Docs

[User Manual](http://docs.google.com/userguide/userguide.html)

[Guides and Tutorials](https://guides.gradle.org)

[DSL Reference](http://docs.google.com/dsl/)

[Javadoc](http://docs.google.com/javadoc/)

[Release Notes](http://docs.google.com/release-notes.html)

[Forums](https://discuss.gradle.org/)

[Training](https://gradle.org/training/)

[Try Gradle Enterprise](https://gradle.com/enterprise)

[PDF](http://docs.google.com/userguide/userguide.pdf)

* [User Manual Home](http://docs.google.com/userguide/userguide.html)
* [Release Notes](http://docs.google.com/release-notes.html)
* [Installing Gradle](http://docs.google.com/userguide/installation.html)
* [Tutorials](https://guides.gradle.org/)

### Reference

* [Groovy DSL Reference](http://docs.google.com/dsl/)
* [Gradle API Javadoc](http://docs.google.com/javadoc/)
* [Core Plugins](http://docs.google.com/userguide/plugin_reference.html)
* [Gradle & Third-party Tools](http://docs.google.com/userguide/third_party_integration.html)

### Getting Started

* [Creating New Gradle Builds](https://guides.gradle.org/creating-new-gradle-builds/)
* [Creating Build Scans](https://guides.gradle.org/creating-build-scans/)
* [Migrating From Maven](https://guides.gradle.org/migrating-from-maven/)

### Running Gradle Builds

* [Command-Line Interface](http://docs.google.com/userguide/command_line_interface.html)
* [Customizing Execution](#gjdgxs)
  + [Configuring the Build Environment](http://docs.google.com/userguide/build_environment.html)
  + [Configuring the Gradle Daemon](http://docs.google.com/userguide/gradle_daemon.html)
  + [Initialization Scripts](http://docs.google.com/userguide/init_scripts.html)
* [Directory Layout](http://docs.google.com/userguide/directory_layout.html)
* [Executing Multi-Project Builds](http://docs.google.com/userguide/intro_multi_project_builds.html)
* [Gradle Wrapper](http://docs.google.com/userguide/gradle_wrapper.html)
* [Troubleshooting](http://docs.google.com/userguide/troubleshooting.html)
* [Using Build Scans](https://docs.gradle.com/build-scan-plugin)
* [Enabling and Configuring the Build Cache](http://docs.google.com/userguide/build_cache.html)
* [Integrating Separate Gradle Builds (Composite Builds)](http://docs.google.com/userguide/composite_builds.html)

### Authoring Gradle Builds

* [Fundamentals](#30j0zll)
  + [Introducing the Basics of Build Scripts](http://docs.google.com/userguide/tutorial_using_tasks.html)
  + [Working with Tasks](http://docs.google.com/userguide/more_about_tasks.html)
  + [Learning More About Build Scripts](http://docs.google.com/userguide/writing_build_scripts.html)
  + [Working with Files](http://docs.google.com/userguide/working_with_files.html)
  + [Creating Custom Task Types](http://docs.google.com/userguide/custom_tasks.html)
  + [Using Gradle Plugins](http://docs.google.com/userguide/plugins.html)
  + [The Standard Gradle Plugins](http://docs.google.com/userguide/standard_plugins.html)
  + [Understanding the Build Lifecycle](http://docs.google.com/userguide/build_lifecycle.html)
  + [Working with Logging](http://docs.google.com/userguide/logging.html)
  + [Configuring Multi-Project Builds](http://docs.google.com/userguide/multi_project_builds.html)
* [Best Practices](#1fob9te)
  + [Authoring Maintainable Build Scripts](http://docs.google.com/userguide/authoring_maintainable_build_scripts.html)
  + [Organizing Gradle Projects](http://docs.google.com/userguide/organizing_gradle_projects.html)
  + [Optimizing Build Performance](https://guides.gradle.org/performance/)
  + [Using the Build Cache](https://guides.gradle.org/using-build-cache/)
* [Dependency Management](#3znysh7)
  + [Introduction to Dependency Management](http://docs.google.com/userguide/introduction_dependency_management.html)
  + [Dependency Management Terminology](http://docs.google.com/userguide/dependency_management_terminology.html)
  + [Dependency Types](http://docs.google.com/userguide/dependency_types.html)
  + [Repository Types](http://docs.google.com/userguide/repository_types.html)
  + [Declaring Dependencies](http://docs.google.com/userguide/declaring_dependencies.html)
  + [Declaring Repositories](http://docs.google.com/userguide/declaring_repositories.html)
  + [Inspecting Dependencies](http://docs.google.com/userguide/inspecting_dependencies.html)
  + [Managing Dependency Configurations](http://docs.google.com/userguide/managing_dependency_configurations.html)
  + [Managing Transitive Dependencies](http://docs.google.com/userguide/managing_transitive_dependencies.html)
  + [Dependency Locking](http://docs.google.com/userguide/dependency_locking.html)
  + [Troubleshooting Dependency Resolution](http://docs.google.com/userguide/troubleshooting_dependency_resolution.html)
  + [Customizing Dependency Resolution Behavior](http://docs.google.com/userguide/customizing_dependency_resolution_behavior.html)
  + [Dependency Cache Internals](http://docs.google.com/userguide/dependency_cache.html)
  + [Working with Dependencies](http://docs.google.com/userguide/working_with_dependencies.html)
* [Publishing Artifacts](http://docs.google.com/userguide/artifact_management.html)
* [C++ Projects](#2et92p0)
  + [Building Native Software](http://docs.google.com/userguide/native_software.html)
  + [Software Model Concepts](http://docs.google.com/userguide/software_model_concepts.html)
  + [Rule-based Model Configuration](http://docs.google.com/userguide/software_model.html)
  + [Implementing Model Rules in a Plugin](http://docs.google.com/userguide/rule_source.html)
  + [Extending the Software Model](http://docs.google.com/userguide/software_model_extend.html)
* [Java Projects](#tyjcwt)
  + [Building Java & JVM projects](http://docs.google.com/userguide/building_java_projects.html)
  + [Testing Java & JVM projects](http://docs.google.com/userguide/java_testing.html)
* [Advanced Techniques](#3dy6vkm)
  + [Configuring Tasks Lazily](http://docs.google.com/userguide/lazy_configuration.html)
  + [Developing Parallel Tasks](https://guides.gradle.org/using-the-worker-api/)
  + [Testing Your Build with TestKit](http://docs.google.com/userguide/test_kit.html)
  + [Using Ant from Gradle](http://docs.google.com/userguide/ant.html)
* [Sample Gradle builds](#1t3h5sf)
  + [Groovy DSL Samples](https://github.com/gradle/gradle/tree/master/subprojects/docs/src/samples)
  + [Kotlin DSL Samples](https://github.com/gradle/kotlin-dsl/tree/master/samples)

### Extending Gradle

* [Writing Custom Plugins](http://docs.google.com/userguide/custom_plugins.html)
* [Plugin Development Guides](https://gradle.org/guides/?q=Plugin+Development)

[Edit this page](https://github.com/gradle/gradle/edit/master/subprojects/docs/src/docs/userguide/)

# The Gradle Daemon

Contents

[Why the Gradle Daemon is important for performance](#4d34og8)

[Running Daemon Status](#2s8eyo1)

[Disabling the Daemon](#17dp8vu)

[Stopping an existing Daemon](#3rdcrjn)

[FAQ](#26in1rg)

[Tools & IDEs](#lnxbz9)

[How does the Gradle Daemon make builds faster?](#35nkun2)

*A daemon is a computer program that runs as a background process, rather than being under the direct control of an interactive user.*

*— Wikipedia*

Gradle runs on the Java Virtual Machine (JVM) and uses several supporting libraries that require a non-trivial initialization time. As a result, it can sometimes seem a little slow to start. The solution to this problem is the Gradle *Daemon*: a long-lived background process that executes your builds much more quickly than would otherwise be the case. We accomplish this by avoiding the expensive bootstrapping process as well as leveraging caching, by keeping data about your project in memory. Running Gradle builds with the Daemon is no different than without. Simply configure whether you want to use it or not - everything else is handled transparently by Gradle.

[Why the Gradle Daemon is important for performance](#4d34og8)

The Daemon is a long-lived process, so not only are we able to avoid the cost of JVM startup for every build, but we are able to cache information about project structure, files, tasks, and more in memory.

The reasoning is simple: improve build speed by reusing computations from previous builds. However, the benefits are dramatic: we typically measure build times reduced by 15-75% on subsequent builds. We recommend profiling your build by using --profile to get a sense of how much impact the Gradle Daemon can have for you.

The Gradle Daemon is enabled by default starting with Gradle 3.0, so you don’t have to do anything to benefit from it.

If you run CI builds in ephemeral environments (such as containers) that do not reuse any processes, use of the Daemon will slightly decrease performance (due to caching additional information) for no benefit, and may be disabled.

[Running Daemon Status](#2s8eyo1)

To get a list of running Gradle Daemons and their statuses use the --status command.

Sample output:

PID VERSION STATUS  
 28411 3.0 IDLE  
 34247 3.0 BUSY

Currently, a given Gradle version can only connect to daemons of the same version. This means the status output will only show Daemons for the version of Gradle being invoked and not for any other versions. Future versions of Gradle will lift this constraint and will show the running Daemons for all versions of Gradle.

[Disabling the Daemon](#17dp8vu)

The Gradle Daemon is enabled by default, and we recommend always enabling it. There are several ways to disable the Daemon, but the most common one is to add the line

org.gradle.daemon=false

to the file «USER\_HOME»/.gradle/gradle.properties, where «USER\_HOME» is your home directory. That’s typically one of the following, depending on your platform:

* C:\Users\<username> (Windows Vista & 7+)
* /Users/<username> (macOS)
* /home/<username> (Linux)

If that file doesn’t exist, just create it using a text editor. You can find details of other ways to disable (and enable) the Daemon in [Daemon FAQ](#26in1rg) further down. That section also contains more detailed information on how the Daemon works.

Note that having the Daemon enabled, all your builds will take advantage of the speed boost, regardless of the version of Gradle a particular build uses.

| **💡** | Continuous integration  Since Gradle 3.0, we enable Daemon by default and recommend using it for both developers' machines and Continuous Integration servers. However, if you suspect that Daemon makes your CI builds unstable, you can disable it to use a fresh runtime for each build since the runtime is *completely* isolated from any previous builds. |
| --- | --- |

[Stopping an existing Daemon](#3rdcrjn)

As mentioned, the Daemon is a background process. You needn’t worry about a build up of Gradle processes on your machine, though. Every Daemon monitors its memory usage compared to total system memory and will stop itself if idle when available system memory is low. If you want to explicitly stop running Daemon processes for any reason, just use the command gradle --stop.

This will terminate all Daemon processes that were started with the same version of Gradle used to execute the command. If you have the Java Development Kit (JDK) installed, you can easily verify that a Daemon has stopped by running the jps command. You’ll see any running Daemons listed with the name GradleDaemon.

[FAQ](#26in1rg)

[How do I disable the Gradle Daemon?](#1ksv4uv)

There are two recommended ways to disable the Daemon persistently for an environment:

* Via environment variables: add the flag -Dorg.gradle.daemon=false to the GRADLE\_OPTS environment variable
* Via properties file: add org.gradle.daemon=false to the «GRADLE\_USER\_HOME»/gradle.properties file

| **✨** | Note, «GRADLE\_USER\_HOME» defaults to «USER\_HOME»/.gradle, where «USER\_HOME» is the home directory of the current user. This location can be configured via the -g and --gradle-user-home command line switches, as well as by the GRADLE\_USER\_HOME environment variable and org.gradle.user.home JVM system property. |
| --- | --- |

Both approaches have the same effect. Which one to use is up to personal preference. Most Gradle users choose the second option and add the entry to the user gradle.properties file.

On Windows, this command will disable the Daemon for the current user:

(if not exist "%USERPROFILE%/.gradle" mkdir "%USERPROFILE%/.gradle") && (echo. >> "%USERPROFILE%/.gradle/gradle.properties" && echo org.gradle.daemon=false >> "%USERPROFILE%/.gradle/gradle.properties")

On UNIX-like operating systems, the following Bash shell command will disable the Daemon for the current user:

mkdir -p ~/.gradle && echo "org.gradle.daemon=false" >> ~/.gradle/gradle.properties

Once the Daemon is disabled for a build environment in this way, a Gradle Daemon will not be started unless explicitly requested using the --daemon option.

The --daemon and --no-daemon command line options enable and disable usage of the Daemon for individual build invocations when using the Gradle command line interface. These command line options have the *highest* precedence when considering the build environment. Typically, it is more convenient to enable the Daemon for an environment (e.g. a user account) so that all builds use the Daemon without requiring to remember to supply the --daemon option.

[Why is there more than one Daemon process on my machine?](#44sinio)

There are several reasons why Gradle will create a new Daemon, instead of using one that is already running. The basic rule is that Gradle will start a new Daemon if there are no existing idle or compatible Daemons available. Gradle will kill any Daemon that has been idle for 3 hours or more, so you don’t have to worry about cleaning them up manually.

idle

An idle Daemon is one that is not currently executing a build or doing other useful work.

compatible

A compatible Daemon is one that can (or can be made to) meet the requirements of the requested build environment. The Java runtime used to execute the build is an example aspect of the build environment. Another example is the set of JVM system properties required by the build runtime.

Some aspects of the requested build environment may not be met by an Daemon. If the Daemon is running with a Java 7 runtime, but the requested environment calls for Java 8, then the Daemon is not compatible and another must be started. Moreover, certain properties of a Java runtime cannot be changed once the JVM has started. For example, it is not possible to change the memory allocation (e.g. -Xmx1024m), default text encoding, default locale, etc of a running JVM.

The “requested build environment” is typically constructed implicitly from aspects of the build client’s (e.g. Gradle command line client, IDE etc.) environment and explicitly via command line switches and settings. See [Build Environment](http://docs.google.com/build_environment.html#build_environment) for details on how to specify and control the build environment.

The following JVM system properties are effectively immutable. If the requested build environment requires any of these properties, with a different value than a Daemon’s JVM has for this property, the Daemon is not compatible.

* file.encoding
* user.language
* user.country
* user.variant
* java.io.tmpdir
* javax.net.ssl.keyStore
* javax.net.ssl.keyStorePassword
* javax.net.ssl.keyStoreType
* javax.net.ssl.trustStore
* javax.net.ssl.trustStorePassword
* javax.net.ssl.trustStoreType
* com.sun.management.jmxremote

The following JVM attributes, controlled by startup arguments, are also effectively immutable. The corresponding attributes of the requested build environment and the Daemon’s environment must match exactly in order for a Daemon to be compatible.

* The maximum heap size (i.e. the -Xmx JVM argument)
* The minimum heap size (i.e. the -Xms JVM argument)
* The boot classpath (i.e. the -Xbootclasspath argument)
* The “assertion” status (i.e. the -ea argument)

The required Gradle version is another aspect of the requested build environment. Daemon processes are coupled to a specific Gradle runtime. Working on multiple Gradle projects during a session that use different Gradle versions is a common reason for having more than one running Daemon process.

[How much memory does the Daemon use and can I give it more?](#2jxsxqh)

If the requested build environment does not specify a maximum heap size, the Daemon will use up to 1GB of heap. It will use the JVM’s default minimum heap size. 1GB is more than enough for most builds. Larger builds with hundreds of subprojects, lots of configuration, and source code may require, or perform better, with more memory.

To increase the amount of memory the Daemon can use, specify the appropriate flags as part of the requested build environment. Please see [Build Environment](http://docs.google.com/build_environment.html#build_environment) for details.

[How can I stop a Daemon?](#z337ya)

Daemon processes will automatically terminate themselves after 3 hours of inactivity or less. If you wish to stop a Daemon process before this, you can either kill the process via your operating system or run the gradle --stop command. The --stop switch causes Gradle to request that *all* running Daemon processes, *of the same Gradle version used to run the command*, terminate themselves.

[What can go wrong with Daemon?](#3j2qqm3)

Considerable engineering effort has gone into making the Daemon robust, transparent and unobtrusive during day to day development. However, Daemon processes can occasionally be corrupted or exhausted. A Gradle build executes arbitrary code from multiple sources. While Gradle itself is designed for and heavily tested with the Daemon, user build scripts and third party plugins can destabilize the Daemon process through defects such as memory leaks or global state corruption.

It is also possible to destabilize the Daemon (and build environment in general) by running builds that do not release resources correctly. This is a particularly poignant problem when using Microsoft Windows as it is less forgiving of programs that fail to close files after reading or writing.

Gradle actively monitors heap usage and attempts to detect when a leak is starting to exhaust the available heap space in the daemon. When it detects a problem, the Gradle daemon will finish the currently running build and proactively restart the daemon on the next build. This monitoring is enabled by default, but can be disabled by setting the org.gradle.daemon.performance.enable-monitoring system property to false.

If it is suspected that the Daemon process has become unstable, it can simply be killed. Recall that the --no-daemon switch can be specified for a build to prevent use of the Daemon. This can be useful to diagnose whether or not the Daemon is actually the culprit of a problem.

[Tools & IDEs](#lnxbz9)

The [Gradle Tooling API](http://docs.google.com/embedding.html#embedding) that is used by IDEs and other tools to integrate with Gradle *always* uses the Gradle Daemon to execute builds. If you are executing Gradle builds from within your IDE you are using the Gradle Daemon and do not need to enable it for your environment.

[How does the Gradle Daemon make builds faster?](#35nkun2)

The Gradle Daemon is a *long lived* build process. In between builds it waits idly for the next build. This has the obvious benefit of only requiring Gradle to be loaded into memory once for multiple builds, as opposed to once for each build. This in itself is a significant performance optimization, but that’s not where it stops.

A significant part of the story for modern JVM performance is runtime code optimization. For example, HotSpot (the JVM implementation provided by Oracle and used as the basis of OpenJDK) applies optimization to code while it is running. The optimization is progressive and not instantaneous. That is, the code is progressively optimized during execution which means that subsequent builds can be faster purely due to this optimization process. Experiments with HotSpot have shown that it takes somewhere between 5 and 10 builds for optimization to stabilize. The difference in perceived build time between the first build and the 10th for a Daemon can be quite dramatic.

The Daemon also allows more effective in memory caching across builds. For example, the classes needed by the build (e.g. plugins, build scripts) can be held in memory between builds. Similarly, Gradle can maintain in-memory caches of build data such as the hashes of task inputs and outputs, used for incremental building.

Docs

* [User Manual](http://docs.google.com/userguide/userguide.html)
* [DSL Reference](http://docs.google.com/dsl/)
* [Release Notes](http://docs.google.com/release-notes.html)
* [Javadoc](http://docs.google.com/javadoc/)

News

* [Blog](https://blog.gradle.org/)
* [Newsletter](https://newsletter.gradle.com/)
* [Twitter](https://twitter.com/gradle)

Products

* [Build Scans](https://gradle.com/build-scans)
* [Build Cache](https://gradle.com/build-cache)
* [Enterprise Docs](https://gradle.com/enterprise/resources)

Get Help

* [Forums](https://discuss.gradle.org/c/help-discuss)
* [GitHub](https://github.com/gradle/)
* [Training](https://gradle.org/training/)
* [Services](https://gradle.org/services/)

Subscribe for important Gradle updates and news

Subscribe

By entering your email, you agree to our [Terms](https://gradle.org/terms/) and [Privacy Policy](https://gradle.org/privacy/), including receipt of emails. You can unsubscribe at any time.

© [Gradle Inc.](https://gradle.com) 2018 All rights reserved.

[Careers](https://gradle.com/careers) | [Privacy](https://gradle.org/privacy) | [Terms of Service](https://gradle.org/terms) | [Contact](https://gradle.org/contact/)