

# Getting started

Beginning in Angular 5.0.0, you can easily enable Angular service worker support in any CLI project. This document explains how to enable Angular service worker support in new and existing projects. It then uses a simple example to show you a service worker in action, demonstrating loading and basic caching.

See the .

## Adding a service worker to a new application

---

If you're generating a new CLI project, you can use the CLI to set up the Angular service worker as part of creating the project. To do so, add the `--service-worker` flag to the `ng new` command:

```
ng new my-project --service-worker
```

The `--service-worker` flag takes care of configuring your app to use service workers by adding the `service-worker` package along with setting up the necessary files to support service workers. For information on the details, see the following section which covers the process in detail as it shows you how to add a service worker manually to an existing app.

## Adding a service worker to an existing app

---

To add a service worker to an existing app:

1. Add the service worker package.
2. Enable service worker build support in the CLI.
3. Import and register the service worker.
4. Create the service worker configuration file, which specifies the caching behaviors and other settings.
5. Build the project.

### Step 1: Add the service worker package

Add the package `@angular/service-worker`, using the yarn utility as shown here:

```
yarn add @angular/service-worker
```

## Step 2: Enable service worker build support in the CLI

To enable the Angular service worker, the CLI must generate an Angular service worker manifest at build time. To cause the CLI to generate the manifest for an existing project, set the `serviceWorker` flag to `true` in the project's `.angular-cli.json` file as shown here:

```
ng set apps.0.serviceWorker=true
```

## Step 3: Import and register the service worker

To import and register the Angular service worker:

At the top of the root module, `src/app/app.module.ts`, import `ServiceWorkerModule` and `environment`.

Add `ServiceWorkerModule` to the `@NgModule` `imports` array. Use the `register()` helper to take care of registering the service worker, taking care to disable the service worker when not running in production mode.

The file `ngsw-worker.js` is the name of the prebuilt service worker script, which the CLI copies into `dist/` to deploy along with your server.

## Step 4: Create the configuration file, `ngsw-config.json`

The Angular CLI needs a service worker configuration file, called `ngsw-config.json`. The configuration file controls how the service worker caches files and data resources.

You can begin with the boilerplate version from the CLI, which configures sensible defaults for most applications.

Alternately, save the following as `src/ngsw-config.json`:

## Step 5: Build the project

Finally, build the project:

```
ng build --prod
```

The CLI project is now set up to use the Angular service worker.

# Service worker in action: a tour

This section demonstrates a service worker in action, using an example application.

## Serving with `http-server`

As `ng serve` does not work with service workers, you must use a real HTTP server to test your project locally. It's a good idea to test on a dedicated port.

```
cd dist
http-server -p 8080
```

## Initial load

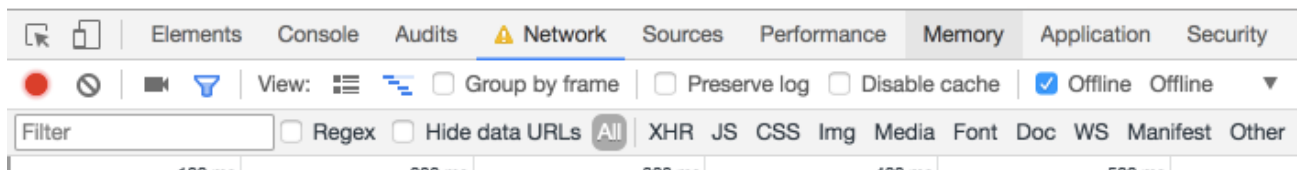
With the server running, you can point your browser at `http://localhost:8080/`. Your application should load normally.

**Tip:** When testing Angular service workers, it's a good idea to use an incognito or private window in your browser to ensure the service worker doesn't end up reading from a previous leftover state, which can cause unexpected behavior.

## Simulating a network issue

To simulate a network issue, disable network interaction for your application. In Chrome:

1. Select **Tools > Developer Tools** (from the Chrome menu located at the top right corner).
2. Go to the **Network** tab.
3. Check the **Offline** box.















Now the app has no access to network interaction.

For applications that do not use the Angular service worker, refreshing now would display Chrome's Internet disconnected page that says "There is no Internet connection".

With the addition of an Angular service worker, the application behavior changes. On a refresh, the page loads normally.

If you look at the Network tab, you can verify that the service worker is active.

Name	Status	Type	Initiator	Size	Time	Waterfall
 localhost	200	docum...	Other	(from ServiceWorker)	4 ms	
 styles.d41d8cd98f00b204e9...	200	stylesh...	(index)	(from ServiceWorker)	10 ms	
 inline.dd7a55677f62886e24...	200	script	(index)	(from ServiceWorker)	9 ms	
 polyfills.6aaab08994953596...	200	script	(index)	(from ServiceWorker)	10 ms	
 vendor.09d746204b73475fb...	200	script	(index)	(from ServiceWorker)	51 ms	
 main.f2ab25821a48c588eb4...	200	script	(index)	(from ServiceWorker)	51 ms	

Notice that under the "Size" column, the requests state is `(from ServiceWorker)`. This means that the resources are not being loaded from the network. Instead, they are being loaded from the service worker's cache.

## What's being cached?

Notice that all of the files the browser needs to render this application are cached. The `ngsw-config.json` boilerplate configuration is set up to cache the specific resources used by the CLI:

- `index.html`.
- `favicon.ico`.
- Build artifacts (JS and CSS bundles).
- Anything under `assets`.

## Making changes to your application

Now that you've seen how service workers cache your application, the next step is understanding how updates work.

1. If you're testing in an incognito window, open a second blank tab. This will keep the incognito and the cache state alive during your test.
2. Close the application tab, but not the window. This should also close the Developer Tools.
3. Shut down `http-server`.
4. Next, make a change to the application, and watch the service worker install the update.
5. Open `src/app/app.component.html` for editing.
6. Change the text `Welcome to {{title}}!` to `Bienvenue à {{title}}!`.
7. Build and run the server again:

```
ng build --prod
cd dist
http-server -p 8080
```

## Updating your application in the browser

Now look at how the browser and service worker handle the updated application.

1. Open `http://localhost:8080` again in the same window. What happens?

# Welcome to app!



What went wrong? Nothing, actually. The Angular service worker is doing its job and serving the version of the application that it has **installed**, even though there is an update available. In the interest of speed, the service worker doesn't wait to check for updates before it serves the application that it has cached.

If you look at the `http-server` logs, you can see the service worker requesting `/ngsw.json`. This is

how the service worker checks for updates.

# Bienvenue a app!



1. Refresh the page.

# Bienvenue a app!



The service worker installed the updated version of your app *in the background*, and the next time the page is loaded or reloaded, the service worker switches to the latest version.