Build garbage collection

- Garbage collection policies
- Configuration
  - Docker daemon configuration file
    - Custom GC policies in the Docker daemon configuration file
  - BuildKit configuration file
    - Custom GC policies in BuildKit

While `docker builder prune` or `docker buildx prune` commands run at once, Garbage Collection (GC) runs periodically and follows an ordered list of prune policies. The BuildKit daemon clears the build cache when the cache size becomes too big, or when the cache age expires.

For most users, the default GC behavior is sufficient and doesn't require any intervention. Advanced users, particularly those working with large-scale builds, self-managed builders, or constrained storage environments, might benefit from customizing these settings to better align with their workflow needs. The following sections explain how GC works and provide guidance on tailoring its behavior through custom configuration.

Garbage collection policies

GC policies define a set of rules that determine how the build cache is managed and cleaned up. These policies include criteria for when to remove cache entries, such as the age of the cache, the amount of space being used, and the type of cache records to prune.

Each GC policy is evaluated in sequence, starting with the most specific criteria, and proceeds to broader rules if previous policies do not free up enough cache. This lets BuildKit prioritize cache entries, preserving the most valuable cache while ensuring the system maintains performance and availability.

For example, say you have the following GC policies:

1. Find "stale" cache records that haven't been used in the past 48 hours, and delete records until there's maximum 5GB of "stale" cache left.

2. If the build cache size exceeds 10GB, delete records until the total cache size is no more than 10GB.

The first rule is more specific, prioritizing stale cache records and setting a lower limit for a less valuable type of cache. The second rule imposes a higher hard limit that applies to any type of cache records. With these policies, if you have 11GB worth of build cache, where:

- 7GB of which is "stale" cache
- 4GB is other, more valuable cache

A GC sweep would delete 2GB of stale cache as part of the 1st policy, leaving 9GB of cache in total, meaning the 2nd policy does not need to clear any more cache.

The default GC policies are (approximately):

1. Remove cache that can be regenerated, such as build contexts from local directories or remote Git repositories, and cache mounts, if hasn't been used for more than 48 hours.
2. Remove cache that hasn't been used in a build for more than 60 days.
3. Remove unshared cache that exceeds the build cache size limit. Unshared cache records refers to layer blobs that are not used by other resources (typically, as image layers).
4. Remove any build cache that exceeds the build cache size limit.

The precise algorithm and the means of configuring the policies differ slightly depending on what kind of builder you're using. Refer to Configuration for more details.

Configuration

> [!NOTE] If you're satisfied with the default garbage collection behavior and don't need to fine-tune its settings, you can skip this section. Default configurations work well for most use cases and require no additional setup.

Depending on the type of build driver you use, you will use different configuration files to change the builder's GC settings:

- If you use the default builder for Docker Engine (the `docker` driver), use the Docker daemon configuration file.
- If you use a custom builder, use a BuildKit configuration file.

Docker daemon configuration file

If you're using the default `docker` driver, GC is configured in the `daemon.json` configuration file, or if you use Docker Desktop, in **Settings > Docker Engine**.

The following snippet shows the default builder configuration for the `docker` driver for Docker Desktop users:

```
{
  "builder": {
    "gc": {
      "defaultKeepStorage": "20GB",
      "enabled": true
    }
  }
}
```

The `defaultKeepStorage` option configures the size limit of the build cache, which influences the GC policies. The default policies for the `docker` driver work as follows:

1. Remove ephemeral, unused build cache older than 48 hours if it exceeds 13.8% of `defaultKeepStorage`, or at minimum 512MB.
2. Remove unused build cache older than 60 days.
3. Remove unshared build cache that exceeds the `defaultKeepStorage` limit.
4. Remove any build cache that exceeds the `defaultKeepStorage` limit.

Given the Docker Desktop default value for `defaultKeepStorage` of 20GB, the default GC policies resolve to:

```
{
  "builder": {
    "gc": {
      "enabled": true,
      "policy": [
        {
          "reservedSpace": "2.764GB",
          "keepDuration": "48h",
          "filter": [
            "type=source.local,type=exec.cachemount,type=source.git.checkout"
          ]
        },
        { "reservedSpace": "20GB", "keepDuration": ["1440h"] },
        { "reservedSpace": "20GB" },
        { "reservedSpace": "20GB", "all": true }
```

```
      ]
    }
  }
}
```

The easiest way to tweak the build cache configuration for the `docker` driver is to adjust the `defaultKeepStorage` option:

- Increase the limit if you feel like you think the GC is too aggressive.
- Decrease the limit if you need to preserve space.

Custom GC policies in the Docker daemon configuration file

If you need even more control, you can define your own GC policies directly. The following example defines a more conservative GC configuration with the following policies:

1. Remove unused cache entries older than 1440 hours, or 60 days, if build cache exceeds 50GB.
2. Remove unshared cache entries if build cache exceeds 50GB.
3. Remove any cache entries if build cache exceeds 100GB.

```
{
  "builder": {
    "gc": {
      "enabled": true,
      "policy": [
        { "reservedSpace": "50GB", "keepDuration": ["1440h"] },
        { "reservedSpace": "50GB" },
        { "reservedSpace": "100GB", "all": true }
      ]
    }
  }
}
```

> [!NOTE] In the Docker daemon configuration file, the "equals" operator in GC filters is denoted using a single `=`, whereas BuildKit's configuration file uses `==`:
>
> | `daemon.json` | `buildkitd.toml` |
> | --- | --- |
> | `type=source.local` | `type==source.local` |
> | `private=true` | `private==true` |

`daemon.json`	`buildkitd.toml`
`shared=true`	`shared==true`

See prune filters for information about available GC filters. GC configuration in `daemon.json` supports all filters except `mutable` and `immutable`.

BuildKit configuration file

For build drivers other than `docker`, GC is configured using a `buildkitd.toml` configuration file. This file uses the following high-level configuration options that you can use to tweak the thresholds for how much disk space BuildKit should use for cache:

Option	Description	Default value
`reservedSpace`	The minimum amount of disk space BuildKit is allowed to allocate for cache. Usage below this threshold will not be reclaimed during garbage collection.	10% of total disk space or 10GB (whichever is lower)
`maxUsedSpace`	The maximum amount of disk space that BuildKit is allowed to use. Usage above this threshold will be reclaimed during garbage collection.	60% of total disk space or 100GB (whichever is lower)
`minFreeSpace`	The amount of disk space that must be kept free.	20GB

You can set these options either as number of bytes, a unit string (for example, `512MB`), or as a percentage of the total disk size. Changing these options influences the default GC policies used by the BuildKit worker. With the default thresholds, the GC policies resolve as follows:

```
# Global defaults
[worker.oci]
  gc = true
  reservedSpace = "10GB"
  maxUsedSpace = "100GB"
  minFreeSpace = "20%"

# Policy 1
[[worker.oci.gcpolicy]]
  filters = [ "type==source.local", "type==exec.cachemount", "type==source.git.checkout" ]
  keepDuration = "48h"
  maxUsedSpace = "512MB"
```

```
# Policy 2
[[worker.oci.gcpolicy]]
  keepDuration = "1440h" # 60 days
  reservedSpace = "10GB"
  maxUsedSpace = "100GB"

# Policy 3
[[worker.oci.gcpolicy]]
  reservedSpace = "10GB"
  maxUsedSpace = "100GB"

# Policy 4
[[worker.oci.gcpolicy]]
  all = true
  reservedSpace = "10GB"
  maxUsedSpace = "100GB"
```

In practical terms, this means:

- Policy 1: If the build cache exceeds 512MB, BuildKit removes cache records for local build contexts, remote Git contexts, and cache mounts that haven't been used in the last 48 hours.
- Policy 2: If disk usage exceeds 100GB, unshared build cache older than 60 days is removed, ensuring at least 10GB of disk space is reserved for cache.
- Policy 3: If disk usage exceeds 100GB, any unshared cache is removed, ensuring at least 10GB of disk space is reserved for cache.
- Policy 4: If disk usage exceeds 100GB, all cache—including shared and internal records—is removed, ensuring at least 10GB of disk space is reserved for cache.

`reservedSpace` has the highest priority in defining the lower limit for build cache size. If `maxUsedSpace` or `minFreeSpace` would define a lower value, the minimum cache size would never be brought below `reservedSpace`.

If both `reservedSpace` and `maxUsedSpace` are set, a GC sweep results in a cache size between those thresholds. For example, if `reservedSpace` is set to 10GB, and `maxUsedSpace` is set to 20GB, the resulting amount of cache after a GC run is less than 20GB, but at least 10GB.

You can also define completely custom GC policies. Custom policies also let you define filters, which lets you pinpoint the types of cache entries that a given policy is allowed to prune.

Custom GC policies in BuildKit

Custom GC policies let you fine-tune how BuildKit manages its cache, and gives you full control over cache retention based on criteria such as cache type, duration, or disk space thresholds. If you need full control over the cache thresholds and how cache records should be prioritized, defining custom GC policies is the way to go.

To define a custom GC policy, use the `[[worker.oci.gcpolicy]]` configuration block in `buildkitd.toml`. Each policy define the thresholds that will be used for that policy. The global values for `reservedSpace`, `maxUsedSpace`, and `minFreeSpace` do not apply if you use custom policies.

Here's an example configuration:

```
# Custom GC Policy 1: Remove unused local contexts older than 24 hours
[[worker.oci.gcpolicy]]
  filters = ["type==source.local"]
  keepDuration = "24h"
  reservedSpace = "5GB"
  maxUsedSpace = "50GB"

# Custom GC Policy 2: Remove remote Git contexts older than 30 days
[[worker.oci.gcpolicy]]
  filters = ["type==source.git.checkout"]
  keepDuration = "720h"
  reservedSpace = "5GB"
  maxUsedSpace = "30GB"

# Custom GC Policy 3: Aggressively clean all cache if disk usage exceeds 90GB
[[worker.oci.gcpolicy]]
  all = true
  reservedSpace = "5GB"
  maxUsedSpace = "90GB"
```

In addition to the `reservedSpace`, `maxUsedSpace`, and `minFreeSpace` threshold, when defining a GC policy you have two additional configuration options:

- `all`: By default, BuildKit will exclude some cache records from being pruned during GC. Setting this option to `true` will allow any cache records to be pruned.
- `filters`: Filters let you specify specific types of cache records that a GC policy is allowed to prune.

See buildx prune filters for information about available GC filters.

□□□ #2
□ □□□ □□□ 2026-10-31 19:04:49 UTC
□ □□□ □□□ 2026-10-31 19:04:51 UTC