Deployment

Overview

Connect IoT supports multiple architectures to best adapt to the integration scenario. The architecture of the application is based on having a main or parent process: the `Automation Manager`, which is responsible for spawning all the subprocesses of ConnectIoT.

Each process that Connect IoT runs is a Node.js process. The smallest amount of Node.js processes that Connect IoT requires to be fully functional is four. It requires the parent process Automation Manager, the `Automation Monitor`, then an Automation Controller and an Automation Driver. Connect IoT also requires a registry with the packages required to run. These packages may be in a directory or in an NPM registry. They can be in a directory to which the MES can access, leading the `Automation Manager` to request the packages from the MES, removing the need for direct access from the manager to the registry.

The Automation Manager also generates logs and can write process data to files using a mechanism called persistency. See Automation Manager Logging Config for more information.

System Requirements

The system requirements vary according to which drivers your implementation is using. Some of them have specific features that are described in the Connect IoT Requirements.

Regarding hardware, it depends on what the Automation Manager is being used for and the level of traffic and processing that is expected. The hardware needs also vary by driver, with some being more resource intensive than others. It is accepted that the bottleneck is typically RAM allocation. See Hardware Requirements for more information.

Creating an Automation Controller Instance

The Automation Manager can be deployed without any controller instance associated. In this case it will boot up and will query the MES for an instance, and if no instance is found it will wait for an instance to be created. To have a running integration we must associate the Automation Manager to an Automation Controller through an instance. See the How to Create a Controller Instance tutorial for more information.

Using the MES GUI

The first interaction most Connect IoT users have with the application is through the Automation Manager GUI, where you can download a zip file with all that you need to run the Automation Manager. The download will also generate a new authentication token for the user selected as integration user for the download. See the How to: Download the Automation Manager as a zip for more information.

It is important to note that the GUI allows the configuration of each and any particular manager. Also, that configuration can be changed for all Automation Managers with the defined configuration entry in `/Cmf/System/Configuration/ConnectIoT/ConfigurationTemplate/`, or alternatively in the specific Automation Manager. This may be very useful to have pre-configured all the configurations for what is relevant in your case.

The zip file that is generated contains the following:

```
├── scripts/
    ├── InstallService.ps1
```

```
    ├── StartConsole.bat
    ├── ...
├── service/
└── src/
    ├── config.json
    ├── npm-shrinkwrap.json
    ├── package.json
    └── README.md
```

Files

Regarding versioning and metadata, that is assured by the `npm-shrinkwrap.json` and `package.json`. The `README.md` contains the documentation on the Automation Manager runtime. The `config.json` controls all the key configuration for the `Automation Manager` and is a key part of the deployment process. All the fields of the `config.json` are described in the documentation under Help IoT Runtime Components Configuration.

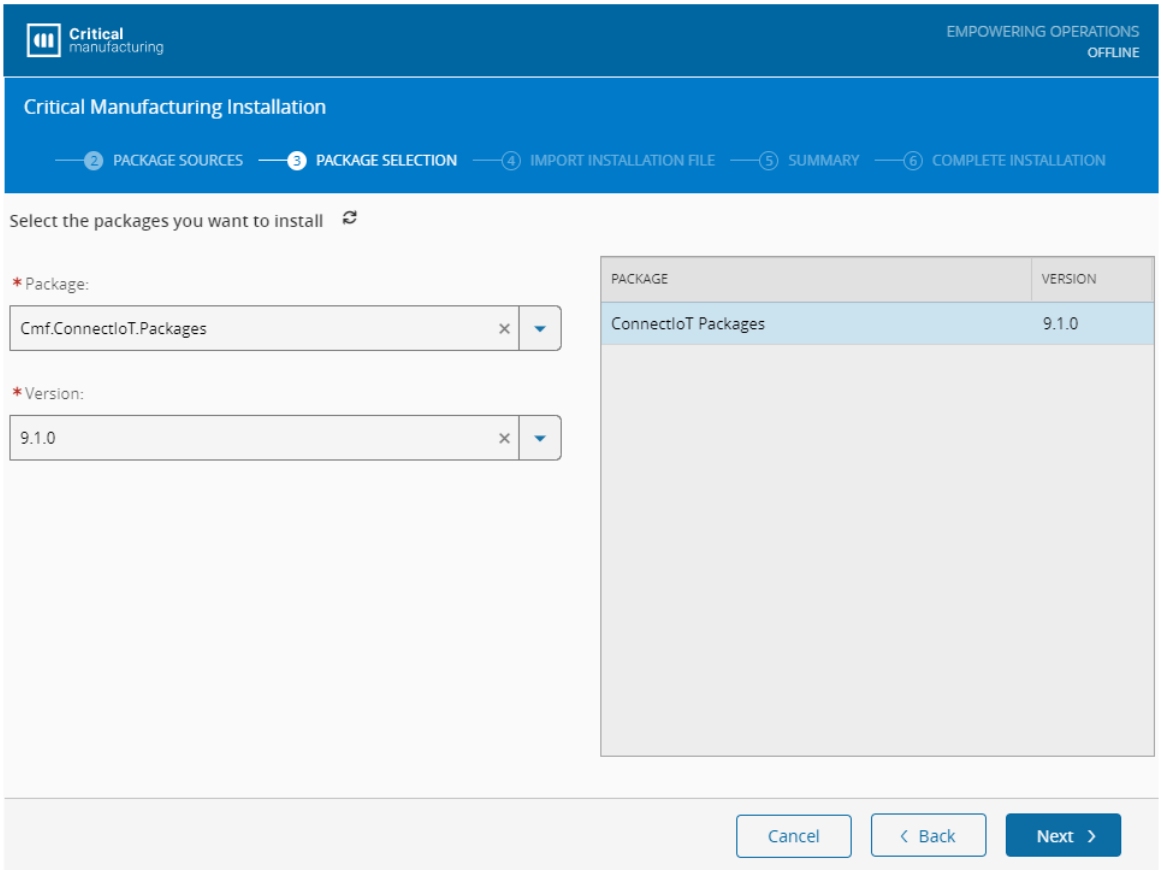
Folders

The **src** folder is where everything needed for the `Automation Manager` to run is located. The scripts and service folder are utilities that are included in the zip file to allow easy usage. The **service** folder has everything needed for the `Automation Manager` to be installed as a Windows Service. The **scripts** has the `StartConsole.bat` batch script which enables running the `Automation Manager` as a console application, while also having the `InstallService.ps1` script that allows you to install the Automation Manager as a Windows Service.

Using the Setup

The Critical Manufacturing MES Setup allows the deployment of an `Automation Manager` or several through a GUI. This method is useful if you want to deploy several managers, making the manual download through the GUI impractical if you prefer to have a simple GUI abstraction.

You can read more about all relevant setup configurations in the Help Connect IoT Installation page. The process involves selecting the `Cmf.ConnectIoT.Packages` deployment option, which enables you to configure all relevant `config.json` fields directly through the GUI. It also supports installation as a Windows Service.



Running as a Console

This method is very useful when developing or troubleshooting an installation. It consists on running the process as a console by using the `StartConsole.bat` script. When you run as a console, all the logs are aggregated as console output to make it easier to spot errors that are occurring.

Productive Deployment

There are several different ways to have a productive deployment of the `Automation Manager`, but before starting with the differences, let's start with the similarities.

In a High Availability scenario, if a manager goes down, another must start to take its place. It's important that whenever a new process starts, it doesn't start from scratch and that it inherits the context that the previous manager was working with. This is very impactful when thinking about implementations that depend on a persistent layer for tracking or data collection. If Manager 1 is tracking Material A and for some reason Manager 1 goes down, it's important that the new Manager 2 has a notion of Material A.

> **Warning**
>
> It is strongly suggested that if using persistency layer for any logic, the Storage/Persistency layer be stored in a shared folder accessible to all manager nodes.

Logging

It is also important to configure the logging appropriately; this is critical for very verbose drivers and implementations. The retention times, rollovers, verbosity and others can be configured. If retaining a

significant volume of logs for an extended period is important, consider offloading them to a shared folder to mitigate the risk of data loss. See Help Logging Configuration for more information.

Certificates

It is required that a certificate is defined for the Automation Manager. The certificate must be placed (or path to the certificate) in an environment variable `NODE_EXTRA_CA_CERTS` Certificate Troubleshooting. For development purposes the flag `NODE_TLS_REJECT_UNAUTHORIZED` can be set to `0`, indicating that the certificate will be used exactly as received.

> **Note**
>
> For more information, see Extra CA Certificates Node.js & Node.js original issue.

Running as a Windows Service

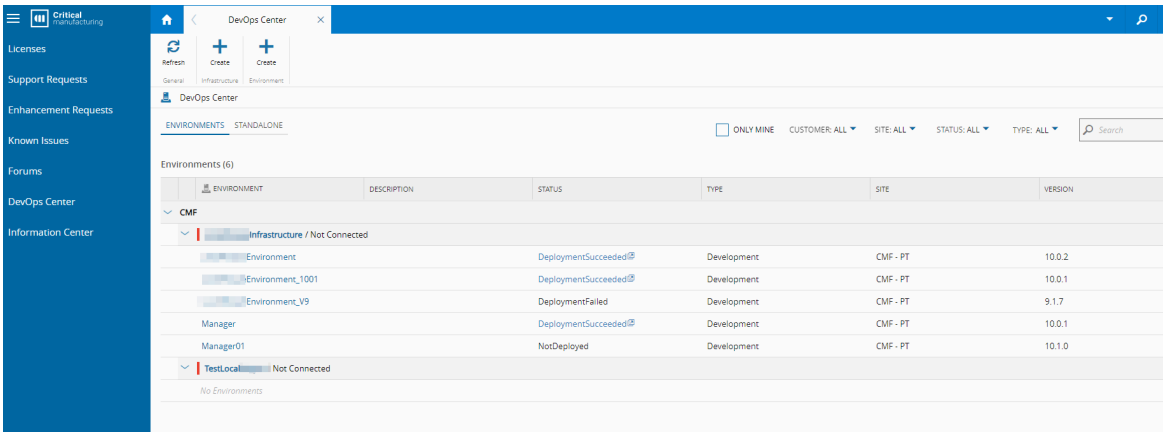
This was traditionally the main approach of a production environment and remains very much a possibility. Some drivers (like OIB and OPC-DA) are Windows-based, either because they are built on .NET Framework or due to the use of the DCOM, both of which are Windows exclusive. As explained before, both the download action in the MES GUI and the Setup provide easy ways to install the `Automation Manager` as a Windows Service Library used for Creating Node.js Windows Services. It is important to highlight that the user defined to run the Windows Service is also very important, as the permissions, or lack thereof may cause side effects. Keep in mind also, that if the password has expiration this will impact the Automation Manager, that is running that user.

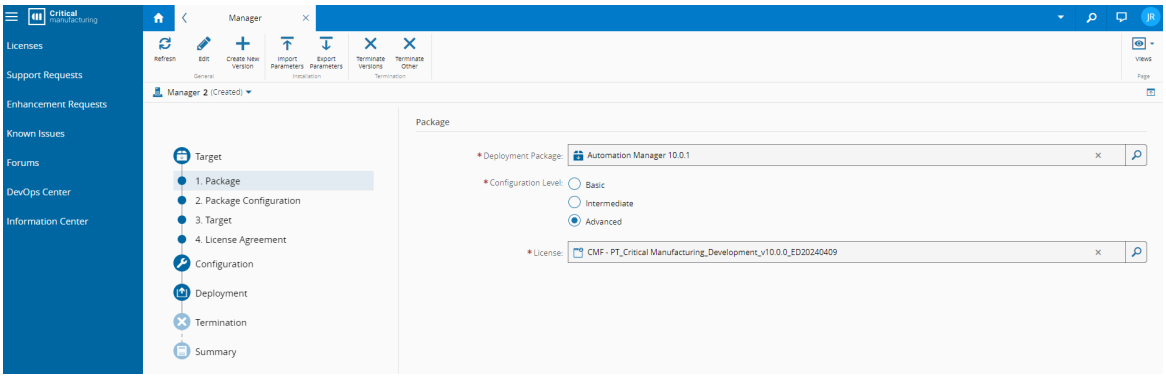
High Availability Running as a Windows Service

Achieving High Availability using Windows Services is possible using Windows Server Failover Clustering (WSFC). This Microsoft technology creates a cluster of different machines, each with the Windows Service installed. The Failover Cluster guarantees that one - and only one - instance runs at any given time across all available nodes. If a node goes down (that is, a Windows Service is shut down), it will start a new Windows Service in the other node. Since all nodes share the same access, configurations, and so on, the transition is smooth and seamless when the failover occurs.

Using the DevOps Center

Critical Manufacturing created a portal to be able to remotely perform all deployments, which is the DevOps Center. It also supports different orchestrators for deployment and different deployment targets deployment targets.





Using the Agent

Assuming there is an agent running in your machine, that agent will have the possibility to deploy Automation Managers by using the Infrastructure, deploying new `Environments`, MES Example, for each Automation Manager.

In order to deploy an **Automation Manager** select the Deployment Package and the Automation Manager for the version you require. Then select the target for the deployment. In the **General Data**, specify the Manager Id that you are deploying and the manager configuration related to system. For more information, see Help System Configuration.

Regarding service resources, the Automation Manager will just run one instance of itself, so only one replica is needed. Under **Volumes**, specify the locations for logs, persistent storage, and the Connect IoT packages

For certain drivers it is also important to create volumes which is the case for all drivers that require file monitoring and processing. The DevOps Center allows you to set those mounts through the GUI as well. Currently, the only way to set open specific ports or ranges is by editing configuration yaml in the manager deployment.

Using the Agent, the agent will automatically deploy the stack.

Standalone Installation

The Standalone Installation works in the same way as using the agent, but it will generate a zip file that can be manually executed. In other words it is the previous method but with a manual deployment.

The zip file that is generated depends on the `Target` selected, but will consist on scripts to deploy and remove the stack and then the needed configuration yaml files.

> **Enabling Inbound Traffic**
>
> When using a driver that requires an external system to connect to it, instead of it connecting to an external system, a port will need to be opened for this access. Currently, the DevOps Center does not support setting it through the GUI. In order to do this, it will require a manual change in the configuration yaml, or in the system that is managing your stack, i.e portainer to open the port.