * [**Performance Testing**: Java 17 offers performance improvements, but it’s important to test your application to ensure it performs as expected under the new runtime](https://dev.to/hieuit96bk/spring-boot-3-and-java-17-migration-guide-b8f)[2](https://www.baeldung.com/java-migrate-8-to-17).
14. **Build and Deployment**:
    * **Build Tools**: Update your build tools (e.g., Maven or Gradle) to support Java 17. [This includes updating the java.version property in your pom.xml or build.gradle](https://dev.to/hieuit96bk/spring-boot-3-and-java-17-migration-guide-b8f)[3](https://dev.to/nichetti/migrating-to-spring-boot-30-and-java-17-a-comprehensive-guide-2pbn).
    * [**GraalVM Native Image**: If you plan to use GraalVM for native images, ensure your application is compatible and test the native build thoroughly1](https://dev.to/hieuit96bk/spring-boot-3-and-java-17-migration-guide-b8f)
15. **Pre-Migration Assessment**:
    * [**Codebase Review**: Identify deprecated APIs and features in your current codebase that are no longer supported in JDK 201](https://www.unlogged.io/post/migrating-from-java-8-11-to-java-21-and-spring-boot-2-to-the-latest-spring-boot-3-2).
    * [**Dependency Audit**: Ensure all third-party libraries and dependencies are compatible with JDK 202](https://www.springcloud.io/post/2022-11/springboot3-upgrade-guide/).
16. **Upgrade Path**:
    * [**Incremental JDK Upgrades**: It’s advisable to upgrade incrementally (e.g., JDK 8 -> JDK 11 -> JDK 17 -> JDK 20) to catch issues early and make the transition smoother2](https://www.springcloud.io/post/2022-11/springboot3-upgrade-guide/).
    * **Spring Boot Version**: Upgrade to a compatible Spring Boot version. [Spring Boot 3.0 or later is recommended for JDK 17 and beyond](https://www.unlogged.io/post/migrating-from-java-8-11-to-java-21-and-spring-boot-2-to-the-latest-spring-boot-3-2)[2](https://www.springcloud.io/post/2022-11/springboot3-upgrade-guide/).
17. **Code Refactoring**:
    * [**API Changes**: Refactor code to replace deprecated APIs and utilize new language features introduced in later JDK versions1](https://www.unlogged.io/post/migrating-from-java-8-11-to-java-21-and-spring-boot-2-to-the-latest-spring-boot-3-2).
    * [**Jakarta EE Migration**: Update javax imports to jakarta if upgrading to Spring Boot 3.0 or later](https://www.unlogged.io/post/migrating-from-java-8-11-to-java-21-and-spring-boot-2-to-the-latest-spring-boot-3-2)[2](https://www.springcloud.io/post/2022-11/springboot3-upgrade-guide/).
18. **Configuration Updates**:
    * [**Configuration Properties**: Use the Spring Boot Properties Migrator to update configuration properties that may have changed2](https://www.springcloud.io/post/2022-11/springboot3-upgrade-guide/).
    * **Build Tools**: Update your build tools (e.g., Maven or Gradle) to support JDK 20. [Ensure the java.version property is set correctly in your pom.xml or build.gradle](https://www.unlogged.io/post/migrating-from-java-8-11-to-java-21-and-spring-boot-2-to-the-latest-spring-boot-3-2)[2](https://www.springcloud.io/post/2022-11/springboot3-upgrade-guide/).
19. **Testing**:
    * [**Unit and Integration Tests**: Run comprehensive tests to ensure your application behaves as expected with the new JDK](https://www.unlogged.io/post/migrating-from-java-8-11-to-java-21-and-spring-boot-2-to-the-latest-spring-boot-3-2)[2](https://www.springcloud.io/post/2022-11/springboot3-upgrade-guide/).
    * [**Performance Testing**: Conduct performance tests to verify that the application performs well under JDK 20](https://www.unlogged.io/post/migrating-from-java-8-11-to-java-21-and-spring-boot-2-to-the-latest-spring-boot-3-2)[3](https://goatswitch.ai/java-21-migration-guide-master-the-upgrade-from-java-8/).
20. **Deployment**:
    * **Environment Compatibility**: Ensure your deployment environment supports JDK 20. [Update Dockerfiles, CI/CD pipelines, and other deployment scripts accordingly](https://www.unlogged.io/post/migrating-from-java-8-11-to-java-21-and-spring-boot-2-to-the-latest-spring-boot-3-2)[2](https://www.springcloud.io/post/2022-11/springboot3-upgrade-guide/).
    * [**Monitoring and Logging**: Enhance monitoring and logging to catch any issues that may arise post-migration](https://www.unlogged.io/post/migrating-from-java-8-11-to-java-21-and-spring-boot-2-to-the-latest-spring-boot-3-2)[3](https://goatswitch.ai/java-21-migration-guide-master-the-upgrade-from-java-8/).

[To identify and handle deprecated APIs, you can use the jdeprscan tool, which helps in scanning your code for deprecated APIs and provides suggestions for replacements3](https://goatswitch.ai/java-21-migration-guide-master-the-upgrade-from-java-8/)

Migrating from JDK 8 to JDK 21 involves dealing with several deprecated APIs that have been removed over the various JDK versions. Here are some key APIs that were present in JDK 8 but are deprecated or removed in JDK 21:

1. **Applet API**:
   * java.applet.Applet
   * java.applet.AppletContext
   * java.applet.AppletStub
   * [java.applet.AudioClip1](https://docs.oracle.com/en/java/javase/21/docs/api/deprecated-list.html)
2. **Finalization Methods**:
   * java.lang.Object.finalize()
   * java.awt.Graphics.finalize()
   * [java.awt.PrintJob.finalize()1](https://docs.oracle.com/en/java/javase/21/docs/api/deprecated-list.html)
3. **Java Activation Framework**:
   * java.rmi.activation.Activatable
   * java.rmi.activation.ActivationDesc
   * java.rmi.activation.ActivationGroup
   * java.rmi.activation.ActivationGroupDesc
   * java.rmi.activation.ActivationID
   * java.rmi.activation.ActivationInstantiator
   * java.rmi.activation.ActivationMonitor
   * java.rmi.activation.ActivationSystem
   * [java.rmi.activation.Activator](https://docs.oracle.com/en/java/javase/21/docs/api/deprecated-list.html)[2](https://docs.oracle.com/en/java/javase/21/migrate/removed-apis.html)
4. **Security Manager Methods**:
   * java.lang.SecurityManager.checkAwtEventQueueAccess()
   * [java.lang.SecurityManager.checkMemberAccess(java.lang.Class, int)](https://docs.oracle.com/en/java/javase/21/docs/api/deprecated-list.html)[2](https://docs.oracle.com/en/java/javase/21/migrate/removed-apis.html)
5. **Other Deprecated APIs**:
   * java.lang.Compiler
   * javax.management.remote.rmi.RMIIIOPServerImpl
   * [java.lang.ThreadGroup.allowThreadSuspension(boolean)](https://docs.oracle.com/en/java/javase/21/docs/api/deprecated-list.html)[2](https://docs.oracle.com/en/java/javase/21/migrate/removed-apis.html)
6. JDK Migration Tooling: Tools like jdeprscan and jdeps are part of the JDK and help identify deprecated APIs and dependencies on internal APIs.