I'm getting OutOfMemoryError

If your Jenkins started choking with `OutOfMemoryError`, there are four possibilities.

1. Your Jenkins is growing in data size, requiring a bigger heap space. In this case you just want to give it a bigger heap.
2. Your Jenkins is temporarily processing a large amount of data (like test reports), requring a bigger head room in memory. In this case you just want to give it a bigger heap.
3. Your Jenkins is leaking memory, in which case we need to fix that.
4. The Operating System kernel is running out of virtual memory.

Which category your `OutOfMemoryError` falls into is not always obvious, but here are a few useful techniques to diagnose the problem.

1. Use VisualVM, attach to the running instance, and observe the memory usage. Does the memory max out while loading Jenkins? If so, it probably just needs a bigger memory space. Or is it slowing creeping up? If so, maybe it is a memory leak.
2. Do you consistently see OOME around the same phase in a build? If so, maybe it just needs a bigger memory.
3. In cases where virtual memory is running short the kernel OOM (Out of Memory) killer may forcibly kill Jenkins or individual builds. If this occurs on Linux you may see builds terminate with exit code 137 (128 + signal number for SIGKILL). The dmesg command output will show log messages that will confirm the action that the kernel took.

If you think it's a memory leak, the Jenkins team needs to get the heap dump to be able to fix the problem. There are several ways to go about this.

- Run JVM with `-XX:+HeapDumpOnOutOfMemoryError` so that JVM will automatically produce a heap dump when it hits `OutOfMemoryError`.
- You can run `jmap -dump:live,file=/tmp/jenkins.hprof pid` where pid is the process ID of the target Java process.
- Use VisualVM, attach to the running instance, and obtain a heap dump
- If your Jenkins runs at `http://server/jenkins/`, request `http://server/jenkins/heapDump` with your browser and you'll get the heap dump downloaded. (1.395 and newer)
- If you are familiar with one of many Java profilers, they normally offer this capability, too.

Once you obtain the heap dump, please post it somewhere, then open an issue (or look for a duplicate issue), and attach a pointer to it. Please be aware that heap dumps may contain confidential information of various sorts.

If the full heap dump is too big, please try to get us the heap histogram (`jmap -histo:live pid`).

In the past, the distributed build support has often been a source of leakage (as this involves in a distributed garbage collection.) To check for this possibility, visit links like `http://yourserver/jenkins/computer/YOURSLAVENAME/dumpExportTable`. If this show too many objects, they may be leaks.

Analyzing the heap dump yourself

If you cannot let us inspect your heap dump, we need to ask you to diagnose the leak.

- First, find the objects with biggest retention size. Often they are various Maps, arrays, or buffers.
- Next, find the path from that object to GC root, so that you can see which Jenkins object owns those big objects.

Report the summary of those findings to the list and we'll take it from there.

Using VisualVM

Unless you already have a preferred memory profiling tool, VisualVM is recommended for analyzing heap dumps. It is a standalone version of the NetBeans profiler, distributed with the Oracle JDK.

Run `jvisualvm` and use *File » Load* and select the heap dump. In the *Classes* tab, look for a class with a suspiciously large number of instances, if not already identified by `jmap -histo`. For example, to debug a Groovy script leak, type `GroovyClassLoader` in the filter field and double-click the line with no $ in it (just `groovy.lang.GroovyClassLoader`).

In the *Instances* tab you should now see all instances. Click on some at random. (If there are more than 500, they will be broken into groups of 500, with the first expanded; so to get a representative instance “from the middle”, collapse the first group, expand a group in the middle, and select some instance from that group.)

Under *References*, right-click `this` and select *Show Nearest GC Root*. Right-click the selected item in the tree and select *Copy Path From Root*. Paste this text, for several examples, into a text file and attach it to a bug report—or continue your investigation into plugin source code.