Consideration for Large Scale Jenkins Deployment

One of the common questions for those who are planning a larger deployment is what they should expect and how they estimate the resource requirements. This page is intended to give you broad overview of how various things impact the resource requirements of Jenkins.

Factors that affect performance

Number of slaves

Each slave maintains a connection to a master, and to service these connections a master launchs several threads, and each thread is costly (2MB or so memory just for a call stack.)

Exactly how many threads each slave incurs on a master depends on the mode of connection. Most connection mechanisms require two threads per slave (SSH slaves and Java Web Start slaves), but if a JWS slave falls back to the communication over HTTP, it'll incur three. See "Obtaining a thread dump" for how to count the number of threads in your environment.

Degree of concurrent HTTP accesses to the master

If you expect a lot of users accessing Jenkins user interfaces, this adds additional CPU overheads to a master. The memory impact on additional concurrent users should be negligible.

When you expect a large number of concurrent users, watch out for the upper bound of the maximum number of HTTP request handling threads. Contrary to the intuition, you do not want to give too big a value, which tends to make contended locks even more contended. You want to keep this value relatively small, and let incoming requests wait in the queue as opposed to try to serve them all in parallel.

Number of jobs

Impact of a larger number of jobs can be seen in several places. One is the start-up time of Jenkins, especially noticable if you have $JENKINS_HOME in a disk that has a large seek time (spinning HDDs, as opposed to SSDs.) Another is the UI organization. You need to utilize views, folders, and other means to organize jobs, and avoid using the "All" view that'd render everything.

Number of builds

Starting around 1.450 to 1.500 Jenkins started lazy-loading builds and discard them from memory if unneeded, but various plugins are still being updated to take advantages of this. Therefore, you'll want to cap the number of builds each job occupies, by using the "discard old build" feature in Jenkins.

To prevent Jenkins from discarding build records only to load them back from the disk again, please give ample heap to Jenkins JVM.

One Big Jenkins master vs Multiple Jenkins masters

Two schools of thoughts exist here, when it comes to whether you want one big Jenkins master or you want many small Jenkins masters. Different people have deployed Jenkins successfully in both ways, and you have to pick one or the other depending on your situations. Here are the common trade-offs.

- Your resource utilization will be higher with One Big Master, as you can use the same set of slaves to serve a large projects and even out their spiky load. With multiple masters, each Jenkins needs to have a reasonable number of slaves.
- Installing plugins, running maintenance scripts, backing up, upgrading, and so on are easier to do with One Big Master, since you only have one place to do it.
- When each team in your organization works autonomously and very differently, Multiple Masters allow individual teams to install additional set of plugins that suit them. This would help them use Jenkins better, feel their instances "closer", and tends to care more about when it reports a failure in a commit, for example.
- If One Big Master goes down, no one can do a build, while if one of Many Masters go down, it affects a smaller number of people.