Service worker in production

This page is a reference for deploying and supporting production applications that use the Angular service worker. It explains how the Angular service worker fits into the larger production environment, the service worker's behavior under various conditions, and available resources and fail-safes.

Prerequisites A basic understanding of the following: * Service Worker Communication.

Service worker and caching of app resources

Conceptually, imagine the Angular service worker as a forward cache or a CDN edge that is installed in the end user's web browser. The service worker's job is to satisfy requests made by the Angular application for resources or data from a local cache, without needing to wait for the network. Like any cache, it has rules for how content is expired and updated.

{@a versions}

App versions

In the context of an Angular service worker, a "version" is a collection of resources that represent a specific build of the Angular application. Whenever a new build of the application is deployed, the service worker treats that build as a new version of the application. This is true even if only a single file is updated. At any given time, the service worker might have multiple versions of the application in its cache and it might be serving them simultaneously. For more information, see the App tabs section below.

To preserve application integrity, the Angular service worker groups all files into a version together. The files grouped into a version usually include HTML, JS, and CSS files. Grouping of these files is essential for integrity because HTML, JS, and CSS files frequently refer to each other and depend on specific content. For example, an `index.html` file might have a `<script>` tag that references `bundle.js` and it might attempt to call a function `startApp()` from within that script. Any time this version of `index.html` is served, the corresponding `bundle.js` must be served with it. For example, assume that the `startApp()` function is renamed to `runApp()` in both files. In this scenario, it is not valid to serve the old `index.html`, which calls `startApp()`, along with the new bundle, which defines `runApp()`.

This file integrity is especially important when lazy loading modules. A JS bundle might reference many lazy chunks, and the filenames of the lazy chunks are unique to the particular build of the application. If a running application at version `X` attempts to load a lazy chunk, but the server has already updated to version `X + 1`, the lazy loading operation will fail.

The version identifier of the application is determined by the contents of all resources, and it changes if any of them change. In practice, the version is determined by the contents of the `ngsw.json` file, which includes hashes for all known content. If any of the cached files change, the file's hash will change in `ngsw.json`, causing the Angular service worker to treat the active set of files as a new version.

`ngsw.json` is the manifest file that is generated at build time based on `ngsw-config.json`.

With the versioning behavior of the Angular service worker, an application server can ensure that the Angular application always has a consistent set of files.

Update checks Every time the user opens or refreshes the application, the Angular service worker checks for updates to the application by looking for updates to the `ngsw.json` manifest. If an update is found, it is downloaded and cached automatically, and will be served the next time the application is loaded.

Resource integrity

One of the potential side effects of long caching is inadvertently caching an invalid resource. In a normal HTTP cache, a hard refresh or cache expiration limits the negative effects of caching an invalid file. A service worker ignores such constraints and effectively long caches the entire application. Consequently, it is essential that the service worker gets the correct content.

To ensure resource integrity, the Angular service worker validates the hashes of all resources for which it has a hash. Typically for an application created with the Angular CLI, this is everything in the `dist` directory covered by the user's `src/ngsw-config.json` configuration.

If a particular file fails validation, the Angular service worker attempts to re-fetch the content using a "cache-busting" URL parameter to eliminate the effects of browser or intermediate caching. If that content also fails validation, the service worker considers the entire version of the application to be invalid and it stops serving the application. If necessary, the service worker enters a safe mode where requests fall back on the network, opting not to use its cache if the risk of serving invalid, broken, or outdated content is high.

Hash mismatches can occur for a variety of reasons:

- Caching layers in between the origin server and the end user could serve stale content.
- A non-atomic deployment could result in the Angular service worker having visibility of partially updated content.
- Errors during the build process could result in updated resources without `ngsw.json` being updated. The reverse could also happen resulting in an updated `ngsw.json` without updated resources.

Unhashed content The only resources that have hashes in the `ngsw.json` manifest are resources that were present in the `dist` directory at the time the manifest was built. Other resources, especially those loaded from CDNs, have content that is unknown at build time or are updated more frequently than the application is deployed.

If the Angular service worker does not have a hash to validate a given resource, it still caches its contents but it honors the HTTP caching headers by using a policy of "stale while revalidate." That is, when HTTP caching headers for a cached resource indicate that the resource has expired, the Angular service worker continues to serve the content and it attempts to refresh the resource in the background. This way, broken unhashed resources do not remain in the cache beyond their configured lifetimes.

{@a tabs}

App tabs

It can be problematic for an application if the version of resources it's receiving changes suddenly or without warning. See the App versions section above for a description of such issues.

The Angular service worker provides a guarantee: a running application will continue to run the same version of the application. If another instance of the application is opened in a new web browser tab, then the most current version of the app is served. As a result, that new tab can be running a different version of the application than the original tab.

It's important to note that this guarantee is **stronger** than that provided by the normal web deployment model. Without a service worker, there is no guarantee that code lazily loaded later in a running application is from the same version as the initial code for the application.

There are a few limited reasons why the Angular service worker might change the version of a running application. Some of them are error conditions:

- The current version becomes invalid due to a failed hash.
- An unrelated error causes the service worker to enter safe mode; that is, temporary deactivation.

The Angular service worker is aware of which versions are in use at any given moment and it cleans up versions when no tab is using them.

Other reasons the Angular service worker might change the version of a running application are normal events:

- The page is reloaded/refreshed.
- The page requests an update be immediately activated using the `SwUpdate` service.

Service worker updates

The Angular service worker is a small script that runs in web browsers. From time to time, the service worker will be updated with bug fixes and feature improvements.

The Angular service worker is downloaded when the application is first opened and when the application is accessed after a period of inactivity. If the service worker has changed, the service worker will be updated in the background.

Most updates to the Angular service worker are transparent to the app—the old caches are still valid and content is still served normally. However, occasionally a bugfix or feature in the Angular service worker requires the invalidation of old caches. In this case, the application will be refreshed transparently from the network.

Bypassing the service worker

In some cases, you might want to bypass the service worker entirely and let the browser handle the request instead. An example is when you rely on a feature that is currently not supported in service workers (for example, reporting progress on uploaded files).

To bypass the service worker, set `ngsw-bypass` as a request header, or as a query parameter. (The value of the header or query parameter is ignored and can be empty or omitted.)

Debugging the Angular service worker

Occasionally, it might be necessary to examine the Angular service worker in a running state to investigate issues or to ensure that it is operating as designed. Browsers provide built-in tools for debugging service workers and the Angular service worker itself includes useful debugging features.

Locating and analyzing debugging information

The Angular service worker exposes debugging information under the `ngsw/` virtual directory. Currently, the single exposed URL is `ngsw/state`. Here is an example of this debug page's contents:

```
NGSW Debug Info:

Driver version: 13.3.7
Driver state: NORMAL ((nominal))
Latest manifest hash: eea7f5f464f90789b621170af5a569d6be077e5c
Last update check: never

=== Version eea7f5f464f90789b621170af5a569d6be077e5c ===
```

```
Clients: 7b79a015-69af-4d3d-9ae6-95ba90c79486, 5bc08295-aaf2-42f3-a4cc-9e4ef9100f65

=== Idle Task Queue ===
Last update tick: 1s496u
Last update run: never
Task queue:
 * init post-load (update, cleanup)

Debug log:
```

Driver state The first line indicates the driver state:

```
Driver state: NORMAL ((nominal))
```

`NORMAL` indicates that the service worker is operating normally and is not in a degraded state.

There are two possible degraded states:

- `EXISTING_CLIENTS_ONLY`: the service worker does not have a clean copy of the latest known version of the application. Older cached versions are safe to use, so existing tabs continue to run from cache, but new loads of the application will be served from the network. The service worker will try to recover from this state when a new version of the application is detected and installed (that is, when a new `ngsw.json` is available).
- `SAFE_MODE`: the service worker cannot guarantee the safety of using cached data. Either an unexpected error occurred or all cached versions are invalid. All traffic will be served from the network, running as little service worker code as possible.

In both cases, the parenthetical annotation provides the error that caused the service worker to enter the degraded state.

Both states are temporary; they are saved only for the lifetime of the ServiceWorker instance. The browser sometimes terminates an idle service worker to conserve memory and processor power, and creates a new service worker instance in response to network events. The new instance starts in the `NORMAL` mode, regardless of the state of the previous instance.

Latest manifest hash

```
Latest manifest hash: eea7f5f464f90789b621170af5a569d6be077e5c
```

This is the SHA1 hash of the most up-to-date version of the application that the service worker knows about.

Last update check

```
Last update check: never
```

This indicates the last time the service worker checked for a new version, or update, of the application. `never` indicates that the service worker has never checked for an update.

In this example debug file, the update check is currently scheduled, as explained the next section.

Version

```
=== Version eea7f5f464f90789b621170af5a569d6be077e5c ===

Clients: 7b79a015-69af-4d3d-9ae6-95ba90c79486, 5bc08295-aaf2-42f3-a4cc-9e4ef9100f65
```

In this example, the service worker has one version of the application cached and being used to serve two different tabs. Note that this version hash is the "latest manifest hash" listed above. Both clients are on the latest version. Each client is listed by its ID from the `Clients` API in the browser.

Idle task queue

```
=== Idle Task Queue ===
Last update tick: 1s496u
Last update run: never
Task queue:
 * init post-load (update, cleanup)
```

The Idle Task Queue is the queue of all pending tasks that happen in the background in the service worker. If there are any tasks in the queue, they are listed with a description. In this example, the service worker has one such task scheduled, a post-initialization operation involving an update check and cleanup of stale caches.

The last update tick/run counters give the time since specific events happened related to the idle queue. The "Last update run" counter shows the last time idle tasks were actually executed. "Last update tick" shows the time since the last event after which the queue might be processed.

Debug log

```
Debug log:
```

Errors that occur within the service worker will be logged here.

Developer Tools

Browsers such as Chrome provide developer tools for interacting with service workers. Such tools can be powerful when used properly, but there are a few things to keep in mind.

- When using developer tools, the service worker is kept running in the background and never restarts. This can cause behavior with Dev Tools open to differ from behavior a user might experience.
- If you look in the Cache Storage viewer, the cache is frequently out of date. Right click the Cache Storage title and refresh the caches.
- Stopping and starting the service worker in the Service Worker pane triggers a check for updates.

Service Worker Safety

Like any complex system, bugs or broken configurations can cause the Angular service worker to act in unforeseen ways. While its design attempts to minimize the impact of such problems, the Angular service worker contains several failsafe mechanisms in case an administrator ever needs to deactivate the service worker quickly.

Fail-safe

To deactivate the service worker, remove or rename the `ngsw.json` file. When the service worker's request for `ngsw.json` returns a `404`, then the service worker removes all of its caches and de-registers itself, essentially self-destructing.

Safety Worker

Also included in the `@angular/service-worker` NPM package is a small script `safety-worker.js`, which when loaded will unregister itself from the browser and remove the service worker caches. This script can be used as a last resort to get rid of unwanted service workers already installed on client pages.

It's important to note that you cannot register this worker directly, as old clients with cached state might not see a new `index.html` which installs the different worker script. Instead, you must serve the contents of `safety-worker.js` at the URL of the Service Worker script you are trying to unregister, and must continue to do so until you are certain all users have successfully unregistered the old worker. For most sites, this means that you should serve the safety worker at the old Service Worker URL forever.

This script can be used both to deactivate `@angular/service-worker` (and remove the corresponding caches) as well as any other Service Workers which might have been served in the past on your site.

Changing your app's location

It is important to note that service workers don't work behind redirect. You might have already encountered the error `The script resource is behind a redirect, which is disallowed`.

This can be a problem if you have to change your application's location. If you setup a redirect from the old location (for example `example.com`) to the new location (for example `www.example.com`) the worker will stop working. Also, the redirect won't even trigger for users who are loading the site entirely from Service Worker. The old worker (registered at `example.com`) tries to update and sends requests to the old location `example.com` which get redirected to the new location `www.example.com` and create the error `The script resource is behind a redirect, which is disallowed`.

To remedy this, you might need to deactivate the old worker using one of the above techniques (Fail-safe or Safety Worker).

More on Angular service workers

You might also be interested in the following: * Service Worker Configuration.