Updater

Tauri offers a built-in updater for the NSIS (Windows), MSI (Windows), AppImage (Linux) and App bundle (macOS) distribution formats.

Once your Tauri project is ready, you can configure Tauri's updater to enable auto updating for your users.

Signing Updates

Tauri's updater has a built-in signature mechanism to ensure that updates are safe to be installed.

To sign your updates you need two things:

1. The *public key*, which will be added to your `tauri.conf.json` file later, to validate update artifacts before the installation.
2. The *private key*, which is used to sign your update artifacts and should NEVER be shared with anyone. Also, if you lose this key, you will NOT be able to publish new updates to your current user base. It is crucial to store it in a safe place where you can always access it.

To generate the keys on Linux and macOS you can use the Tauri CLI:

npm Yarn pnpm Cargo

```
$ cargo tauri signer generate -w ~/.tauri/myapp.key
```

If you are on Windows, you should use `$HOME/.tauri/myapp.key` or a different path of your choice instead:

npm Yarn pnpm Cargo

```
$ cargo tauri signer generate -w $HOME/.tauri/myapp.key
```

Tauri Configuration

Now you need to configure Tauri's updater. To do this, add this to your Tauri config:

tauri.conf.json

```json
{
  "tauri": {
    "updater": {
      "active": true,
      "endpoints": [
        "https://releases.myapp.com/{{target}}/{{arch}}/{{current_version}}"
      ],
      "dialog": true,
      "pubkey": "YOUR_UPDATER_SIGNATURE_PUBKEY_HERE"
    }
  }
}
```

The required keys are `"active"`, `"endpoints"` and `"pubkey"` to enable the updater. `"dialog"` is optional and will default to `true` if not set.

`"active"` must be a boolean. By default it's set to false.

`"endpoints"` must be an array of updater endpoint URLs as strings. TLS is enforced in production mode. Each updater URL can contain the following variables allowing you to determine server-side if an update is available:

- `{{current_version}}`: The version of the app that is requesting the update.
- `{{target}}`: The operating system name (one of `linux`, `windows` or `darwin`).
- `{{arch}}`: The architecture of the machine (one of `x86_64`, `i686`, `aarch64` or `armv7`).

`"pubkey"` must be a valid public key generated with Tauri's CLI above.

`"dialog"` if present must be a boolean. By default it's set to true. If enabled, updater events will be disabled as the built-in dialog handles everything. If you need custom events, you must turn off the built-in dialog.

`installMode` on Windows

On Windows there is an additional optional config `"installMode"` to change how the update is installed.

tauri.conf.json

```json
{
  "tauri": {
    "updater": {
      "windows": {
        "installMode": "passive"
      }
    }
  }
}
```

- `"passive"`: There will be a small window with a progress bar. The update will be installed without requiring any user interaction. Generally recommended and the default mode.
- `"basicUi"`: There will be a basic user interface shown which requires user interaction to finish the installation.
- `"quiet"`: There will be no progress feedback to the user. With this mode the installer cannot request admin privileges by itself so it only works in user-wide installations or when your app itself already runs with admin privileges. Generally not recommended.

Update Artifacts

Tauri's bundler will automatically generate and sign update artifacts once the updater is correctly configured and enabled.

Before building your app, you need to set environment variables for the private key and password:

- `TAURI_PRIVATE_KEY`: Path or content of your private key
- `TAURI_KEY_PASSWORD`: Your private key password (optional)

If you want to set these variables for the current console session you could execute these commands in the console which you will use to build the app later:

Bash PowerShell

```bash
$ export TAURI_PRIVATE_KEY="content of the generated key"
$ export TAURI_KEY_PASSWORD="password"
```

After that, you can run `tauri build` as usual and Tauri will generate the update bundle and its signature.

- **Linux**: On Linux, Tauri will create a `.tar.gz` archive from the AppImage inside the `target/release/bundle/appimage/` folder:
  - `myapp.AppImage` - the standard app bundle.
  - `myapp.AppImage.tar.gz` - the updater bundle.
  - `myapp.AppImage.tar.gz.sig` - the signature of the update bundle.
- **macOS**: On macOS, Tauri will create a `.tar.gz` archive from the application bundle inside the `target/release/bundle/macos/` folder:
  - `myapp.app` - the standard app bundle.
  - `myapp.app.tar.gz` - the updater bundle.
  - `myapp.app.tar.gz.sig` - the signature of the update bundle.
- **Windows**: On Windows, Tauri will create `.zip` archives from the MSI and NSIS installers inside the `target/release/bundle/msi/` and `target/release/bundle/nsis` folders:
  - `myapp-setup.exe` - the standard app bundle.
  - `myapp-setup.nsis.zip` - the updater bundle.
  - `myapp-setup.nsis.zip.sig` - the signature of the update bundle.
  - `myapp.msi` - the standard app bundle.
  - `myapp.msi.zip` - the updater bundle.
  - `myapp.msi.zip.sig` - the signature of the update bundle.

The signature can be uploaded and shared safely as long as your private key is secure.

Server Support

Tauri's updater supports two ways of announcing update data:

- A static JSON file (to use on services like S3 or GitHub gists)
- A dynamic update server

The static JSON file is easier to use while a dynamic update server will give you finer control over the update mechanism.

Static JSON File

With this approach, Tauri will always request the same JSON file and determine if the app needs to be updated by comparing the version field of the response with the requesting app's current version. Tauri will expect a response in this format:

```json
{
  "version": "v1.0.0",
  "notes": "Test version",
  "pub_date": "2020-06-22T19:25:57Z",
  "platforms": {
    "darwin-x86_64": {
      "signature": "Content of app.tar.gz.sig",
      "url": "https://github.com/username/reponame/releases/download/v1.0.0/app-x86_64.app.tar.gz"
    },
    "darwin-aarch64": {
      "signature": "Content of app.tar.gz.sig",
      "url": "https://github.com/username/reponame/releases/download/v1.0.0/app-aarch64.app.tar.gz"
    },
    "linux-x86_64": {
      "signature": "Content of app.AppImage.tar.gz.sig",
      "url": "https://github.com/username/reponame/releases/download/v1.0.0/app-amd64.AppImage.tar.gz"
    },
    "windows-x86_64": {
      "signature": "Content of app-setup.nsis.sig or app.msi.sig, depending on the chosen format",
      "url": "https://github.com/username/reponame/releases/download/v1.0.0/app-x64-setup.nsis.zip"
    }
  }
}
```

The required keys are `"version"`, `"platforms.[target].url"` and `"platforms.[target].signature"`; the others are optional.

- `"version"` must be a valid semver, with or without a leading `v`, meaning that both `1.0.0` and `v1.0.0` are valid.
- `"platforms"`: Each platform key is in the `OS-ARCH` format, where `OS` is one of `linux`, `darwin` or `windows`, and `ARCH` is one of `x86_64`, `aarch64`, `i686` or `armv7`.
- `"url"` must be a valid url to the update bundle.
- `"signature"` must be the **content** of the generated `.sig` file. The signature may change each time you run `tauri build` so make sure to always update it.

- `"notes"`: Here you can add notes about the update, like release notes. Tauri's default dialog will present this to the user when it asks if it's allowed to update.
- `"pub_date"` must be formatted according to RFC 3339 if present.

Note that Tauri will validate the *whole* file before checking the version field, so make sure all existing platform configurations are valid and complete.

Dynamic Update Server

With this approach, Tauri will follow the update server's instructions. To disable the internal version check you can overwrite Tauri's version comparison to always install the version sent by the server. This could be useful if you need to roll back your app version quickly.

Your server can use variables defined in the `endpoint` url above to determine if an update is required. If you need more data, you can include additional request headers in Rust to your liking.

Your server should respond with a status code of `204 No Content` if there is no update available.

If an update is required, your server should respond with a status code of `200 OK` and a JSON response in this format:

```json
{
  "version": "0.2.0",
  "pub_date": "2020-09-18T12:29:53+01:00",
  "url": "https://mycompany.example.com/myapp/releases/myrelease.tar.gz",
  "signature": "Content of the relevant .sig file",
  "notes": "These are some release notes"
}
```

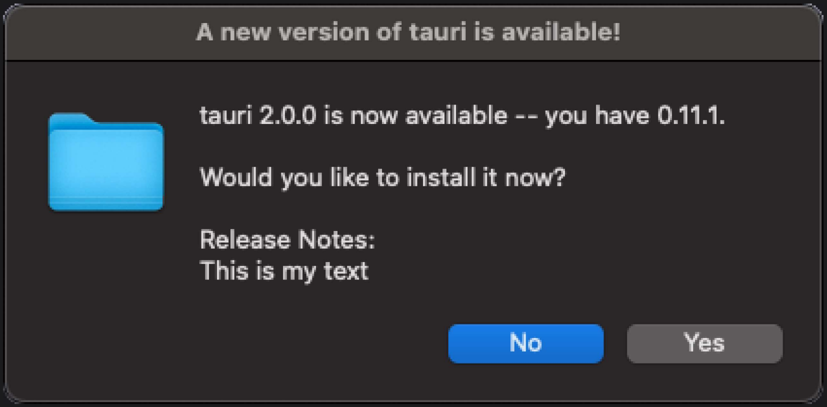
The required keys are "url", "version" and "signature"; the others are optional.

- `"version"` must be a valid semver, with or without a leading `v`, meaning that both `1.0.0` and `v1.0.0` are valid.
- `"url"` must be a valid url to the update bundle.
- `"signature"` must be the **content** of the generated `.sig` file. The signature may change each time you run `tauri build` so make sure to always update it.
- `"notes"`: Here you can add notes about the update, like release notes. Tauri's default dialog will present this to the user when it asks if it's allowed to update.
- `"pub_date"` must be formatted according to RFC 3339 if present.

Checking for Updates

Built-In Dialog

By default, the updater shows a dialog using Tauri's dialog.ask API internally. The dialog will only check for a new update when the app was just launched or when you manually emit the `"tauri://update"` event.



A new version of tauri is available!

tauri 2.0.0 is now available -- you have 0.11.1.

Would you like to install it now?

Release Notes:
This is my text

The dialog release notes are represented by the update `notes` provided by the server. If the user accepts, the update is downloaded and installed. Afterwards, the user is prompted to restart the application.

Custom Dialog

CAUTION

You need to disable the built-in dialog in your Tauri configuration to enable the JavaScript APIs and updater events!

Rust

Please see the updater module documentation at docs.rs for the Rust API.

JavaScript

For the complete API docs see here. An example using the JavaScript API looks like this:

updater.ts

```ts
import {
  checkUpdate,
  installUpdate,
  onUpdaterEvent,
} from '@tauri-apps/api/updater'
import { relaunch } from '@tauri-apps/api/process'
```

```
const unlisten = await onUpdaterEvent(({ error, status }) => {
  // This will log all updater events, including status updates and errors.
  console.log('Updater event', error, status)
})

try {
  const { shouldUpdate, manifest } = await checkUpdate()

  if (shouldUpdate) {
    // You could show a dialog asking the user if they want to install the update here.
    console.log(
      `Installing update ${manifest?.version}, ${manifest?.date}, ${manifest?.body}`
    )

    // Install the update. This will also restart the app on Windows!
    await installUpdate()

    // On macOS and Linux you will need to restart the app manually.
    // You could use this step to display another confirmation dialog.
    await relaunch()
  }
} catch (error) {
  console.error(error)
}

// you need to call unlisten if your handler goes out of scope, for example if the component is unmounted.
unlisten()
```

Edit this page

*Last updated on **Jul 13, 2023***