# Module Specification v0.30.0

CSP – MCERTS – Configuration Service

Intrasoft International 01/08/2017

# TABLE OF CONTENTS

# OVERVIEW

The MCERTS (CSP) platform is a collaboration platform for secure exchange of information related to Cyber Security. This document describes the configuration service Distribution Module (a specific artefact used to distribute new versions of software for the MCERTS platform), it's usage and contents and the supporting software. It *does not* detail the Configuration Service itself or it's API used for registration, status and update distribution.

## DOCUMENT HISTORY

| Version | Who | Changes implemented | Pages | Date |
|---|---|---|---|---|
| **0.10** | TA | Initial Version | All | 1/8/2017 |
| **0.15** | TA | Updated version in line with CSP beta phase | As needed | 1/10/2017 |
| **0.20** | TA | Updated with comments from OA/KS | As needed | 26/11/2017 |
| **0.30** | TA | Final changes w/cspinstaller 1.10 | As needed | 5/2/2018 |
| **0.50** | TA | Updated for inclusion in M2 Tender doc | Minimal | 28/6/2019 |

# DISTRIBUTION MODULE SPECIFICATION

This document describes the distribution module format for the Configuration Service. The distribution module contains an artefact (a CSP application) together with its dependencies (if any) and instruction/manifest documents that indicate potential actions on deployment and/or restart.
It is necessary to understand the concept of the module vs. the container and the process itself. According to Docker best practices (https://github.com/FuriKuri/docker-best-practices) the containers must contain one process and perform one task:

- Module – the artefact that has all necessary bootstrap and setup information to start a container. It also may contain a container "image" produced via a *docker save* command.
- Container – the "sandbox" in which the process runs. The container should have only one process at any given time – e.g. it can start with an "entrypoint.sh" script, that performs some magic and then starts the main process.
- Process – a monitored, sandboxed, controller process living within a container

There is a default global module that contains all master images. This module is the first module installed and the dependencies of it are all together in one archive, for convenience.

## DISTRIBUTION MODULE CONTENTS

The distribution module is a ZIP file. This file is uploaded in the Configuration Service, Central, and then it becomes available per CSP instance.
The contents of the zip file are listed below:

| File | Mandatory | Description | First Appeared | Notes |
|---|---|---|---|---|
| **<image>.tar.bz2** | No | Zero or more TAR archives shall be present in the ZIP file. The system will use the "docker load" command to upload into the docker system the contents of each file, hence creating the container images. Possible acceptable formats: "bz2", "tar". | v1.0 | It is not required to have a docker TAR archive although in many cases this is the recommended (self-contained) approach. |
| **first-time.sh** | No | If such file is present, it will be executed once, at the installation phase of the system. It is the responsibility of the shell author to verify that the script executes in the environment. Variables for various configuration | v1.0 | Usage of it should be avoided; it is preferred that the container on start checks that necessary initialization has occurred or not and acts accordingly; any required volumes should already |

| File | Mandatory | Description | First Appeared | Notes |
|---|---|---|---|---|
| | | elements are available in an ".env" file and the author should source them if necessary, from the current directory.<br>Any Docker Volumes that are used by docker-compose file should be created here. | | be mounted at the correct security level (rw/ro). Also check "sh-first" element in manifest v1.1 |
| **last-time.sh** | No | ** SUPPORTED: v1.1** This file will be called immediately before the module deletion, in case some housekeeping is necessary | v1.1 | The "sh-last" element controls the name of the file, if it is different than the expected one. |
| **docker-compose.yml** | No | A compose file indicates this is a service. The system takes into consideration the module priorities and will start the services at the right order. | v1.0 | The docker-compose file contains the necessary orchestration for the services, their capacity (CPU,RAM) and their networking requirements. Pay special attention to the CSP internal network that is defined as "external" at the end of this compose file for things to work. |
| **env.json** | Yes | ** SUPPORTED: v1.1** *File; this file will be appended to the **site global environment file*** | v1.1 | The env.json file is used as input by the installer (using the j2 template system) in order to create ENV Variables and vhost files.<br><br>The structure of env.json file is described in a later section, see Env.json Structure |
| **site-config/** | No | ** SUPPORTED: v1.1** Directory; the directory should contain a "module.conf.j2" file, that can be used to generate a | v1.1 | The installer will execute the template vs. the env.json provided, to create an apache |

| File | Mandatory | Description | First Appeared | Notes |
|---|---|---|---|---|
| | | ".conf" file to populate the vhost configuration for this module. | | configuration file for vhost configuration |
| **Any other file** | No | All files shall be extracted in a specific directory and all executions shall be relative to that (working) directory. Such files may be referenced in the compose or first-time.sh files. | v1.0 | |
| **manifest.json** | Yes | A file indicating information about the module. Currently this file must exist and the content should be *at least* the following json text (later versions may have more content):<br><br>`{"format":1.0}`<br><br>For valid and future versions of this file, please check the relevant section: "Manifest versions" | v1.0 | The CSP Configuration service should reject modules without this file. In the future, the file may contain further metadata entries. |
| **proc_ready.source** | No | A file to include a function prototype as below:<br><br>`function is_ready() { return 1; }`<br><br>if the file is present, it will be extracted in the working directory and sourced. The method shall be invoked *immediately after the container has been started.* | v1.0 | This will allow more fine-grained control over the start process and flexibility to decide whether the container is available. The function should return immediately and will be called multiple times within 10 second delays to ensure that the process(es) within the container are ready to function.<br>Expected return values:<br><br>• 0: not ready, wait<br>• 1: process is ready |

| File | Mandatory | Description | First Appeared | Notes |
|---|---|---|---|---|
| | | | | • 100: error detected |

## MANIFEST VERSIONS

The following table describes the available manifest versions and their support lifecycle:

| Version | Version Specification | Description | Notes | Introduced | EOL |
|---|---|---|---|---|---|
| **1.0** | `"format":1.0` | The first manifest.json format has this version; the installer just verifies the presence of this file to accept the module. If the file and content are not present the manifest file is rejected | No special content exists in the file. | 1/Aug/2017 | **1/May/2018** |
| **1.1** | `"format":1.1` | The 1.1 version makes template generation more configurable; in particular the modules may specify own vhost entries and configuration variables | A number of fields are introduced; see section below | 1/Dec/2017[1] | - |
| **1.2** | `"format":1.2` | The 1.2 version shall include UI configuration elements for more dynamic "at installation time" data entry for elements used in the module. | A number of fields are introduced; see section below | TBC | TBC |

## MANIFEST 1.1 DESCRIPTION

The following table describes the manifest fields and their possible values:

| Field | Field type | Description | Notes | Mandatory |
|---|---|---|---|---|
| **format** | Double; fixed value; **1.1 for this manifest** | Fixed – the value is required for detection of a 1.1 manifest file | N/A | Yes |

[1] Subject to deployment of the Configuration Service Client module on existing CSPs

| | | | | |
|---|---|---|---|---|
| **sh-first** | String; "first-time.sh" | The field contains the name of the script that initializes the module. By default it is called "first-time.sh". | The existence of the file is not mandatory and hence this field is also not mandatory | No |
| **sh-last** | String; "last-time.sh" | The field contains the name of the script to clean up just before deletion of the module. | This file is not mandatory; if present, the system shall execute before deletion and clean-up for this module should be performed at that stage. | No |

## MANIFEST 1.2 DESCRIPTION

The following table describes the manifest fields and their possible values:

| Field | Field type | Description | Notes | Mandatory |
|---|---|---|---|---|
| **format** | Double; fixed value; **1.2 for this manifest** | Fixed – the value is required for detection of a 1.2 manifest file | N/A | Yes |
| **sh-first** | String; "first-time.sh" | The field contains the name of the script that initializes the module. By default it is called "first-time.sh". | The existence of the file is not mandatory and hence this field is also not mandatory | No |
| **sh-last** | String; "last-time.sh" | The field contains the name of the script to clean up just before deletion of the module. | This file is not mandatory; if present, the system shall execute before deletion and clean-up for this module should be performed at that stage. | No |
| **config** | Map; contains the data array for configuration | A mapping between expected environment variables and their metadata | If such map is present, the system shall generate one or more entry fields to request values from the user. | No |
| **config.data** | Array; objects | A list of the objects that information should be provided for; | Same as above; | No |
| **config.data[x]** | An object | The following fields are available for configuration:<br>• key – the name of the variable<br>• descr – the description to be shown to the user | Types of fields:<br>• Integer<br>• Double<br>• String<br>• Password | No |

| | | | | |
|---|---|---|---|---|
| | | • type – the type of field expected<br>• required – true or false | | |

Example for 1.2 manifest:


```
{
  "format":  1.2,
  "sh-first":"first-time.sh",
  "sh-last":"last-time.sh",
  "config":{
    "data":[
      {
        "key":"SMTPHOST",
        "descr":"The host DNS name of the SMTP server",
        "type":"String",
        "required":true
      },
      {
        "key":"SMTPPORT",
        "descr":"The port of the SMTP server, 587 or 465",
        "type":"Integer",
        "required":true
      },
      {
        "key":"SMTPUSER",
        "descr":"The account username for authentication",
        "type":"String",
        "required":true
      },
      {
        "key":"SMTPPASS",
        "descr":"The account password for authentication",
        "type":"Password",
        "required":true
      }
    ]
  }
}
```


## ENV.JSON STRUCTURE

```
{
  "services": [
    {
      "internal_name": "service1",
      //Internal Service name. It will be used for the creation of the internal domain i.e,
service1.local.csp-domain
      "external_name": "service1",
      //External Service name. It will be used for the creation of the public service domain i.e,
service1.csp-domain
      "version": "0.3.0-SNAPSHOT",
      // Application Version
      "docr_name": "service1",
      // This should be exactly the same with the container name on the docker-compose file
      "docr_port": "9000",
```

```
      // The exposed docker port
      "protocol": "http",
      // The protocol that is used for communication with the service, default is http
      "central_only": false,
      // True if the service is executed only in central node
      "base_path": "/",
      // The base path of the service
      "skip_reverse_proxy_paths": [
        "/policies/",
        "/api"
        // A list of contexts that should not be proxied
      ],
      "mutual_ssl": {
        "external": false,
        // true for public APIs, False for UIs
        "internal": true
        // true for all internal Endpoints (Only for the case of Elastic search is False)
      },
      "agent": true,
      // true if the external service domain should be protected by an OpenAM Agent (Default is
true)
      "paths": {
        // Endpoints that should be used by other services
        "circles": "/api/v1/circles",
        "teams": "/api/v1/teams"
      },
      "env_properties": {
        // Any other configurable ENV properties that should be used by the service
        "EVENT_APPS": "misp",
        "THREAT_APPS": "misp",
        "INCIDENT_APPS": "rt,intelmq",
        "VULNERABILITY_APPS": "taranis,misp",
        "ARTEFACT_APPS": "misp,viper"
      }
    }
  ]
}
```

## CREATING A DISTRIBUTION MODULE

The general "workflow" for creating a distribution module should be as follows:

- Creating a new application or transforming a traditional application to a Docker native one
- Doing necessary adjustments to work with CSP Docker environment
- Verifying environment and connectivity, working with csp-apache and j2 template for vhosts
- Packaging as a distribution module

For each of the above steps, more information is presented in subsequent sections.

### CREATING OR TRANSFORMING AN APPLICATION

A "Docker native" application is an application that can be created to self-contained images and executed by runtime containers. If an application already exists, a good starting point in defining the strategy to transition to a Docker native application is this link: https://goo.gl/FBkUKP

In principle, all applications can be executed in a Docker environment, however complex they are. The fundamentals in "Docker-speak" are:

- An **image** is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files
- A **container** is a runtime instance of an image—what the image becomes in memory when actually executed. It runs completely isolated from the host environment by default, only accessing host files and ports if configured to do so.

To complete a transformation from a MTA to a Docker application, the following layers must be defined:

- Image – the definition of requirements for a single process. The image is a runtime filesystem layer (or collection of layers) that gets instantiated (as an container). Containers, are the runtimes created out of an image, are running in isolation in a docker "sandbox". To have access from the outside world, the image designer will define which ports can be *exposed* and which internal networks should exist or be created.
- Services – the collection of one or more images, to be run as containers, together with their volumes and networks.
- Stack – a complete manifest of an application, as a collection of services.

For more information on the above, must-read is the "Get started" guide:
https://docs.docker.com/get-started/

## CREATING WITHIN CSP

Having the knowledge of the technology is important, but more important is to understand the CSP specifics and environment in which the services created get deployed and executed.

- Operating System: Alpine Linux 3.6.2 or later, security enforced.
- Memory: the memory on the system is limited, and in production systems the machine contains 48GB of *total memory*.
- Disk: the disk allocated for CSP in the production systems is ranging from 40GB to 400GB per application, not counting application logs. All logs should be ingested to Elastic Search for further processing, so logs should not be kept within containers. Also, writing to console takes disk space, and as such this should be minimized.
- Storage locations:
    - /opt/csp/modules/<module name and hash> - the directory where the module is to be expanded. Each module version will use a different directory so do not expect anything to persist within. This directory will contain all items present within the Module when the module is deployed. Only add the following in the
    - .env – the global environment. This environment is available to every module and each module can add to this via "env.json" configuration
    - vhost configuration – adding vhost configuration is possible. The configuration file will be copied to the vhosts configuration of the system before system restart.

## HANDLING PERSISTENCE

It is vital to understand that Containers, being instances of Images, do not have persistent storage by default; they operate within the filesystem of the composed image layers. To be able to handle persistence and deal with the use case of "re-install" the image design should include a persistent external volume. Various articles exist on the matter; some are shown below:

- https://docs.docker.com/engine/admin/volumes/volumes/
- http://container-solutions.com/understanding-volumes-docker/

An external volume should be created by the "first-time.sh" script. The volume created can then be mounted via standard compose configuration, on the "volumes" section. Don't forget to explicitly mark as external (external: true) so the volume is attached and not created internally (which will not survive).

## INSTALLATION PROCESS

The module is assigned to a CSP for deployment; the admin user needs to proceed with download and installation steps. When the module is installed, the following steps occur in the order below. Note that the version indicated shows time the step becomes valid, and *is supported from that version onwards*.

- [v1.0] **Module creation**: The module hash and module name is used to create a new directory to expand the module contents.
- [v1.0] **Module extraction**: The system will use the supplied module archive and extract contents (subdirectories also) to the directory.
- [v1.0] **Docker image installation**: For every archive ending with ".tar" or ".bz2" the system will use the "docker load" command to insert as an image to the system. The module maintainer should try to use CSP approved image layers as parent layers in order to reduce the attack surface.
- [v1.0] **Global Environment**: (".env") is copied to the directory
- [v1.2] **Module UI setup**: if the v1.2 manifest contains configuration, at this stage, the user will be prompted to enter configuration variables to continue with installation. The installation will not proceed unless values have been provided. These values will be saved on the "env.json" file, appended at the end *if the file already exists in the module directory*.
- [v1.0] **Module setup**: If "first-time.sh" exists, the system will execute the script. The script *may source ".env"* to use global defined variables. This script should do initial setup including any required local volumes.
- [v1.1] **Module environment** – the system will append "env.json", if configured, to the global ".env" and copy across all modules at this point
- [v1.1] **Module vhost** – the system will copy all files ending in ".conf" in the apache vhost configuration
- [v1.0] **End of installation**: the system marks the module as installed.

## DELETION PROCESS

The module may be deleted at any point – functionality and services/modules that depend on this module will be affected. The following steps occur in the order below. Note that the version indicated shows time the step becomes valid, and *is supported from that version onwards*.

- [v1.1] **Module last time cleanup:** the system will execute the "last-time.sh" file (or whatever configured name) to clean up resources
- [v1.0] **Module deletion**: the system deletes all files found in the directory of the module
- [v1.0] **End of deletion:** the system marks the module as DOWNLOADED and removes it from the services list.