

Get Started With Workbox For npm Script



By Kayce Basques

Technical Writer for Chrome DevTools

In this codelab, you use Workbox to make a simple web app work offline.

If you'd like a conceptual overview of Workbox before starting this tutorial, see the [Overview](#).

Step 1: Set up your project

The project that you're going to add Workbox to is hosted on [Glitch](#). First, you need to set up Glitch so that you can edit your own copy of the project.

1. Open the [demo](#).

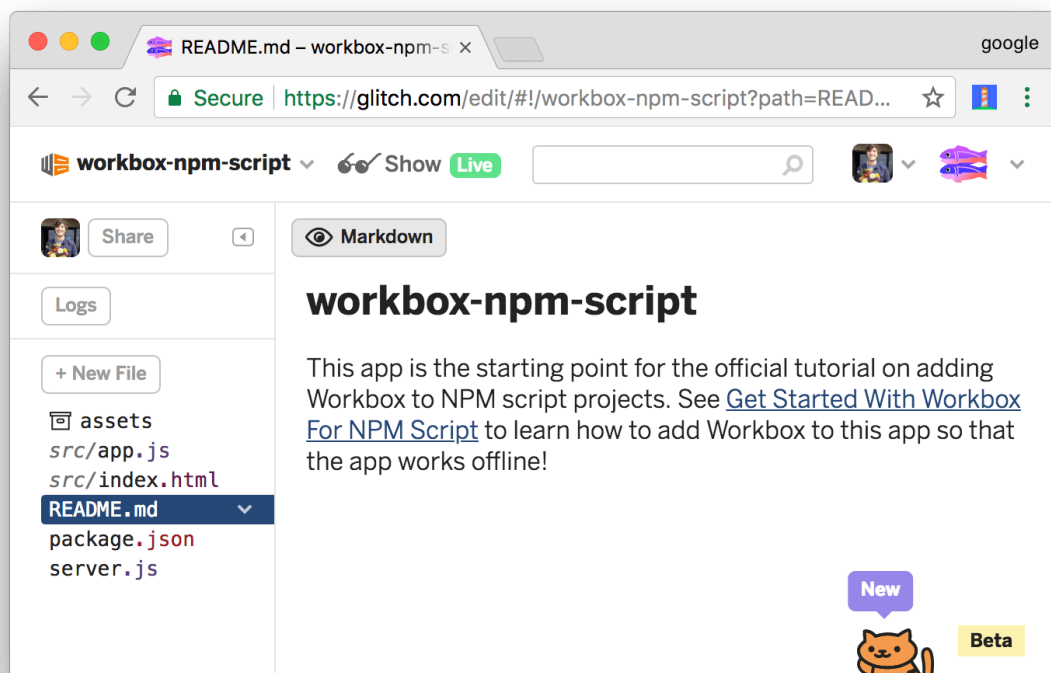


Figure 1. The starting point demo, hosted on Glitch

2. Click **workbox-npm-script** at the top-left of the page. The **Project info and options** dropdown appears.

3. Click **Remix This**. Your browser redirects to an editable copy of the project.

Try out the initial app

The client-side JavaScript in the app fetches the top 10 Hacker News (HN) articles, and then populates the HTML with the content.

Note: This tutorial uses Google Chrome and Chrome DevTools to demonstrate how the web app behaves when offline. You can use [any browser that supports service workers](#).

1. Click **Show**. The live app appears in a new tab.

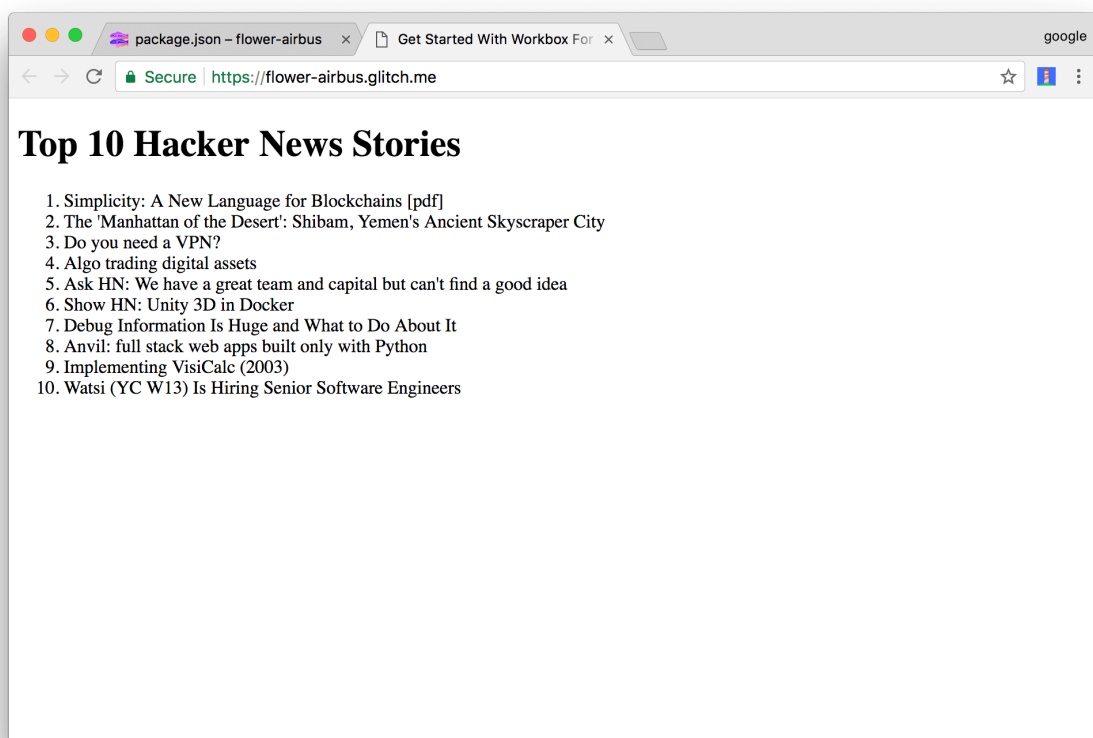


Figure 2. The live app

2. In the tab that's running the live app, press Command+Option+J (Mac) or Control+Shift+J (Windows, Linux) to open DevTools.
3. Focus DevTools and press Command+Shift+P (Mac) or Control+Shift+P (Windows, Linux) to open the **Command Menu**.
4. Type **Offline**, select **Go offline**, then press Enter. Google Chrome now has no connection to the Internet in this tab.

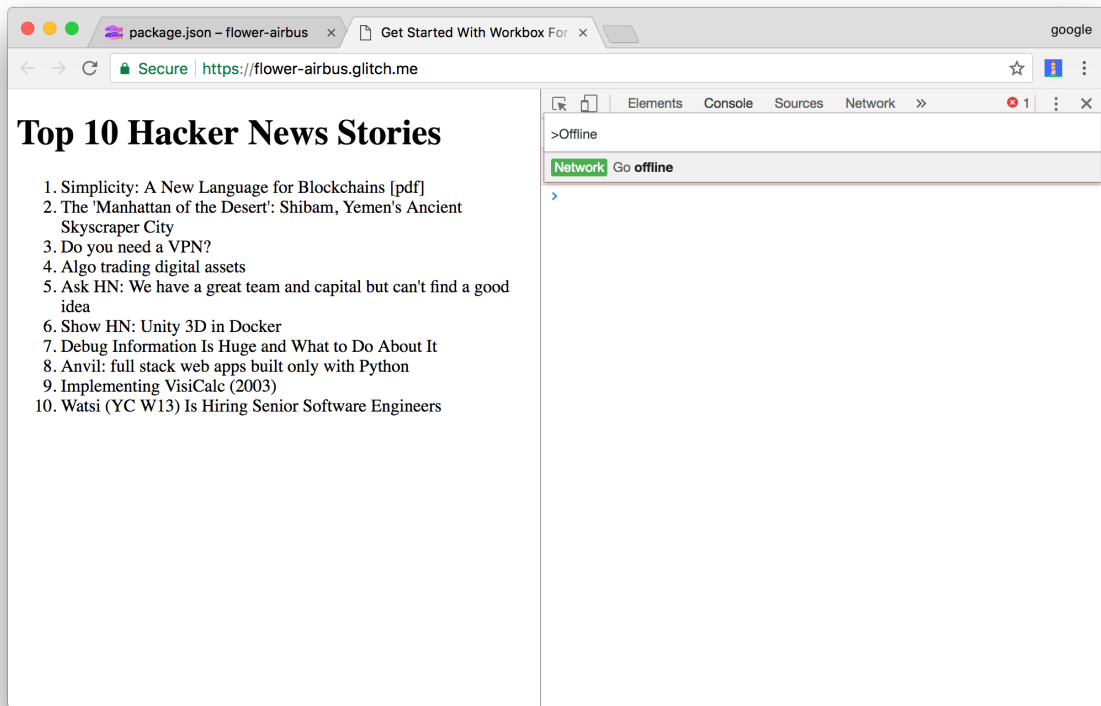


Figure 3. The **Go Offline** command

5. Reload the page. Google Chrome says that you're offline. In other words, the app doesn't work at all when offline.

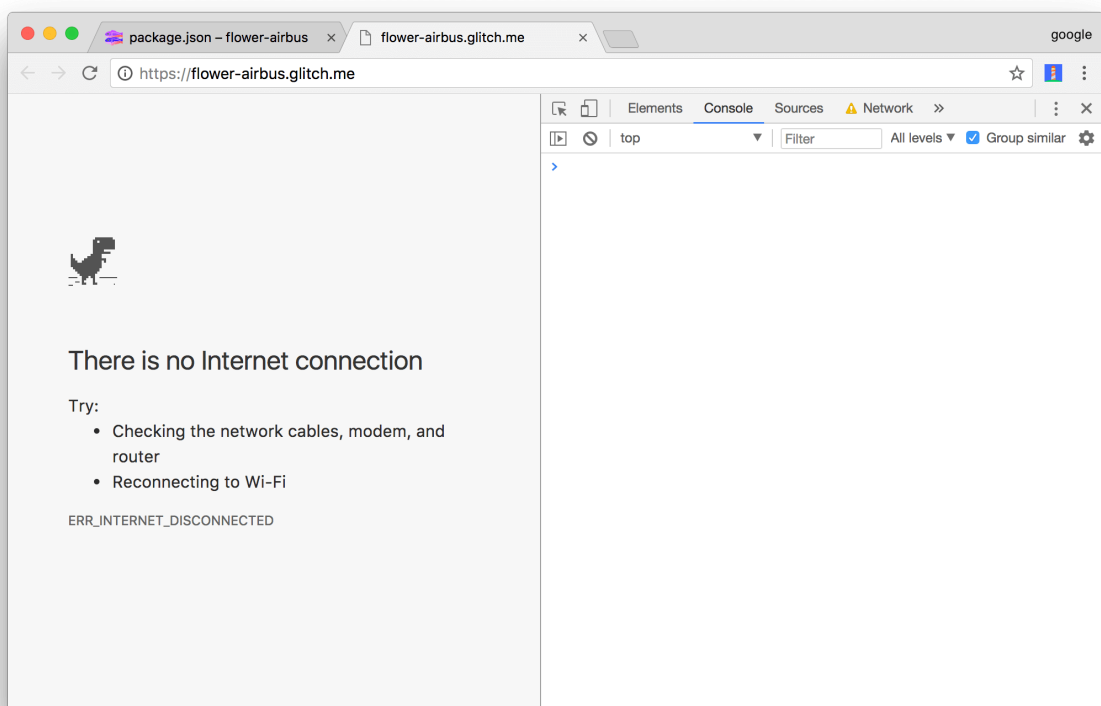


Figure 4. The initial app doesn't work at all when offline

6. Open the **Command Menu** again, type **Online**, and select **Go online** to restore your internet connection in this tab.

Step 2: Install Workbox

Next, you're going to add Workbox to the project to enable an offline experience.

1. Re-focus the tab that shows you the source code of the project.
2. Click `package.json` to open that file.
3. Click **Add package**.
4. Type `workbox-cli` within the **Add Package** text box, then click on the matching package to add it to the project.

★ **Note:** This is equivalent to running `npm install workbox-cli`. In your own projects, you'll probably want to save Workbox as a [development dependency](#) instead by running `npm install workbox-cli --save-dev`, since `workbox-cli` is a build-time tool.

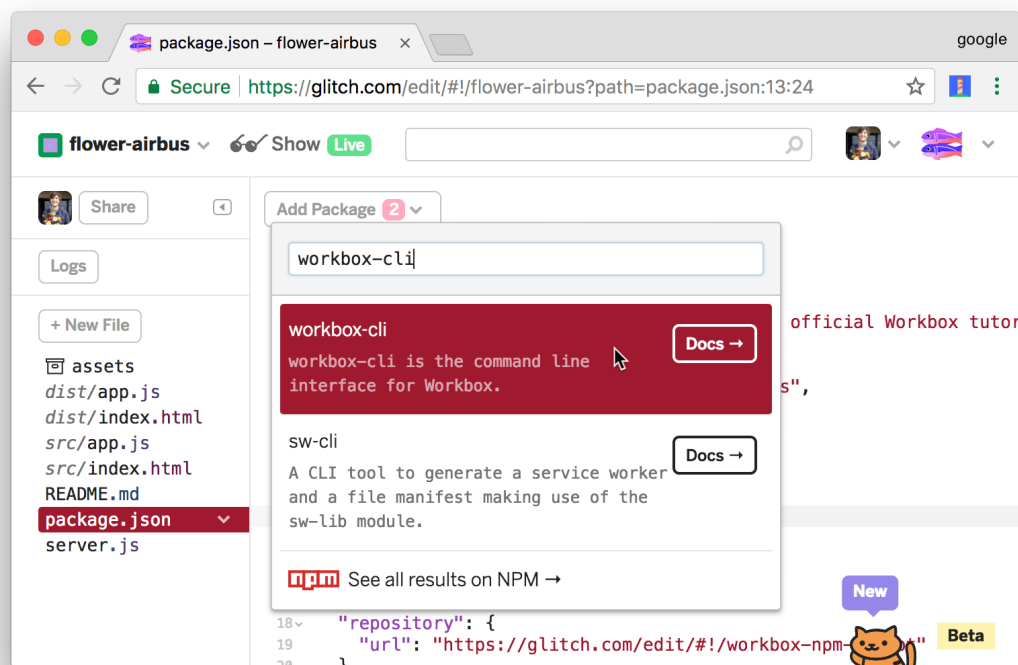


Figure 5. Adding the `workbox-cli` package

Every time you make a change to your code, Glitch automatically re-builds and re-deploys your app. The tab running the live app automatically refreshes, too.

Step 3: Add Workbox to your npm Script build process

Workbox is installed, but you're not using it in your build process, yet.

1. Click **New File**, type `workbox-config.js`, then press Enter.
2. Add the following code to `workbox-config.js`.

```
module.exports = {  
  globDirectory: './dist/',  
  globPatterns: [  
    '**/*.html',  
    '**/*.js'  
  ],  
  swDest: './dist/sw.js',  
  clientsClaim: true,  
  skipWaiting: true  
};
```



3. Open `package.json`.
4. Update your npm scripts to call Workbox as the last step in your build process. The bold code is what you need to add.

```
{  
  ...  
  "scripts": {  
    ...  
    "build": "npm run clean && npm run copy && workbox generateSW"  
  },  
  ...  
}
```



Optional: How the config works

The `build` script in `package.json` controls how the app is built. The build script calls Workbox (`workbox generateSW`) as the last step.

The `workbox generateSW` command automatically searches for a Workbox config file called `workbox-config.js` in the current working directory. If it finds one, it uses that config. Otherwise, a CLI wizard prompts you to configure how Workbox runs.

The object that you define in `workbox-config.js` configures how Workbox runs.

- `globDirectory` is where Workbox watches for changes. `globPatterns` is relative to `globDirectory`.

★ **Note:** A glob is a wildcard pattern. See [Glob Primer](#) to learn more.

- `globPatterns` is a glob of what files to precache. In plain English, the wildcard pattern `**/*.html,js` translates to "cache every HTML and JS file in `globDirectory`, or any of its sub-directories".
- `swDest` is where Workbox outputs the service worker that it generates.
- `clientsClaim` instructs the latest service worker to take control of all clients as soon as it's activated. See [clients.claim](#).
- `skipWaiting` instructs the latest service worker to activate as soon as it enters the waiting phase. See [Skip the waiting phase](#).

Step 4: Register and inspect the generated service worker

Workbox has generated a service worker, but there's no reference to it from your app, yet.

1. Click `src/app.js` to open that file.
2. Register your service worker at the bottom of `init()`.

```
function init() {  
  ...  
  if ('serviceWorker' in navigator) {  
    window.addEventListener('load', () => {  
      navigator.serviceWorker.register('/sw.js').then(registration => {  
        console.log('SW registered: ', registration);  
      }).catch(registrationError => {  
        console.log('SW registration failed: ', registrationError);  
      });  
    });  
  }  
}
```

3. Re-focus the tab that's running the live version of your app. In the DevTools **Console** you see a message indicating that the service worker was registered.
4. Click the **Application** tab of DevTools.

5. Click the **Service Workers** tab.

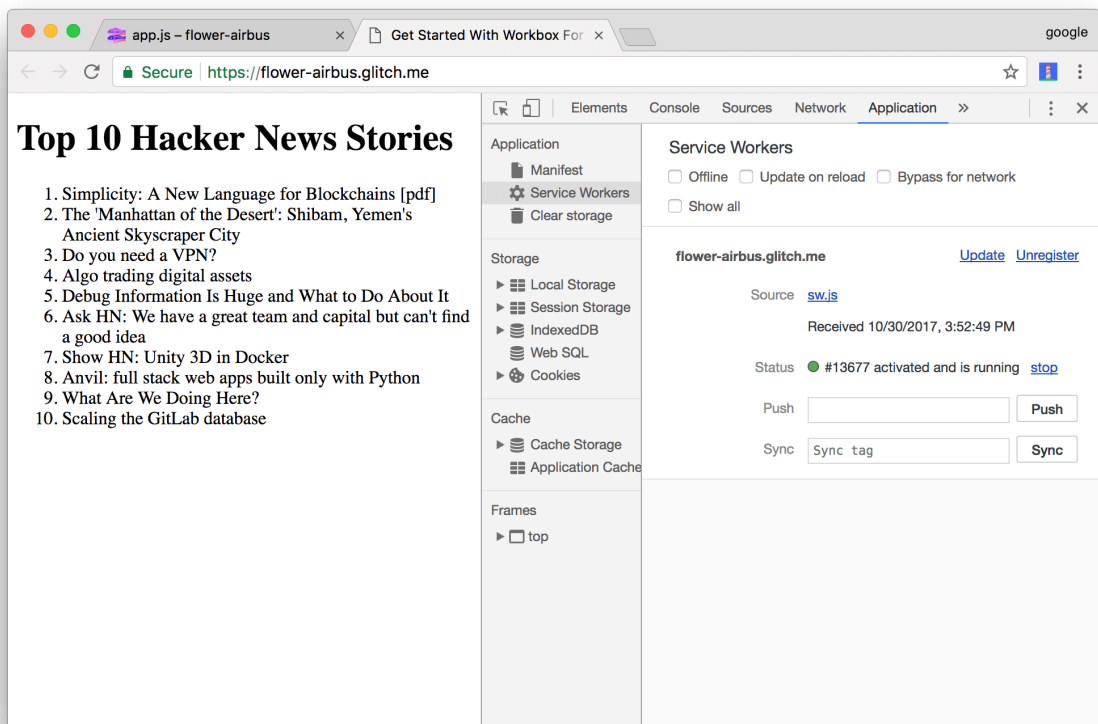


Figure 6. The Service Workers pane

6. Click **sw.js**, next to **Source**. DevTools displays the service worker code that Workbox generated. It should look close to this:



Figure 7. The generated service worker code

Try out the offline-capable app

Your app now sort-of works offline. Try it now:

1. In the live version of your app, use DevTools to go offline again. Focus DevTools and press `Command+Shift+P` (Mac) or `Control+Shift+P` (Windows, Linux) to open the **Command Menu**. Type `Offline`, select **Go offline**, then press `Enter`.

2. Reload the page. The title of the page appears, but the list of the top 10 stories doesn't.
3. Click the **Network** tab in DevTools. The request for `topstories.json` is red, meaning that it failed. This is why the list isn't appearing. The app tries to make a request for `https://hacker-news.firebaseio.com/v0/topstories.json`, but the request fails since you're offline and you haven't instructed Workbox to cache this resource, yet.

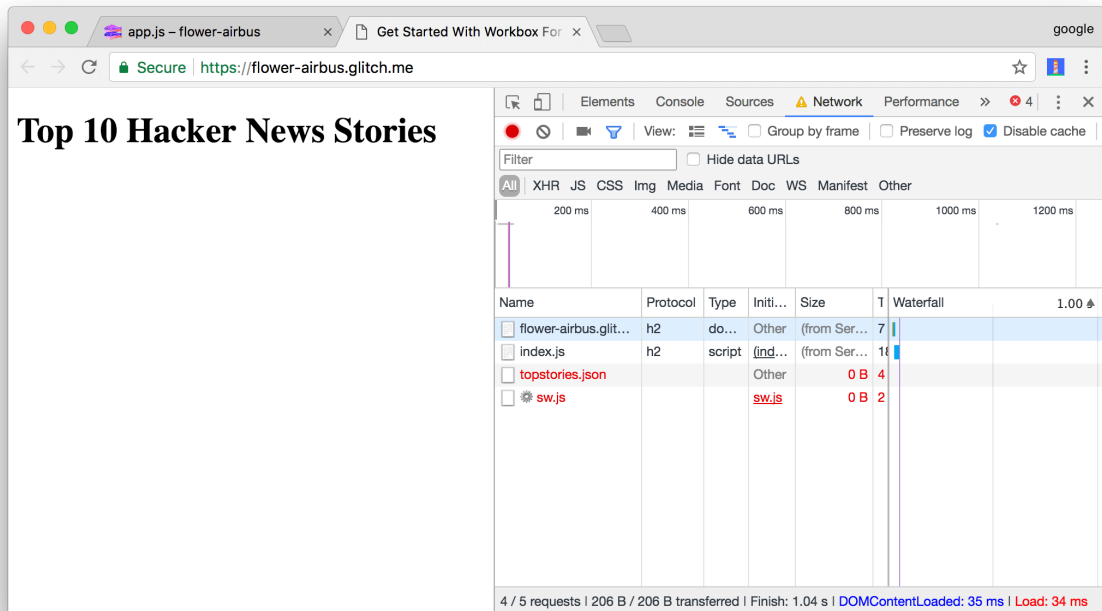


Figure 8. The incomplete offline experience

4. Use the **Command Menu** in DevTools to go back online.

Optional: How the service worker code works

The service worker code is generated based on your Workbox configuration.

- `importScripts('https://storage.googleapis.com/workbox-cdn/releases/3.4.1/workbox-sw.js');` imports Workbox's service worker library. You can inspect this file from the **Sources** panel of DevTools.

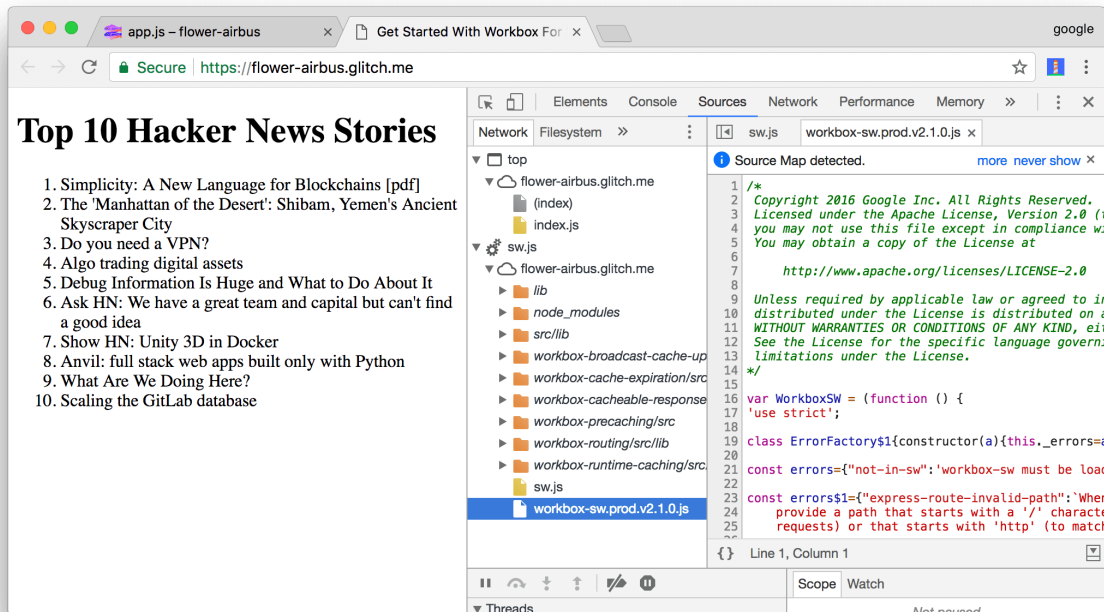


Figure 9. The code for Workbox's service worker library

- The `self.__precacheManifest` array lists all of the resources that Workbox is precaching.
- Each resource has a `revision` property. This is how Workbox determines when to update a resource. Each time you build your app, Workbox generates a hash based on the contents of the resource. If the contents change, then the `revision` hash changes.
- When the service worker runs, it writes the `url` and `revision` of each resource to IndexedDB (IDB) if it doesn't exist. If the resource does exist, the service worker checks that the `revision` in its code matches the `revision` in IDB. If the hashes don't match, then the resource has changed, and therefore the service worker needs to download the updated resource and update the hash in IDB.

In sum, Workbox only re-downloads resources when they change, and ensures that your app always caches the most up-to-date version of each resource.

Step 5: Add runtime caching

Runtime caching lets you store content that's not under your control when your app requests it at runtime. For example, by runtime caching the Hacker News content which this app relies on, you'll be able to provide an improved offline experience for your users. When users visit the app while offline, they'll be able to see the content from the last time that they had an internet connection.

1. Re-focus the tab that shows you the source code of your project.
2. Open `workbox-config.js` again.
3. Add a `runtimeCaching` property to your Workbox configuration. `urlPattern` is a regular expression pattern telling Workbox which URLs to store locally. `handler` defines the caching strategy that Workbox uses for any matching URL. See [The Offline Cookbook](#) for more on caching strategies.

```
module.exports = {
  ...
  runtimeCaching: [{
    urlPattern: new RegExp('https://hacker-news.firebaseio.com'),
    handler: 'staleWhileRevalidate'
  }]
}
```



Try out the complete offline experience

The app now provides a complete offline experience. Try it now:

1. Reload the live version of your app.
2. Use the DevTools **Command Menu** to go back offline.
3. Reload the app. The app now displays the content from the last time that you were online. If you're still only seeing the page title, go back online, reload the page, and then try again.

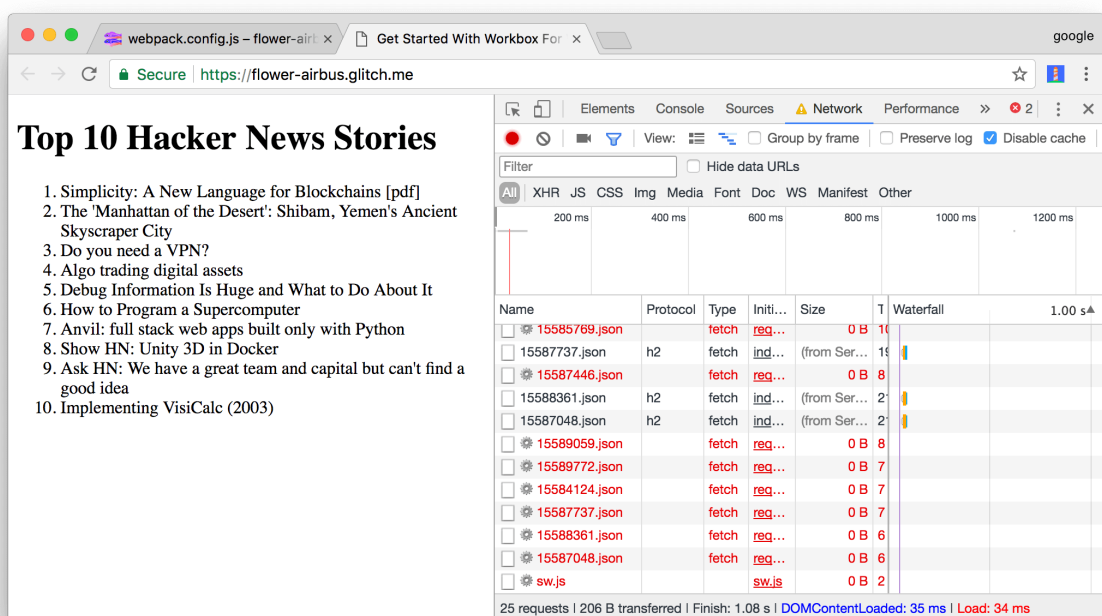


Figure 10. The complete offline experience

4. Use the DevTools **Command Menu** to go back online.


Step 6: Create your own service worker

Up until now, you've been letting Workbox generate your entire service worker. If you've got a big project, or you want to customize how you cache certain resources, or do custom logic in your service worker, then you need to create a custom service worker that calls Workbox instead. Think of the service worker code you write as a template. You write your custom logic with placeholder keywords that instruct Workbox where to inject its code.

In this section, you add push notification support in your service worker. Since this is custom logic, you need to write custom service worker code, and then inject the Workbox code into the service worker at build-time.

1. Re-focus the tab containing your project source code.
2. Add the following line of code to the `init()` function in `app.js`.

```
function init() {  
  ...  
  if ('serviceWorker' in navigator) {  
    window.addEventListener('load', () => {  
      navigator.serviceWorker.register('/sw.js').then(registration => {  
        console.log('SW registered: ', registration);  
        registration.pushManager.subscribe({userVisibleOnly: true});  
      }).catch(registrationError => {  
        ...  
      });  
    });  
  }  
}
```

 **Warning:** For simplicity, this demo asks for permission to send push notifications as soon as the service worker is registered. Best practices strongly recommend against out-of-context permission requests like this in real apps. See [Permission UX](#).

3. Click **New File**, enter `src/sw.js`, then press Enter.
4. Add the following code to `src/sw.js`.

```
importScripts('https://storage.googleapis.com/workbox-cdn/releases/3.4
```

```
// Note: Ignore the error that Glitch raises about workbox being undefined.
workbox.skipWaiting();
workbox.clientsClaim();

workbox.routing.registerRoute(
  new RegExp('https://hacker-news.firebaseio.com'),
  workbox.strategies.staleWhileRevalidate()
);

self.addEventListener('push', (event) => {
  const title = 'Get Started With Workbox';
  const options = {
    body: event.data.text()
  };
  event.waitUntil(self.registration.showNotification(title, options));
});

workbox.precaching.precacheAndRoute([]);
```

★ **Important:** `workbox.precaching.precacheAndRoute([])` is a placeholder keyword. At build-time, Workbox injects the list of files to cache into the array.

5. Open `workbox-config.js`.
6. Remove the `runtimeCaching`, `clientsClaim`, and `skipWaiting` properties. These are now handled in your service worker code.
7. Add the `swSrc` property to instruct Workbox to inject its code into a custom service worker. The complete `workbox-config.js` file now looks like this:

```
module.exports = {
  globDirectory: './dist/',
  globPatterns: [
    '**/*.html',
    '**/*.js'
  ],
  swDest: './dist/sw.js',
  swSrc: './src/sw.js'
};
```



8. Open `package.json`.
9. In your build script, change `generateSW` to `injectManifest`.

```

{
  ...
  "scripts": {
    ...
    "build": "npm run clean && npm run copy && workbox injectManifest"
  }
  ...
}

```

Try out push notifications

The app is now all set to handle push notifications. Try it now:

1. Re-focus the tab running the live version of your app.
2. Click **Allow** when Chrome asks you if you want to grant the app permission to send push notifications.
3. Go back to the **Service Workers** tab in DevTools.
4. Enter some text into the **Push** text box, then click **Push**. Your operating system displays a push notification from the app.

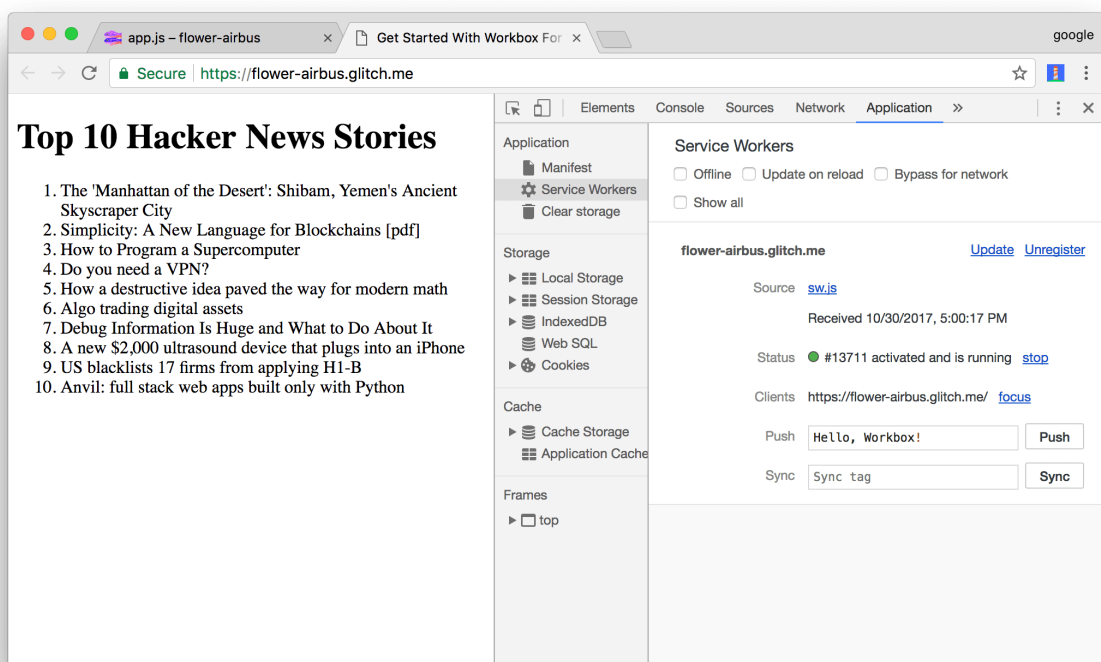


Figure 11. Simulating a push notification from DevTools

★ **Note:** If you don't see the **Push** text box, you're running an older version of Chrome. Click the **Push** link instead. DevTools sends a notification with the text **Test push message from DevTools**.

Optional: How service worker injection works

At the bottom of your custom service worker, you call `workbox.precaching.precacheAndRoute([]);`. This is a placeholder. At build-time, the Workbox plugin replaces the empty array with the list of resources to precache. Your Workbox build configuration still determines what resources get precached.

Next steps

- Read the [Overview](#) to learn more about the benefits that Workbox can provide to your project and how Workbox works.
- If you plan on building a custom service worker, it's helpful to understand [the service worker lifecycle](#). See [Service Workers: An Introduction](#) to learn the basics of service workers.
- If you build projects with Workbox and run into questions or issues, [ask a question on Stack Overflow and tag it with workbox](#).

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated July 16, 2018.