@criticalmanufacturing/cli Documentation

Critical Manufacturing SA

Table of contents

1. About cmf-cli

cmf-cli is a Command Line Interface used for Critical Manufacturing developments.

1.1 Use cmf-cli to . . .

- Scaffold a new repository
- Generate new package structures
- Adapt packages of code for Critical Manufacturing MES
- Manage multiple versions of packages and package dependencies
- Create packages that can be used by any developer or customer
- View the package tree
- Restore packages for local development
- Assemble a release bundle

and a lot more!

1.2 Getting started

To get started with cmf-cli, you need to use the command line interface (CLI) to install cmf-cli. We look forward to seeing what you create!

2. Installation

To be able to install cfm-cli, you must install Node.js and the npm command line interface using either a Node version manager or a Node installer. **We strongly recommend using a Node version manager like nvm to install Node.js and npm.** We do not recommend using a Node installer, since the Node installation process installs npm in a directory with local permissions and can cause permissions errors when you run npm packages globally.

```
npm install --global @criticalmanufacturing/cli
```

2.1 Checking your version of cmf-cli and Node.js

To see if you already have Node.js and npm installed and check the installed version, run the following commands:

```
node -v
cmf -v
```

3. Plugins

The Critical Manufacturing cli is designed with a plugin system for extensibility. In the future, it will be possible to search for plugins straight from cli. Check issue #11 for progress.

In the meanwhile, some plugins are already in development. Here follows a non-exaustive plugin list:

- Portal SDK - command line tools to interact with the Critical Manufacturing Customer Portal

4. Telemetry

4.1 Telemetry implementation

Basic telemetry currently only tracks the CLI startup and logs:

- CLI name and version
- latest version available in NPM
- if the CLI version is stable or testing
- if the CLI is outdated

no identifiable information is collected in basic telemetry

However, any user can optionally enable extended telemetry, which might help with troubleshooting. **Extended telemetry includes identifiable information** and as such should be used with care. This includes the basic telemetry, plus:

- for the version (startup) log, it also includes
- current working directory
- hostname
- IP
- username

It also tracks and logs the package tree if, for any command, the tree must be computed. This includes all of the above information plus the package name, for each package in the tree.

Enabling telemetry can be done via environment variables:

- `cmf_cli_enable_telemetry` - enable basic telemetry. If this is off (the default), no telemetry will be collected, even if extended telemetry is on. To enable, set to `true` or `1`, do not set or set to `false` or `0` to disable.
- `cmf_cli_enable_extended_telemetry` - enable extended telemetry. Note the above warnings regarding the impact of keeping this on. To enable, set to `true` or `1`, do not set or set to `false` or `0` to disable.
- `cmf_cli_telemetry_enable_console_exporter` - also print the telemetry information to the console. This is for auditing or troubleshooting as it makes the console output extremely verbose
- `cmf_cli_telemetry_host` - specify an alternate telemetry endpoint (if you're hosting your own)

5. assemble

5.1 Usage

```
cmf assemble [options] [<workingDir>]
```

5.1.1 Arguments

Name	Description
`<workingDir>`	Working Directory [default: .]

5.1.2 Options

Name	Description
`-o, --outputDir <outputDir>`	Output directory for assembled package [default: Assemble]
`--cirepo <cirepo>`	Repository where Continuous Integration packages are located (url or folder)
`-r, --repo, --repos <repos>`	Repository or repositories where published dependencies are located (url or folder)
`--includeTestPackages`	Include test packages on assemble
`-?, -h, --help`	Show help and usage information

6. Scaffolding

6.1 Pre-requisites

Though `@criticalmanufacturing/cli` runs with the latest `node` version, to run scaffolding commands the versions required by the target MES version are **mandatory**.

For **MES v10**, the recommended versions are:

- latest node 18 (Hydrogen)
- latest npm 9 (should come with node)

For MES v8 and v9, the recommended versions are:

- latest node 12 (Erbium)
- latest npm 6 (should come with node)

Apart from those, scaffolding also needs the following dependencies:

```
npm install -g windows-build-tools
npm install -g gulp@3.9.1
npm install -g yo@3.1.1
```

For **MES v10**, you will also need angular cli

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

6.1.1 NuGet and NPM repositories

Rarely changing information, possibly sensitive, like NuGet or NPM repositories and respective access credentials are considered infrastructure. More information on how to set up your own is available at Infrastructure

6.1.2 Environment Config

A valid MES installation is required to initialize your repository, either installed via Setup or via DevOps Center. For the Setup: - in the final step of the Setup, click Export to obtain a JSON file with your environment settings For DevOps Center: - Open your environment and click Export Parameters

Both these files contain sensitive information, such as user accounts and authentication tokens. They need to be provided to the `init` command with:

```
cmf init --config <config file.json> --infra...
```

6.2 Scaffolding a repository

Let's start by cloning the empty repository.

```
git clone https://git.example/repo.git
```

Move into the repository folder

```
cd repo
```

For a classic project example, check the traditional structure documentation.

For more advanced structures, you'll probably be using Features.

6.3 Pipelines

By default, our scaffolding doesn't provide any built-in CI/CD pipelines, giving you the flexibility to choose any tool/platform that suits your needs.

However, we can share as a reference our internal process:

6.3.1 Pull Requests (PRs)

For each changed package, we run the command `cmf build --test`, which compiles the package and runs unit tests if available, comparing with the target branch.

We consider a package as "changed" when any file is modified inside a folder with a cmfpackage.json file.

An alternative is to run `cmf build --test` for all packages.

6.3.2 Continuous Integration (CI)

After merging code into the main branch, we perform the following steps:

1. Run `cmf build --test` to ensure successful building of all packages and passing of unit tests.
2. Run `cmf pack` to generate a package that can be installed via DevOps Center or Critical Manufacturing Setup.

6.3.3 Continuous Deployment (CD)

Traditional (Windows VMs)

1. Follow the instructions in the documentation
2. In the Package Sources step, add the path where your packages are located.

Containers

1. Follow the instructions in the documentation
2. Copy the generated packages to the folder defined in volume **Boot Packages**.
3. In the Configuration > General Data step, set the Package to Install as `RootPackageId@PackageVersion`