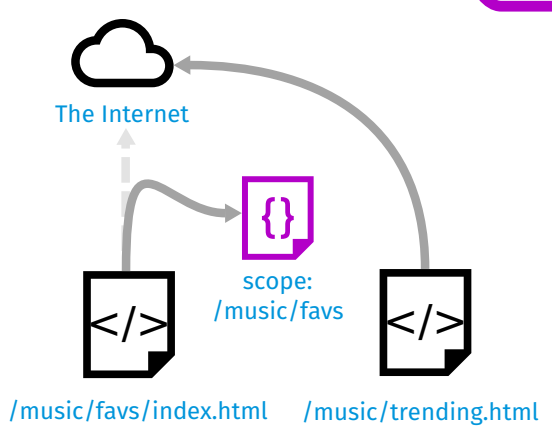
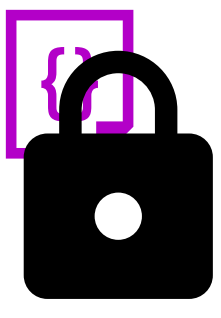
SERVICE WORKERS 101

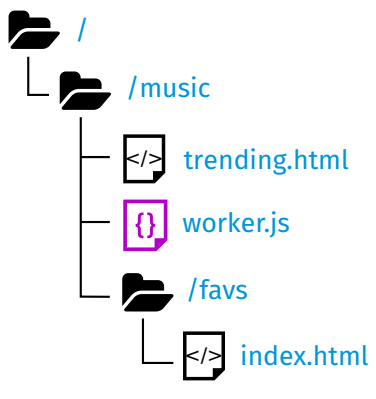
Important!



The service worker will catch requests from the clients **under** scope only.



The service worker must be served over **https**.



The **max scope** for a service worker is the location of the worker.

Pro tip: if you serve a service worker along with the `Service-Worker-Allowed` header, you can specify here a list of **max scopes** for this worker.

Events

- install
- activate
- message

Functional events

- fetch
- sync
- push

Worker lifecycle

INSTALLING

This stage marks the beginning of registration. It's intended to allow to setup worker-specific resources such as offline caches.

REGISTRATION: **Installing** | Waiting | Active

install

Use **event.waitUntil()** passing a promise to extend the installing stage until the promise is resolved.

Use **event.skipWaiting()** in the install handler to skip installed stage and directly jump to activating stage without waiting for currently controlled clients to close.

INSTALLED

The service worker has finished its setup and it's waiting for clients using other service workers to be closed.

REGISTRATION: Installing | **Waiting** | Active

ACTIVATING

There are no clients controlled by other workers. This stage is intended to allow the worker to finish the setup or clean other worker's related resources like removing old caches.

REGISTRATION: Installing | Waiting | **Active**

activate

Use **event.waitUntil()** passing a promise to extend the activating stage until the promise is resolved.

Use **self.client.claim()** in the activate handler to start controlling all open clients without reloading them.

ACTIVATED

The service worker can now handle functional events.

REGISTRATION: Installing | Waiting | **Active**

REDUNDANT

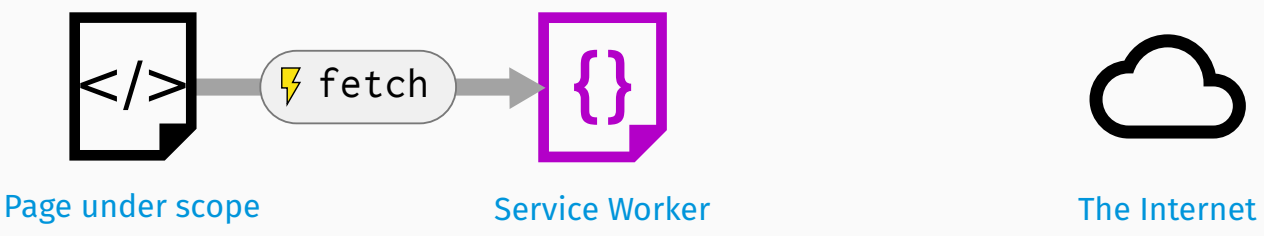
This service worker is being replaced by another one.

REGISTRATION: Installing | Waiting | Active

fetch

1. After registering a service worker to controll a scope, each time a client in that scope request a resource from **anyplace** in the network...

2. The service worker for that scope enters the game and **hijacks** the request.



3. Through **event.respondWith()** and passing a response object, the service worker can answer the client's request with a custom response made out of different sources: the network, storage or its own code.