



Web SDK

Last updated: 03/03/2026

This content applies to the latest CD version of Cumulocity.

Specifications contained herein are subject to change and these changes will be reported in subsequent versions.

Copyright © 2026 Cumulocity GmbH.

The name Cumulocity GmbH and all Cumulocity GmbH product names are either trademarks or registered trademarks of Cumulocity GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

This software may include portions of third-party products. Third-party terms are set out in a 3rd-party-licenses file linked to or included with each installation package.

Table of Contents

Table of Contents	3
INTRODUCTION	5
WHEN CAN I USE THE WEB SDK?	5
PACKAGES	5
@c8y/client: Accessing data	6
@c8y/ngx-components: The component library	6
@c8y/style: Branding and theming	6
@c8y/less-apps: Extendable applications	6
Build tooling: @c8y/websdk and @c8y/devkit	6
VERSIONING: THE WEBSDK USES SEMANTIC VERSION NUMBERS	6
NEXT STEPS	7
GETTING STARTED	8
STEP 1: SELECT THE VERSION	8
STEP 2: SELECT THE BASE PROJECT TO START FROM	8
STEP 3: START THE LOCAL DEVELOPMENT SERVER	9
CREATE YOUR FIRST CUSTOM COMPONENT	9
DEPLOYING YOUR APPLICATION	10
NEXT STEPS	10
APPLICATION CONFIGURATION	11
APPLICATION OPTIONS	11
BRANDING YOUR APPLICATION	13
STYLING BY EXTENDING @C8Y/STYLE	14
SETUP STEPS	14
EXAMPLE CUSTOMIZATIONS	15
APPROACH 1: USING CSS VARIABLES (RUNTIME) — RECOMMENDED	16
APPROACH 2: USING LESS VARIABLES (BUILD-TIME) — ADVANCED	16
LANGUAGES CUSTOMIZATION	17
How to add your own translations at build time	17
How to add your own translations at runtime	18
UPGRADE	19
UPDATING THE WEB SDK VERSION	19
PREPARATION	19
UPDATING	19
DIFFING TO REAPPLY CHANGES	19
VERIFYING THE UPDATE	20
CONCLUSION	20
ANGULAR CLI BEFORE 10.19.X.X	20
INSTALL ANGULAR CLI	20
CREATE A NEW PROJECT	21
ADD CUMULOCITY CLI	21
RUN APPLICATION	21
UPGRADING FROM ANGULAR 19 TO ANGULAR 20	21
BREAKING CHANGES	21
TRACK CHANGES BETWEEN RELEASES	22
C8Y COMMAND LINE TOOL (CLI)	22
GENERAL USAGE	22
OPTIONS	23
COMMANDS	23
THE NEW COMMAND	23
APPLICATION OPTIONS	24
WEBPACK OPTIONS	25
UPGRADING FROM ANGULAR 18 TO ANGULAR 19	25
BREAKING CHANGES	25
TRACK CHANGES BETWEEN RELEASES	26

UPGRADING FROM ANGULAR 17 TO ANGULAR 18	27
UPGRADING FROM ANGULAR 16 TO ANGULAR 17	27
UPGRADING FROM ANGULAR 15 TO ANGULAR 16 AND NG CLI	28
UPGRADING FROM ANGULAR 14 TO ANGULAR 15	29
UPGRADING TO ANGULAR 14	30

INTRODUCTION

This guide provides information on the Web SDK which enables you to:

- Develop web applications that can be deployed to the platform.
- Communicate authenticated with our API.
- Apply default or branded UI components to your custom application.

WHEN CAN I USE THE WEB SDK?

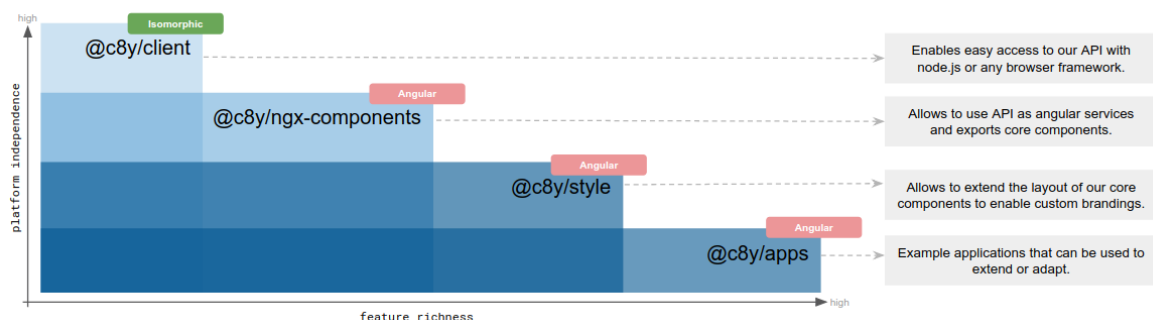
The Web SDK is designed to extend default application or build new IoT solutions. It provides many default components to compose such applications. However, if one of our default applications already fits your usecase, you should think about extending it via the less-code approach and install plugins to them. If you simply have very special needs or can't find the right plugin for your use-case, consider building a custom application with the help of the Web SDK. There you basically have three options:

1. Extend existing applications.
2. Build a new application with the help of the Web SDK.
3. Build a application from scratch.

The third option is the hardest option as you need to build nearly everything new. However, you are free to choose your frameworks and tools. To support you on either of the options, we have developed and published certain packages to npm. Those packages help you to connect to our data, provides you with default components or help you test and deploy the app. Depending on your needs, you can decide which package you want to include. However we always encourage everyone to start with our boilerplates and the full Web SDK. You can find a description on how to use them in the [Getting started](#) guide.

PACKAGES

The Web SDK consists of the following packages deployed to npm in the scope `@c8y` and available under [Apache 2.0](#) license:



These packages depend on each other from top to bottom. While the `@c8y/client` is a very low-level API interface with nearly no dependencies, the `@c8y/cockpit` (for example `@c8y/cockpit`) provide feature rich applications by including `@c8y/ngx-components` and `@c8y/client`.

The goal of these splittings is to provide the right package for every use case. For example, if you want to build a small application with React you could use the `@c8y/client` to do the API interaction. If you need a brandable feature rich application which is close to our Cockpit or Device Management application you could use `@c8y/ngx-components` together with `@c8y/style`.

Following is a list which explains the use cases of each package.

- `@c8y/client`: Use this client to access our API. The client is isomorphic, that means it could be used in node.js and in the browser.
- `@c8y/ngx-components`: A components collection and data access layer for Angular applications. This package can be used to build Angular applications.
- `@c8y/style`: The styles for the look and feel of an application. Extend this package to apply a custom branding to your application.
- `@c8y/cockpit`: The different applications. For example, `@c8y/cockpit`, `@c8y/devicemanagement` and `@c8y/administration`.
- `@c8y/websdk`: The scaffolding package that you can use to add the Cumulocity application to your Angular application.
- `@c8y/devkit`: The builders to build or run a dev server.

@c8y/client: Accessing data

The @c8y/client is an isomorphic (node and browser) Javascript client library for the [Cumulocity REST API](#). It can be used for getting data from the platform. In an Angular application you will mostly use the injected services from [@c8y/ngx-components](#).

@c8y/ngx-components: The component library

ngx-components is a components collection and data access layer for Angular applications. It allows you to access our platform from within an Angular application as well as to provide the core components. To achieve this, the ngx-components consists of two basic imports:

- core ([@c8y/ngx-components](#)) which contains all core components like title, navigator or tabs.
- api ([@c8y/ngx-components/api](#)) which enables dependency injection of the [@c8y/client](#) services.
- features ([@c8y/ngx-components/feature-name](#)) a set of features that add functionality to an application

@c8y/style: Branding and theming

We recommend you to use our build-in branding editor to change the look and feel of all applications in one go. If you still want to change certain style options, the [@c8y/style](#) package contain all our .less based styling. You can extend this styling and build your own custom theme.

@c8y/ apps: Extendable applications

The following table provides an overview on the current packages existing for applications. You can scaffold a new application from any of this applications and then change or extend it.

Name	Description
@c8y/application	An empty application to quickly bootstrap new applications. It is the default application.
@c8y/hybrid	Also an empty application but running in hybrid mode. That means it can import angularjs plugins and therefore can be used for migration purposes.
@c8y/tutorial	An application that already assembles most of the concepts of the @c8y/ngx-components . Use this to get real code examples.
@c8y/widget-plugin	An Module Federation plugin that can be used to create your own custom widgets.
@c8y/package-blueprint	A blueprint (template) for an application which can be uploaded as a package and act as a starting point for a custom application.
@c8y/cockpit	The Cockpit default application. Use this to extend the existing Cockpit application.
@c8y/devicemanagement	The Device Management default application. Use this to extend the existing Device Management application.
@c8y/administration	The Administration default application. Use this to extend the existing Administration application.

Build tooling: @c8y/websdk and @c8y/devkit

Additionally, two build tools are added, which help with scaffolding ([@c8y/websdk](#)) and developing ([@c8y/devkit](#)) the application. The package [@c8y/websdk](#) is quite simple and only provides the prompts for creating a new application. The overall heavy lifting is done by the [@c8y/devkit](#). It replaces the default dev-server and builder options in the [angular.json](#) and extends the webpack configuration with everything needed.

VERSIONING: THE WEBSDK USES SEMANTIC VERSION NUMBERS

Since version 1019.0.0 the versioning schema of the Web SDK isn't aligned anymore to the versioning schema of the Cumulocity platform. The versioning schema is now reflecting via semantic versioning the changes in the Web SDK:

- Major version (for example, **1019.x.x**): Can contain breaking changes. Updating to such versions needs proper testing and validation. Often those versions also contain an Angular upgrade.
- Minor version (for example, **x.3.x**): Contains features that should work without breaking anything. However it is recommended to properly verify the feature a minor version contains.
- Fix version (for example, **x.x.7**): Contains only fixes that should not break anything.

INFO

As npm and semantic versioning only support three parts in the version number, the WebSDK will not use the commonly seen four parts versioning. For example a version sometimes referred to 10.19.0.0, will in the WebSDK be displayed as 1019.0.0. For simplicity this guide will only show three parts versioning numbers.

We recommend to use the `^` or `~` in the `package.json` for all `@c8y` libraries. When you are using a yearly long-term support release, it is best to use the npm tag in your `package.json`. The long-term support versions always end with `-lts`.

As our releases are bound to the Angular versioning, you must ensure that you scaffold the right Angular version. Otherwise the scaffolding process will fail and give you a peer dependency error. The following table shows an overview of the supported versions:

Angular version	Web SDK version	Comment
18.x.x	1021.x.x	No support for the standalone flag
17.x.x	1020.x.x	No support for the standalone flag
16.x.x	1019.x.x	Using Angular CLI tooling .
15.x.x	1018.1.x - 1018.x.x	Using <code>c8ycli</code> tooling, only yearly release
14.x.x	1016.x.x - 1018.0.x	Using <code>c8ycli</code> tooling

INFO

If you want to use an older version then `1019.x.x` you must to use our old tooling based on the `c8ycli` tool-set. For more information see [C8Y Command Line Tool \(CLI\)](#).

NEXT STEPS

If you just want to get started, we recommend to read the [next chapter](#) which explains how you can setup your first Web SDK based Angular application. If you already have setup an application and want to understand more details of the concept, we recommend to jump to our [Developer Codex](#) documentation which explains the concepts, list all the components and defines guidelines for the styling.

GETTING STARTED

This guide will setup your first application. The first step is to install the `@angular/cli` in the right version. Server Side Rendering (SSR) and applications based on the standalone API are not supported and therefore set to `false` :

```
npx @angular/cli@19 new --style=less --ssr=false
```

Second, navigate to the folder and add the `@c8y/websdk` package to your Angular application:

```
ng add @c8y/websdk
```

INFO

Required is a node.js installation. If you have the wrong node.js version installed, the `npm install` step will prompt you with the needed version number.

The CLI will prompt you in two steps for the version and the base template. Afterwards your application is set and you can start your first development server in step 3.

STEP 1: SELECT THE VERSION

```
? Which base version do you want to scaffold from? (Use arrow keys)
> 1019.0.X
> 1019.X.0
> other
```

In the first step, the base scaffolding version must be selected. The interface will provide the latest available release. Additionally a version can be manually entered by selecting the `other` option. If you do not know which version to select, we recommend to use the latest.

STEP 2: SELECT THE BASE PROJECT TO START FROM

```
? Which base project do you want to scaffold from?
administration
application
cockpit
devicemanagement
hybrid
login
tutorial
sample-plugin
package-blueprint
```

In step two, the base project to scaffold from must be selected. You can select any of the default Cumulocity applications to reuse the functions provided there. In alternative, you could start a blank application by selecting the “application” project.

As an alternative to scaffolding, you can get applications from the list above directly from their GitHub repositories:

- [Cumulocity Administration](#)
- [Cumulocity Application](#)
- [Cumulocity Cockpit](#)
- [Cumulocity Device Management](#)
- [Cumulocity Hybrid](#)
- [Cumulocity Login](#)

- [Cumulocity Tutorial](#)
- [Cumulocity Sample plugin](#)
- [Cumulocity Package blueprint](#)

STEP 3: START THE LOCAL DEVELOPMENT SERVER

Now you can start the application by running the `npm start` command. By default, the application will proxy to the Cumulocity cloud platform, however, you can proxy to a different application using the `-u` flag. For example:

```
npm start -- -u http://mytenant.acme.iot
```

or

```
ng serve <appName> -u http://mytenant.acme.iot
```

When you start the command the application begins to compile. After it is compiled, you can navigate to `http://localhost:4200/apps/<<your-app-name>>/` and login to your tenant.

INFO

You must provide your tenant name or the tenant ID on login (as the application cannot derive it from the URL on localhost). If you don't know your tenant name or the tenant ID you can click on your username in your tenant and get the information from the section Platform Information.

INFO

It is possible that node.js needs more memory to compile the project. If you run into an out-of-memory error, assign more memory by setting the environment variable `NODE_OPTIONS` to `--max_old_space_size=4096`.

You are now setup. Any changes you make to your local files will lead to recompiling. After a refresh you will see your changes.

CREATE YOUR FIRST CUSTOM COMPONENT

After creating the empty bootstrapping application you might want to start with your first content. To do so, add a new component in the `src/app` to your project and save it as `hello.component.ts`:

```
import { Component } from "@angular/core";

@Component({
  selector: "app-hello",
  template: `
    <h1>Hello World</h1>
    <p>My first content.</p>
  `,
  standalone: false
})
export class HelloComponent {}
```

Both standalone and module oriented components are supported. For the latter ones `standalone: false` must be added because this property is `true` by default since Angular 19.

To hook the new component into the application, you must declare the new component and add it to a route in the `app.module.ts`. In

the following example we extended the `application` project, which gives you a very clear application frame.

```
import { NgModule } from "@angular/core";
import { BrowserAnimationsModule } from "@angular/platform-browser/animations";
import { RouterModule as ngRouterModule } from "@angular/router";
import { CoreModule, BootstrapComponent } from "@c8y/ngx-components";
import { HelloComponent } from "../hello.component";

@NgModule({
  imports: [
    BrowserAnimationsModule,
    ngRouterModule.forRoot(
      [{ path: "", component: HelloComponent }], // hook the route here
      { enableTracing: false, useHash: true }
    ),
    CoreModule.forRoot(),
  ],
  bootstrap: [BootstrapComponent],
  declarations: [HelloComponent], // add deceleration here
})
export class AppModule {}
```

If you start this application and login, you will see an application similar to the following screenshot.



The application uses a customized router from the Web SDK and the `CoreModule`. The `CoreModule` contains all the necessary components, directives, pipes and services that allow you to [extend](#) the application even further. But first we will release the application and deploy it.

DEPLOYING YOUR APPLICATION

The Angular CLI provides a custom `deploy` command to upload the application. You can run the command `ng deploy` and the current application will be deployed.

For deployment you need an application role, username, password and a tenant. You can also run it by providing this information as parameters. Use the following code to build and deploy the application without prompting:

```
ng deploy -u http://yourtenant.cumulocity.com -T t12345 -U acme -P "*****"
```

In this example we use the custom deploy command added to Angular. You need to provide the option `-T` (tenant), `-U` (user) and `-P` (password) to authenticate on your tenant. The deploy command also accepts environment variables if you do not want to store them, prefixed with `C8Y_`. So for example `C8Y_USER` for the `-U` flag.

NEXT STEPS

- Refer to the [Cumulocity Developer Codex](#) for more information on developing applications in the Cumulocity environment. Moreover find various related tutorials in the [Cumulocity Tech Community](#).

APPLICATION CONFIGURATION

All applications created with the Web SDK are fully customizable through a custom build. We recommend you to leverage branding and configuration options without building the application, as this configuration can then be shared across multiple applications. This section will initially outline the “no-build option” before detailing how to customize a built application.

APPLICATION OPTIONS

The easiest option to customize your application are application options. They apply to any Web SDK application and use inheritance which allows you to apply the same configurations to all your applications. One example configuration is `hideNavigator`, which is a simple flag that configures if the navigator must be shown on application start or not. You can configure this in 3 ways:

1. as a URL parameter
2. as a dynamic public option
3. as a static private option

With regard to inheritance, the option “a URL parameter” contains the highest privilege, followed by the “dynamic public option” and the “static private option” respectively. Simply add a URL parameter to your application for testing out certain options. For example, to hide the navigator you would simply use the URL: `apps/<<your-app-name>>/index.html?hideNavigator=true#/route`. Note that the URL parameter needs to be set before the #-hash navigation.

The dynamic public option is requested by each Web SDK based application upon startup. The default fetch URL for this options is stored in the `dynamicOptionsUrl` which is by default set to `"/apps/public/public-options/options.json"`. As you can see by the context-path, the default setting points to an application deployed to your tenant. To create this application on your own create a zip file called `public-options.zip` and add a `options.json` to it:

```
{
  hideNavigator: true
}
```

If you upload this application to your tenant and subscribe it to at least one of your subtenants, all Web SDK based applications hide the navigator by default.

INFO

If you are an Enterprise tenant customer, the easiest way to manipulate this options is to use the branding manager in administration. It allows you to set most of the settings without the need to manually generate a JSON file and upload any applications.

The static private option can only be defined by a custom application. They are the lowest option level and can be overwritten by any of the upper options (1 and 2). They are also private, meaning they only apply to the current application they are applied to. You can define those options in the `cumulocity.config.ts` file by adding them to the `runTime` fragment:

```
[...]
export default {
  runTime: {
    [...]
    hideNavigator: true
  },
}
[...]
```

The options in the `cumulocity.config.ts` file can also be set dynamically at build time, for example, by accessing the environment

variables:

```
[...]  
export default {  
  runTime: {  
    [...]  
    hideNavigator: process.env.C8Y_HIDE_NAVIGATOR === 'true'  
  },  
}  
}  
[...]
```

By setting the environment variable `C8Y_HIDE_NAVIGATOR` to `true` before building the application via `ng build` you can adjust the behavior of the application. On Linux systems you can run the build command with that environment option just set like this:

```
C8Y_HIDE_NAVIGATOR=true ng build
```

If you have, for example, different “flavors” of your application, you can also switch between the different flavors via an environment variable in your `cumulocity.config.ts` :

```
const flavor = process.env.C8Y_APP_FLAVOR || 'default';  
let standardOptions: ConfigurationOptions = {  
  runTime: {  
    // [...]  
    rightDrawer: true,  
  },  
  buildTime: {  
    // [...]  
  }  
};  
  
switch (flavor) {  
  case 'customerA':  
    standardOptions.runTime.hideNavigator = true;  
    break;  
  
  case 'customerB':  
    standardOptions.runTime.hideNavigator = false;  
    break;  
}  
  
export default standardOptions;
```

For flavor-specific branding, specify different branding files in your `angular.json`:

```
{
  "projects": {
    "your-app": {
      "architect": {
        "build": {
          "configurations": {
            "customerA": {
              "styles": [
                "src/branding/branding-customerA.less"
              ]
            },
            "customerB": {
              "styles": [
                "src/branding/branding-customerB.less"
              ]
            }
          }
        }
      }
    }
  }
}
```

You can then build the different flavors using:

```
ng build --configuration=customerA
```

It is good practice to use the URL option in order to verify an option behavior, or to use the dynamic option in order to set the option platform wide (branding) and to use the private static option in order to set the default for your custom application.

To see all available options, refer to [Styling with application options](#) in the Cumulocity Codex.

BRANDING YOUR APPLICATION

A branding must always be applied to all of your applications. Therefore, it is recommended to use dynamic public options for branding your application. You must set the right [design tokens](#) in the `brandingCssVars` application option. Those are CSS variables that get applied to all default style sheets of Cumulocity and will show your custom branding for any Web SDK application. Your `options.json` then look like this:

```
{
  "brandingCssVars": {
    "--brand-primary": "#B10F2E",
    "--brand-complementary": "#DE7C5A",
    "--brand-dark": "#280000",
    "--brand-light": "#DE7C5A",
    "--palette-status-realtime": "#f0f"
  }
}
```

You can add other options, for example, the `hideNavigator` or add your own CSS file:

```
{
  "brandingCssVars": {
    "--brand-primary": "#B10F2E",
    "--brand-complementary": "#DE7C5A",
    "--brand-dark": "#280000",
    "--brand-light": "#DE7C5A",
    "--palette-status-realtime": "#f0f"
  },
  "hideNavigator": true,
  "extraCssUrls": "./custom.css",
}
```

In the CSS file you can add your own styles:

```
h1 {
  color: #00f;
}
```

Follow the steps below:

1. Zip the files to ensure that they are in the root of the zip without any wrapping folder.
2. Name the zip file `public-options.zip`.
3. Upload it as a web application in **Administration > Ecosystem > Applications**.
4. Subscribe your applications to one of your tenants under **Tenants > Subtenants**.

Afterwards, you can update the existing application with new variables in the detail view of the application.

INFO

If you are an Enterprise tenant customer, the easiest way to manipulate this options is to use the branding manager in administration. It provides an form to set most of the settings without any manual generating of a JSON file and uploading applications.

STYLING BY EXTENDING @C8Y/STYLE

For styling the application global CSS created with [LESS](#) is used. The original LESS source is distributed via the npm package [@c8y/style](#). By extending these styles it is possible to change any detail of the application but the vast majority of developers want to change: colors, logos and fonts and these can be very easily achieved by replacing a few variables.

IMPORTANT

Recommended approach: For most use cases, we recommend using the **branding editor** in the Administration application with **CSS variables** (Approach 1 below). This allows runtime customization without rebuilding applications and provides a user-friendly interface for non-developers.

Use **LESS variables** (Approach 2) only for advanced scenarios requiring build-time customization or when you need deeper control over styling that goes beyond what the branding editor offers.

To override variables, **Custom CSS Properties** —also known as **CSS Variables** —can be utilized, offering configurability at runtime or during the build process.

SETUP STEPS

1. Ensure that your project is based on the Angular CLI (whether upgraded or created from scratch).
2. Make sure you have installed the `@c8y/style` package. If not, you can install the base styles from npm using the following command:

```
npm install @c8y/style
```

The example is based on this file structure:

```
my-application
| ...
| angular.json
| package.json
| ...
└── src
    | styles.less
    | favicon.ico
    | ...
    └── assets
        | logo.jpg
        | ...
```

3. If `styles.less` already exists, add the line `@import '~@c8y/style/extend.less';` at the **top** of the file. If it does not exist, create it and add the mentioned line:

```
/* src/styles.less */
@import '~@c8y/style/extend.less';
```

4. Include the `styles.less` file in the `styles` entry in `angular.json` under your project entry:

```
{
  "projects": {
    "your-app": {
      "architect": {
        "build": {
          "options": {
            "styles": [
              "src/styles.less"
            ]
          }
        }
      }
    }
  }
}
```

! IMPORTANT

The import order is critical: Always import `@c8y/style/extend.less` **first** at the top of your file, then override variables **after**. This ensures the conditional guard system can detect your overrides correctly.

EXAMPLE CUSTOMIZATIONS

At this point, we can modify the desired variables to suit our needs. To implement these changes, follow the examples below and add the specified code to your `styles.less` file.

APPROACH 1: USING CSS VARIABLES (RUNTIME) — RECOMMENDED

You can set CSS variables directly, which allows for runtime customization:

Benefits:

- No rebuild or redeployment needed
- Changes apply instantly at runtime
- Can be managed via the branding editor UI
- Works across all applications tenant-wide
- Easier for non-developers

```
/* src/styles.less */
@import '~@c8y/style/extend.less';

:root {
  --brand-primary: red;
  --brand-logo-img: url(/apps/<applicationContextPath>/assets/logo.jpg);
  --brand-logo-img-height: 48%;
}
```

The `applicationContextPath` can be any application that you uploaded to the platform and which contains the `logo.jpg` file.

User interface elements like buttons, active navigation nodes, and active tabs will use your custom brand color.

APPROACH 2: USING LESS VARIABLES (BUILD-TIME) — ADVANCED

For advanced build-time customization scenarios, override LESS variables **after** importing `extend.less` :

When to use:

- Advanced styling needs beyond CSS variables
- Build-time baking of styles
- Custom application with specific styling requirements

Note: This requires rebuilding and redeploying your application for changes to take effect.

```
/* src/styles.less */
/* 1. Import Cumulocity styles FIRST */
@import '~@c8y/style/extend.less';

/* 2. Override LESS variables AFTER with direct values */
@brand-primary: #e30613; /* Your custom brand color (must use direct hex value) */

/* 3. Logo configuration (paths relative to your styles.less file) */
@brand-logo-img-fallback: './assets/logo.jpg';
@brand-logo-height-fallback: 48%;

/* 4. Navigator logo */
@navigator-platform-logo-fallback: './assets/logo-white.png';
@navigator-platform-logo-height-fallback: 28%;

/* Apply navigator logo (required for extend.less pattern) */
.navigator .tenant-brand {
  background-image: var(--c8y-navigator-platform-logo);
  padding-bottom: var(--c8y-navigator-platform-logo-height);
}
```

! IMPORTANT

Important rules for LESS variables:

1. Always use direct color values (for example, `#e30613` , not `@my-color`).
2. Import `extend.less` **before** overriding variables.
3. Logo paths are relative to your `styles.less` file location.
4. Add explicit CSS rule for navigator logo when using LESS variables.

LANGUAGES CUSTOMIZATION

The platform UI strings used for internationalization are stored in [gettext](#). If you want to add a new language to the platform you need a software to edit these files, for example, [Poedit](#).

Each translated catalog is loaded at runtime in a JSON format. To convert .po (gettext) files into .json files we rely on [@angular/cli](#) installed during the first step.

How to add your own translations at build time

1. Download the string catalog from [@c8y/ngx-components@1004.0.6/locales/locales.pot](#) (starting from version 1004.0.6, `latest` can be replaced by your current used version).
2. Load the downloaded locales.pot template file in your preferred .pot file editor and create a new translation from it. Select the target language of the translation, for example, Afrikaans, and translate each string. Repeat the process for as many languages as you like. The outcome of this step is a .po catalog file for each language. Make sure to store these files in a safe place, as they are useful when updating the strings in subsequent versions.
3. Transform the newly created .po file into a .json file using the `c8ycli` :

```
c8ycli locale-compile path/to/language.po
```

4. Copy the generated .json file into the `ui-assets` folder.
5. Update the languages fragment in `public-options/options.json`.

```
languages?: {  
  [langCode: string]: {  
    name: string;  
    nativeName: string;  
    url: string;  
  }  
}
```

In the example provided in the repository, available as downloaded, you can find an example of a Ukrainian translation which looks like this:

```
[...]
export default {
  runTime: {
    [...]
    languages: {
      uk: {
        name: "Ukraine",
        nativeName: "українська",
        url: "/apps/public/ui-assets/uk.json"
      }
    }
  },
  buildTime: {
    [...]
  }
}
[...]
```

You can change the imported language in the UI after login. To do so, click the User icon at the top right, select User settings from the menu and in the upcoming window select the language of your choice.

How to add your own translations at runtime

You can translate certain strings at runtime, which means they do not have to be included in the build and can simply be added to the [application options](#). However, this concept doesn't allow to add new languages. You can only add new strings to existing languages or align certain translations on existing ones. To translate a certain key you must add the following structure to the application options:

```
i18nExtra?: {
  [langCode: string]: {
    [key: string]: string;
  };
};
```

Example:

The following will translate a custom cookie banner:

```
[...]
export default {
  runTime: {
    [...]
    i18nExtra: {
      de: {
        "About cookies on Cumulocity": "Informationen zu Cookies in Cumulocity",
        "Click Agree and Proceed to accept cookies and go directly to the platform or click on Privacy Policy to see detailed descriptions of the used cookies.": "Klicken Sie auf Zustimmung und fortfahren, um Cookies zu akzeptieren und direkt zur Plattform zu gelangen, oder klicken Sie auf Datenschutzrichtlinie, um detaillierte Beschreibungen der verwendeten Cookies anzuzeigen."
      }
    }
  },
  buildTime: {
    [...]
  }
}
[...]
```

UPGRADE

UPDATING THE WEB SDK VERSION

From version 1019.x.x onwards the Web SDK is following semantic versioning. Meaning that every major version bump (for example, from 1019 to 1020) may contain breaking changes, but every minor or fix bump should not break your application. So it is safe to update to any minor or fix version, but if you update to any major version, you might need to migrate things.

Easiest way for migration at the moment is still comparing the diff with git:

PREPARATION

We recommend you to use a Source Control System to backup the data and to get better diffing of the code. If you are not using an SCM yet, see below for an introduction on how to use [git](#) to store your changes. If you are already using an SCM or you don't want to use any, you can jump to the next section. If you decide not to use an SCM, backup your application before running the update.

Ensure that you have [git](#) installed on your system. Then open a terminal and run the following commands:

```
cd <<path/to-you-app>>
git init
git add .
git commit -m "init commit"
```

Now your code is committed to a local git repository stored in the `.git` folder. Next this recipe explains you how to update the Web SDK. If you don't want to use git anymore after the update, you can simply erase the `.git` folder.

UPDATING

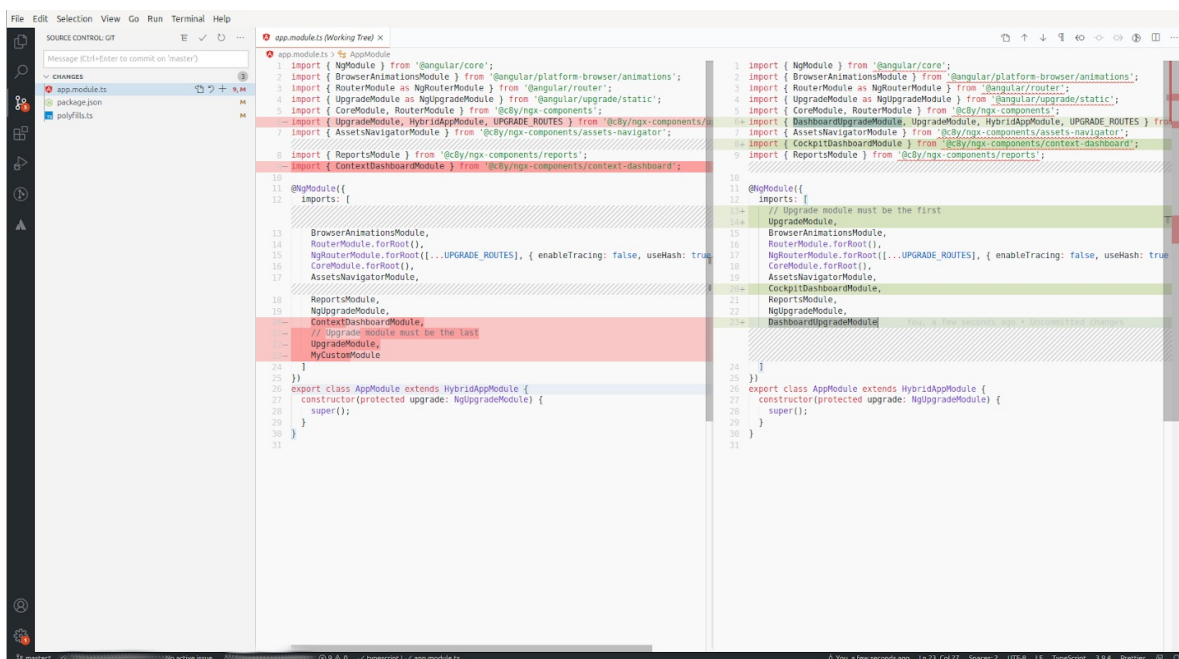
To update the Web SDK you must create a new application with the desired version and copy over the files. The diff then tells you, where merge conflicts might happen. Fix any merge conflicts prior to merging.

```
ng new cockpit
cd cockpit
ng add @c8y/websdk
```

If you select a newer version of the application you want to scaffold, you get the latest scaffolding files. If you copy them over to your solution which was checked in before, you can see the difference between the versions.

DIFFING TO REAPPLY CHANGES

A git diffing tool is useful to identify which changes have been made with the upgrade and which have been made earlier and now must be reapplied. In the following screenshot we are using [Visual Studio Code](#) to identify the changes, as it has a well integrated diffing tool for git (mostly all other IDEs have support for git diffing as well):



With that tool it is easy to compare which file was changed with the upgrade and where custom changes may have to be reapplied. In this case `MyCustomModule` must only be placed in the upgrade `app.module.ts`. When this change is done, the update can be verified.

VERIFYING THE UPDATE

To check if the version update worked, it is usually a good practice to run it locally first. Therefore, you need first to install the dependencies again. Remove the current `node_modules` directory and run `npm install` (or `yarn`) again to refresh the dependencies. After this, start the application with `npm start`. After login you can check the current UI version by clicking on your username.

If everything worked as expected, you can now deploy your application by running the command `npm run deploy`.

CONCLUSION

The update process is sometimes a bit tricky, especially if you have many changes in the `app.module.ts`. However, with git and Visual Studio Code the visual diffing may help you to accomplish this task. Also, it is a good practice to put your own Angular customizations into a module and only to make changes to the `app.module.ts` when it is absolutely necessary.

ANGULAR CLI BEFORE 10.19.X.X

When developing a pure Angular you can create an Angular CLI (`ng -cli`) project and add Cumulocity CLI to it. This functionality is available for:

- Angular 7: Supported from version 10.4.2.0
- Angular 8: Supported from version 10.5.9.0
- Angular 11: Supported from version 10.10.4.0
- Angular 12: Supported from version 10.11.45.0
- Angular 14: Supported from version 10.15.132.0
- Angular 15: Supported from version 10.18.157.0

INSTALL ANGULAR CLI

Follow the [instructions](#) to install `@c8y/cli` globally.

```
npm install -g @angular/cli@v8-lts
```

CREATE A NEW PROJECT

```
ng new my-first-iot-project  
cd my-first-iot-project
```

ADD CUMULOCITY CLI

```
ng add @c8y/cli
```

RUN APPLICATION

```
ng serve
```

In your browser, open <http://localhost:4200/> to see the new application run.

You can configure the [application options](#) inside the package.json file and customize [branding](#) with LESS or CSS custom variables.

UPGRADING FROM ANGULAR 19 TO ANGULAR 20

Starting with version 1023.0.0, the Web SDK supports Angular 20. The following configuration changes are required before you can run the application:

- Run the command `ng update @angular/core@20 @angular/cli@20` to update Angular core and CLI to version 20.
- Update all `@c8y` dependencies to version `1023.x.x` in your `package.json`.
- Update `ngx-bootstrap` to version `20.0.2`.
- Update `@angular/cdk` to version `20.x.x`.
- Update TypeScript to version `5.9.3` or higher.
- Ensure compatibility for `Node.js`, `TypeScript`, `RxJS`: [Version compatibility](#).
- Follow the [Angular 20 upgrade guide: Updating to version 20](#).
- Adjust the `main.ts` and `bootstrap.ts` files according to the git diffs [mentioned below](#).

BREAKING CHANGES

The Web SDK version 1023.0.0 introduces multiple breaking changes.

Global time context API changes

The API for global time context has been completely redesigned. Custom widgets using the old Global Context API will no longer have access to global time context. To add global context support to a custom widget, you must integrate the new components:

- `GlobalContextConnectorComponent` - links your widget to the dashboard time context
- `LocalControlsComponent` - provides standalone time controls for your widget

ngx-translate upgrade

The `@ngx-translate/core` package has been upgraded to version 17.0.0. A separate `TranslateService` instance is now provided per plugin. If your code directly utilizes the `TranslateService`, you may need to adapt your implementation to account for this scoped service approach.

QueriesUtil type additions

The `QueriesUtil` class now includes type definitions. Depending on your usage of this class, you may encounter TypeScript compilation errors that need to be resolved by adding appropriate type annotations.

BulkSingleOperationsListModule removed

The `BulkSingleOperationsListModule` has been removed. Use `SingleOperationsListComponent` as a standalone component instead.

TRACK CHANGES BETWEEN RELEASES

Want to see exactly what changed within the code between versions? You can easily review the differences by examining the git diffs for each application:

- Administration: [v1022.47.5...v1023.0.0](#)
- Cockpit: [v1022.47.5...v1023.0.0](#)
- Device Management: [v1022.47.5...v1023.0.0](#)

C8Y COMMAND LINE TOOL (CLI)

! IMPORTANT

The `c8ycli` tool was deprecated and should not be used anymore with the 1019.0.0 release of the WebSDK. Use the default Angular CLI instead and add our `@c8y/websdk` library to it as shown in the [Getting Started Guide](#)

To support you with bootstrapping, running and deploying applications we have build a Command Line Interface. The tool is the successor of the `cumulocity-node-tools`. To avoid conflicts, it listens to the new command `c8ycli` instead of `c8y`. You can install it via npm:

```
npm install -g @c8y/cli
```

If you don't want to install the `@c8y/cli` globally you can also run it with the npx command. For example, you can scaffold a new project quickly by executing the following command:

```
npx @c8y/cli new
```

A new project creates a local `@c8y/cli`. You can run the project by navigating to its directory and executing the following commands:

```
npm install && npx c8ycli serve
```

The `serve` command starts a local development server. It supports two important flags:

- `-u` : The `-u` parameter specifies the Cumulocity instance to which all API requests are proxied. This means data is actually pulled from the configured Cumulocity instance.
- `-p` : The port to use. If not defined, port 9000 is used. If you have a server running on this port already, the command will fail. The application will then be served at the URL `http://localhost:<<port>>/apps/<<your-application-name>>/`. Tip: Click the URL in the terminal while holding the "control" key.

GENERAL USAGE

```
c8ycli [options] [command]
```

INFO

The commands must be executed from the root path of the project.

OPTIONS

<code>-u, --url <url></code>	The URL of the remote instance
<code>--version</code>	Provides version number
<code>-h, --help</code>	Provides usage information

COMMANDS

All the commands except of `new` take an array of [glob patterns](#). These will be resolved to directories or entry point manifests.

<code>new [name] [template]</code>	Creates a folder to start a new application or extend an existing one
<code>serve [options] [appPaths...]</code>	Runs local development server
<code>build [options] [appPaths...]</code>	Builds the specified applications
<code>deploy [options] [appPaths...]</code>	Deploys applications from the specified paths
<code>locale-extract [options] [srcPaths...]</code>	Extracts all strings for translation and outputs the .po files to defined folder

THE `new` COMMAND

The `c8ycli new [name] [template]` command creates an empty application or extends an existing application (Cockpit, Devicemanagement or Administration). To extend an existing application use the name of the existing application as `[name]` and `[template]` like this:

```
$ c8ycli new cockpit cockpit
```

TIP

The `c8ycli new` command has a `-a` flag which defines which package to use for scaffolding. This way you can also define which version of the app you want to scaffold, for example:

```
c8ycli new my-cockpit cockpit -a @c8y/apps@1004.11.0 will scaffold an app with the version 10.4.11.0
```

The `c8ycli new` command can also be provided on its own without the `[name]` and `[template]` options. In this case, follow the steps below to complete the process via the interface before the application is scaffolded.

Step 1:

```
? Enter the name of the project: (my-application)
```

The first step asks for the project name. If no project name is entered, the default value `my-application` is used.

INFO

This step can also be skipped if the name is provided in the initial command: `c8ycli new my-application`.

Step 2:

```
? Which base version do you want to scaffold from? (Use arrow keys)
> 1010.0.X (latest)
> 1011.X.0 (next)
> 1011.0.X
> 1009.0.X
> 1007.0.X
> 1006.0.X
> other
```

In the second step, the base scaffolding version must be selected. The interface will provide the CD version (latest), as well as older yearly releases. Additionally a version can be manually entered by selecting the `other` option.

Step 2 (other):

```
? Enter the desired version:
```

In this step, the desired version must be entered manually, for example, `1010.0.0`.

INFO

This question will appear only if `other` was selected in the previous step.

Step 3:

```
? Which base project do you want to scaffold from?
administration
application
cockpit
devicemanagement
hybrid
tutorial
```

In the final step, the base project to scaffold from must be selected.

INFO

This step will only show projects which are available for the selected version in Step 2.

APPLICATION OPTIONS

The application options can be defined with `--app.<option>=<value>`. These will be applied to all applications found with `[appPaths...]`.

```
--app.name="My Application"
--app.key=myapp-key
--app.contextPath=myapplication
```


INFO

The `--app.brandingEntry` option was deprecated in version 1021.0.0 (Angular 18). Use the Angular CLI `styles` configuration in `angular.json` instead. See [Branding your application](#) for the current approach.

WEBPACK OPTIONS

The webpack options can be defined with `--env.<option>=<value>`. These will be directly passed to the webpack configuration.

```
--env.mode="production"
--env.hmr
```

UPGRADING FROM ANGULAR 18 TO ANGULAR 19

Starting with version 1022.0.0, the Web SDK supports Angular 19. The following configuration changes are required before you can run the application:

- Run the command `ng update @angular/core@19 @angular/cli@19` to update Angular core and CLI to version 19.

IMPORTANT

Angular directives, components and pipes are now standalone by default. Specify `standalone: false` for declarations that are currently declared in an NgModule.

This also applies to module federation plugins, including plugins that use earlier versions of Angular, but are utilized in applications migrated to Angular 19.

- Update all `@c8y` dependencies to version `1022.x.x` in your `package.json`.
- Update `ngx-bootstrap` to version `19.0.2`.
- Update `@angular/cdk` to version `19.x.x`.
- `Node.js`, `TypeScript`, `RxJS`: [Version compatibility](#).
- Follow the [Angular 19](#) upgrade guide: [Updating to version 19](#).
- Adjust the `main.ts` and `bootstrap.ts` files according to the git diffs [mentioned below](#).

BREAKING CHANGES

The Web SDK version 1022.0.0 introduces multiple breaking changes.

Dashboard setting component as secondary router outlet

The dashboard setting component has been refactored to use a secondary router outlet in order to make these type of views hookable. This requires adding `rootContext: ViewContext.Dashboard` to the context dashboard routes.

The reason for this is that the new **Import/Export** tab has been added to the dashboard settings with the `hookTab` function. This generic solution allows adding tabs to named tab outlets. This specific tab allows to export dashboards to JSON files, import dashboards from previously exported JSON files and edit the dashboard in an editor. It allows to copy dashboards across applications. Example of how the tab and route for this tab can be added:

```
hookTab(
  [
    {
      label: gettext('Import / Export'),
      icon: 'input-output',
      priority: 5,
      path: [
        {
          outlets: {
            'dashboard-details': 'advanced'
          }
        }
      ],
      tabsOutlet: 'dashboardTabs'
    }
  ],
),
hookRoute([
  {
    path: 'advanced',
    loadComponent: () => import(...),
    outlet: 'dashboard-details',
    context: ViewContext.Dashboard
  }
])
```

In this example an additional tab is added to the dashboard settings, and the hook route allows to display a view for this tab. This is breaking change, as each component that uses context dashboards must have 'rootContext: ViewContext.Dashboard' in the route definition to make these settings tabs and views visible (even if no new tab was added). For example:

```
hookRoute({
  path: "home2",
  component: CockpitDashboardComponent,
  rootContext: ViewContext.Dashboard,
});
```

Login as separate application

The login flow has been changed. The Web SDK no longer includes built-in login functionality in each application. Instead, a separate login application now manages all authentication flows.

Web applications developed using Web SDK version 1022.0.0 or later will automatically redirect users to this standalone login application whenever authentication is needed.

This change benefits customers creating their own UI applications, as they no longer need to implement custom login flows. They can simply redirect users to the new login application. The login page has also been redesigned as part of this update, improving its usability and visual appeal.

Note: Customers who embed the UI within an iframe and require in-iframe login may need to modify their implementation to support this new login flow.

Deprecated Angular modules removed

A set of previously deprecated angular modules of widgets have been removed. These modules have been migrated to standalone components, so their modules became obsolete. The affected modules are: `CockpitLegacyWelcomeWidgetModule`, `CockpitWelcomeWidgetModule`, `DeviceControlMessageWidgetModule`, `HelpAndServiceModule`, `ImageWidgetModule`, `InfoGaugeWidgetModule`, `KpiWidgetModule`, `LinearGaugeModule`, `MarkdownWidgetModule` and `ThreeDRotationWidgetModule`.

TRACK CHANGES BETWEEN RELEASES

Want to see exactly what changed within the code between versions? You can easily review the differences by examining the git diffs for each application:

- Administration: [v1021.81.0...v1022.0.0](#)
- Cockpit: [v1021.81.0...v1022.0.0](#)
- Device Management: [v1021.81.0...v1022.0.0](#)

UPGRADING FROM ANGULAR 17 TO ANGULAR 18

Angular 18 is supported from version [1021.0.0](#). The following configuration changes are required before you can run the application:

- Update all [@c8y](#) dependencies to version [1021.x.x](#) in your *package.json*.
- Run the command `ng update @angular/core@18 @angular/cli@18` to update Angular core and CLI to version 18.
- Update `ngx-bootstrap` to version [18.0.0](#).
- Update `@angular/cdk` to version [18.x.x](#).
- The `brandingEntry` application option can no longer be used to customize the global style of your application. Instead, global styles should now be specified via [the mechanism Angular provides](#).

Migration steps:

1. Create a *styles.less* file in your *src/* directory with the following content:

```
// Import Cumulocity styles first
@import '~@c8y/style/extend.less';
// Add your variable overrides here (optional)
// @brand-primary: #your-color;
```

2. Reference this file in the `styles` array of your *angular.json*:

```
{
  "projects": {
    "your-app": {
      "architect": {
        "build": {
          "options": {
            "styles": [
              "src/styles.less"
            ]
          }
        }
      }
    }
  }
}
```

3. Remove the `brandingEntry` from your *cumulocity.config.ts* file.

Important: Always import `@c8y/style/extend.less` first, then override variables after. See [Branding your application](#) for details.

- [Node.js](#), [TypeScript](#), [RxJS](#): [Version compatibility](#).
- Follow the [Angular 18](#) upgrade guide: [Updating to version 18](#).

UPGRADING FROM ANGULAR 16 TO ANGULAR 17

Angular 17 is supported from version [1020.0.0](#). The following configuration changes are required before you can run the application:

- Update all [@c8y](#) dependencies to version [1020.x.x](#) in your *package.json*.
- Replace `import 'zone.js/dist/zone'` with `import 'zone.js'` in the *src/polyfills.ts* file.
- Replace all occurrences of `"browserTarget"` with `"buildTarget"` in the *angular.json* file.

- Run the command `ng update @angular/core@17 @angular/cli@17` to update Angular core and CLI to version 17.
- Update `ngx-bootstrap` to version `12.0.0`.
- Update `@angular/cdk` to version `17.x.x`.
- Remove any reference of `loginOptions` in the `src/main.ts` file. The `loginOptions` function is now called under the hood as part of the `loadOptions` function.
- Add the `@c8y/options` package as a devDependency in your `package.json`.

UPGRADING FROM ANGULAR 15 TO ANGULAR 16 AND NG CLI

Angular 16 is supported from version `1019.0.0`. With the latest release of version 1019.0.0, the Web SDK has transitioned to using `ng-cli`, marking the end of further development for `c8ycli`.

This switch aims to align with industry standards, making it more convenient for developers to work with the Web SDK. If you are an Angular developer looking to migrate your existing Web SDK project to `ng-cli`, the following step-by-step guide leads you through the process.

1. Create an ng-cli Project

First, download the appropriate version of `ng-cli` and scaffold your project using the `ng new` command. Make sure to use the name of your existing project during this process.

2. Add @c8y/websdk to Build an App

Integrate the `@c8y/websdk` into your `ng-cli` project by running `ng add @c8y/websdk`. Ensure you select the correct project type: `application` or `hybrid`.

3. Copy Over Source Files

Copy all your components, directives, modules, tests, and services files to the new `ng-cli` project. Also, migrate any non-standard files, such as assets, to the new project.

4. Align the package.json

Update your `package.json` file by verifying and comparing dependencies. Ensure that you retain the new dependencies while adding any custom dependencies. Update other properties from your original project, excluding the "c8y" property object. Change all scripts to use `ng` instead of `c8ycli` and rename any occurrences of "server" to "serve."

5. Align cumulocity.config.ts

Transfer all "c8y.application" properties from the `package.json` file to the new `cumulocity.config` `runTime` object. Format it as JavaScript and check for errors. Check if federation sharing is sufficient; otherwise, adjust as needed. Some options may need to be moved to `buildTime`.

6. Verify tsconfig Files

Review and align the `tsconfig` files according to your project's needs. You might need to disable certain strict configurations, such as `noPropertyAccessFromIndexSignature` or `noImplicitReturns`, which are enabled by default.

7. Install Dependencies and Verify

Install dependencies and ensure that all peer dependencies match. Update any dependencies that may require modification to run seamlessly with Angular 16.

8. Check the Angular Update Guide

As `ng-cli` uses Angular 16, check the Angular update guide to ensure compatibility. [Angular Update Guide](#)

9. Start it

Finally, attempt to run your application using `npm start` or `npx ng <<name-of-your-app>>`. Check for compilation errors and resolve any issues. For Angular 16, remove all `entryComponents` arrays in the modules, as they were deprecated in version 15 and removed in version 16.

Common Pitfalls

- Ensure that the rxjs `Subject` is correctly typed with `void` if it should only emit without a value.
- Remove all `entryComponents` from your code.
- Import `ApplicationOptions` from the `@c8y/devkit/dist/options` package, and consider importing only the types to avoid unnecessary bundling (`import type from '@c8y/devkit/dist/options';`).

UPGRADING FROM ANGULAR 14 TO ANGULAR 15

Angular 15 is supported from version `1018.157.0`. The following configuration changes are required before you can run the application:

- Update all `@angular/*` dependencies to `15.2.7`.
- Update `TypeScript` to version `4.9.5`.
- Follow the [Angular 15 upgrade guide: Updating to version 15](#).
- Use Node version `^14.20.0 || ^16.13.0 || ^18.10.0`.
- Delete `node_modules` and reinstall them.
- Token hooks were deprecated and replaced by function hooks. Check the table below to see to which function hooks the token hooks have been migrated to.

Deprecated HOOK Tokens and Their Replacements

Deprecated Token	Replaced By
<code>HOOK_TABS</code>	<code>hookTab</code>
<code>HOOK_NAVIGATOR_NODES</code>	<code>hookNavigator</code>
<code>HOOK_ACTION</code>	<code>hookAction</code>
<code>HOOK_BREADCRUMB</code>	<code>hookBreadcrumb</code>
<code>HOOK_SEARCH</code>	<code>hookSearch</code>
<code>HOOK_ONCE_ROUTE</code>	<code>hookRoute</code>
<code>HOOK_COMPONENTS</code>	<code>hookComponent</code>
<code>HOOK_WIZARD</code>	<code>hookWizard</code>
<code>HOOK_STEPPER</code>	<code>hookStepper</code>

INFO

For more information on the multi provider, refer to the [Cumulocity Developer Codex](#).

INFO

At this point, AOT (Ahead-of-time compilation) is not yet supported.

UPGRADING TO ANGULAR 14

Angular 14 is supported from version `1015.132.0`. The following configuration changes are required before you can run the application:

- Update all `@angular/*` dependencies to `14.0.6`.
- Update `TypeScript` to version `4.7.4` as [TypeScript 4.6](#) is now required by Angular.
- Follow the [Angular 14](#) upgrade guide: [Updating to version 14](#).
- Install new dev dependencies:
 - `"html-loader": "4.1.0"`,
 - `"style-loader": "3.3.1"`,
- Use Node version `14`.
- Delete `node_modules` and reinstall them.

INFO

AOT (Ahead-of-time compilation) is not yet supported.

INFO

If you use Visual Studio Code, make sure that in the Angular Language Service Plugin the option "Use legacy View Engine language service" is not selected.