**Ques: - A production application crashes with OutOfMemoryError in metaspace. What steps would you take?**

**Ans: -** Metaspace issues can occur due to memory leaks, excessive class loading, or inadequate memory allocation.

Here are the steps to diagnose and resolve the issue:

**Immediate Actions**

1. **Restart the Application**: Restart the application to restore service. This is a temporary measure and does not address the root cause.
2. **Capture Logs and Diagnostic Data**:

* Collect application logs, JVM thread dumps, and heap dumps to analyze the state of the application before the crash.
* Use tools like jcmd, jmap, or JVM arguments like -XX:+HeapDumpOnOutOfMemoryError to generate a heap or Metaspace dump.

**Analysis Steps**

1. **Analyze the Cause of Metaspace Exhaustion**:

* Use tools like Eclipse MAT or VisualVM to inspect the heap and Metaspace dumps.
* Look for:
* **Excessive Class Loading**: Identify whether many classes are being loaded dynamically(e.g., from reflection, proxies, or libraries like Hibernate).
* **ClassLoader Leaks**: Check for ClassLoaders not being garbage collected, typically in environments like application servers (e.g., Tomcat, WebLogic).
* **Third-Party Library Issues**: Ensure libraries do not generate or retain excessive classes dynamically.

**Configuration Adjustments**

1. **Increase Metaspace Size**:

* Temporarily increase the -XX:MaxMetaspaceSize to allow more memory for class metadata.
* For example

java -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=512m -jar app.jar

* This provides a buffer while diagnosing the root cause

1. **Set Class Unloading Parameters**:

* Ensure that unused classes and class loaders are unloaded by enabling class unloading:
* -XX:+ClassUnloading -XX:+ClassUnloadingWithConcurrentGC

**Code Review and Fixes**

1. **Investigate Dynamic Class Loading**:

* Review the application code and third-party libraries for patterns that generate dynamic classes, such as:
* Reflectio
* Dynamic proxies
* Frequent usage of frameworks like Hibernate, Spring, or JasperReports
* Optimize these to reduce the number of generated class

1. **Fix ClassLoader Leaks**:

* Ensure ClassLoaders are not retained unnecessarily. Common culprits include:
* ThreadLocal variables
* Static fields
* References in application server caches
* Use tools like ClassLoader Leak Prevention Library to mitigate such issues

**Application Server-Specific Actions**

1. **Check Deployment Configuration**:

* If running in an application server, verify that deployments are correctly cleaned up.
* Ensure that hot deployments or redeployments do not leave old class loaders lingering.