# Working Offline with Cloudify 3.4

## Overview

The expression "working offline", when it comes to Cloudify, refers to the practice of installing and configuring Cloudify Manager in a *self-contained* manner; that is, all resources required by Cloudify Manager — ranging from the bootstrap process to workflow execution — are contained within the Cloudify Manager, rather than being retrieved from any other source, be it an internal network or (more commonly) the public network.

## Why Work Offline?

Having a Cloudify topology work offline bears with it, first and foremost, the benefits of stability and security. Consider that access to any network — especially the public network — embodies a certain level of risk. For example, if Cloudify Manager requires access to the public network for the purpose of downloading YAML files (or plugin packages), then outbound access to the public network needs to be allowed. For certain organizations, such access is strictly prohibited, be it at the corporate level or, at times, even legally; for other organizations, providing access to the public network raises audit compliance issues (which may or may not be eventually accepted). Regardless, though, providing outbound access to the public network can always be perceived as an added security concern, which should (or must, depending on the circumstances) be avoided.

## General Considerations

### Cloudify's Global `types.yaml` file

Included with each release of Cloudify is a YAML file containing Cloudify's basic types. This file is usually referred-to by the name `types.yaml`, and is included in the `cloudify_manager` package (for example: https://github.com/cloudify-cosmo/cloudify-manager/blob/master/resources/rest-service/cloudify/types/types.yaml; note that this file may change between versions).

Certain types in that file make references to scripts that are hosted on GitHub; this is documented in **CFY-6335** - types.yaml refers to GitHub resources **OPEN** . Until the issue is resolved, you will have to maintain an edited copy of this file and use it instead of the out-of-the-box one throughout the instructions in this document.

## The Bootstrap Process

The bootstrap process involves using the Cloudify CLI to install a manager blueprint. Behind the scenes, the CLI runs in what's called "*local mode*" (AKA "*cfy local*"), installing a manager blueprint in much the same way that a Cloudify Manager runs the *install* workflow on any application blueprint. There is nothing special about a manager blueprint from that perspective.

The bootstrap process in Cloudify 3.4 has a few online prerequisites:

| Online Prerequisites | Must Be Available For... |
|---|---|
| Any YAML file imported by the manager blueprint | The machine where the bootstrap process is executing from |
| Any other YAML files specified via the `dsl_resources` manager blueprint input | |
| Plugin packages, denoted by the "`source`" field of each and every plugin declaration contained within the manager blueprint | |
| The Cloudify Manager resources package, which is a `tar.gz` file containing all packages that comprise Cloudify Manager and its dependencies | The machine that is going to host the Cloudify Manager |

Let's look at how to prepare the bootstrap client machine (that is, the machine where the bootstrap process is executing from; for brevity, we will refer to it as the *CLI machine*), and the target Cloudify Manager VM, so the bootstrap process can be be performed completely offline.

## Setting Up the Bootstrap Client Machine (the CLI Machine)

### YAML Files Imported by Manager Blueprint

1. Take note of all externally-available YAML files' URL's imported by the manager blueprint, transitively (that is: YAML files which are imported directly or indirectly by the manager blueprint).
2. Look at the YAML files' URL's and aggregate them based on common URL prefixes, trying to minimize the number of groups. For example, consider the following list of URL's:

```
http://www.getcloudify.org/spec/cloudify/3.4.1/types.yaml
http://www.getcloudify.org/spec/fabric-plugin/1.4.1/plugin.yaml
http://www.some-site.org/some-path/plugin.yaml
http://www.some-site.org/some-other-path/plugin.yaml
```

   You can easily notice that the most economical way of aggregating them would be as follows:

```
http://www.getcloudify.org/*
http://www.some-site.org/*
```

   (Usually, grouping per domain name would be the best way to go)
3. Download all YAML files into the CLI machine. The files need to be laid out in a directory hierarchy, which will make it feasible to later write *Import Resolver rules* for them. This can be done in the following manner:
   a. For each group concluded in the previous step, create a separate directory on the CLI machine. Continuing the aforementioned example, you could create the following directories:
      i. `/tmp/yaml/cloudify`
      ii. `/tmp/yaml/some-site`
   b. Place YAML files into these directories, each file located relatively to the root with accordance to the original YAML URL. Example:
      i. `/tmp/yaml/cloudify/spec/cloudify/3.4.1/types.yaml`
      ii. `/tmp/yaml/cloudify/spec/fabric-plugin/1.4.1/plugin.yaml`
      iii. `/tmp/yaml/some-site/some-path/plugin.yaml`
      iv. `/tmp/yaml/some-site/some-other-path/plugin.yaml`
4. Follow the documentation for writing Import Resolver rules, in the directory from which you are going to invoke the bootstrap process. For example, if — on the CLI machine — you are going to run the bootstrap process from `/home/isaac/cfy/boot`:

```
cd /home/isaac/cfy/boot
cfy init -r
vim .cloudify/config.yaml
```

— and then add the Import Resolver rules as necessary. For the example above:

```
import_resolver:
  parameters:
    rules:
      - "http://www.getcloudify.org": "file:///tmp/yaml/cloudify"
      - "http://www.some-site.org": "file:///tmp/yaml/some-site"
```

By following the steps above, you make sure that the Cloudify DSL parser is able to read all imported YAML files by using its Import Resolver, without accessing the external network.

## DSL Resources

The `dsl_resources` manager blueprint input is a useful mechanism for uploading arbitrary files to the Cloudify Manager machine. Most typically, it is used to upload YAML files, so they become available for serving by the Cloudify Manager itself, through Import Resolver rules.

> It is important to differentiate between this step and the preceding step.
>
> The preceding step discussed how to make YAML files, required by the bootstrap process itself, available *to the bootstrap process*.
>
> This step, on the other hand, discusses how to make YAML files — which may be required by blueprints uploaded to the manager — available offline (that is, having them served by the Cloudify Manager instead of depending on these resources being available on the external network).

DSL resources, uploaded to the Cloudify Manager machine during the bootstrap process, are always — without exception — uploaded to a path which is relative to `/opt/manager/resources`.

The default value of the `dsl_resources` input handles the following YAML files:

- The Cloudify global types file (`types.yaml`)
- The `plugin.yaml` file of the following plugins:
  - Fabric
  - Diamond
  - OpenStack
  - AWS
  - vCloud
  - vSphere

You may, of course, add entries to the `dsl_resources` input (or remove entries from it). However, you must make sure that all resources denoted by `source_path` are:

- Available to the CLI machine at the time of bootstrap; and
- Provided as either file paths or URL's (**NOTE**: as long as **CFY-6505** - Using "file://" URL's in dsl_resources fails **OPEN** is not resolved, "`file://`" URL's cannot be used, unless they contain a network location, such as `file://localhost/my-file.yaml`).

### Manager-Side Import Resolver Rules

By default, the Cloudify DSL parser at the Cloudify Manager side is configured with a single Import Resolver rule:

http://www.getcloudify.org/spec  file:///opt/manager/resources/spec

This single rule, combined with the value of the `dsl_resources` input, is sufficient to serve all out-of-the-box Cloudify YAML files offline. If you have your own YAML files which are uploaded to the manager (either via the `dsl_resources` input, or manually after bootstrap), you will need to update the Import Resolver rules accordingly.

> Currently, there is no official mechanism to update the Import Resolver rules after bootstrap. Contact Cloudify Support for more information.

### Plugin Packages

The bootstrap process itself, orchestrated by the CLI, requires access to the code of each and every plugin referred-to by the manager blueprint.

Typically, the `--install-plugins` switch takes care of that. When specified, the "`source`" value of each "`plugin`" declaration included in the manager blueprint, is retrieved and then installed (using "`pip install`" semantics) on the active Python virtual environment (`virtualenv`). If the plugin's source has dependencies, these dependencies are downloaded and installed as well (again, through `pip`'s own dependency resolution mechanism).

However, in an offline environment, or an environment where access to the public network is limited, `--install-plugins` may not be applicable. To make it applicable:

- Out-of-the-box YAML files (such as `plugin.yaml` files for Cloudify's official plugins) would need to be edited so the `source` field points to an accessible location.
- If the plugin package has dependencies, those dependencies would have to exist in a `pip` mirror which is accessible to the CLI machine (and, of course, the local `pip` configuration would have to be modified to search the mirror instead of the official `pip` site).

While possible, this approach is not recommended. Doing so requires system administrators to modify out-of-the-box YAML files, as well as set up a local `pip` mirror — both pose an administration overhead.

Instead, if `--install-plugins` is not applicable, the recommended approach is to pre-install plugin packages on the CLI's `virtualenv`, and avoid specifying that switch in the command-line:

- Download each plugin package to the CLI machine (follow the `source` field)
- Download all dependencies for all plugin packages
- Install all dependencies and plugin packages on the CLI's `virtualenv`

### Cloudify Manager Resources Package

During the bootstrap process, the Cloudify Manager Resources Package (located where the `manager_resources_package` input points to) is downloaded into the manager's machine. The download is initiated from within the manager's machine; therefore, the URL specified by `manager_resources_package` must be accessible from there.

By default, `manager_resources_package` points to Cloudify's S3 bucket, which exists on the public network. If the manager's machine has no access to the public network, you'll need to download the manager resources package and host it in an accessible location — for example, on the file system of the manager.

> As long as **CFY-6498** - Unable to specify a file path for manager_resources_package **CLOSED** is not resolved, the `manager_resources_package` input cannot be provided as a `file://` URL.

## Post-Bootstrap Adjustments

### Offline Plugins

During deployment creation, Cloudify retrieves the code for each plugin used by the deployment. The retrieval algorithm works as follows:

- If the plugin definition contains `package_name` and `package_version`:
    - Conclude the operating system distribution and version, for the machine on which the operation is currently being run (the manager's machine if the operation's executor is `central_deployment_agent`, or the agent machine if the executor is `host_agent`).
    - Look up an *offline pluging package* (also called "Wagon") that satisfies the `package_name`, `package_version` and the OS distribution/version in question.
    - If a match is found, obtain the plugin's source from the Wagon and exit.
- Obtain the plugin's code from the URL denoted by the `source` key.
- Install the plugin's code using `pip`.

Therefore, in order to make the Cloudify installation self-contained, it is required to upload Wagons to the Cloudify Manager.

- A Wagon file can be thought of as a self-contained archive containing a Python module and all of its transitive dependencies.
- Wagons need only be uploaded once; they are stored using a composite key of `package_name`, `package_version` and distribution/version, and are shared between all deployments on the same Cloudify Manager installation.

Cloudify's official plugins are provided as Wagons for various distributions and versions (except for plugins that typically run on the Cloudify Manager machine, in which case, only RHEL 7.x / CentOS 7.x Wagons are provided). For custom plugins, you can use the Wagon tool to create Wagon files; these files can then be uploaded to the Cloudify Manager instance using the command-line interface or the REST API.

## Step-by-Step Guide

This section consolidates everything that has been discussed so far, into a step-by-step guide for bootstrapping offline.

This guide makes references to downloading certain files to certain locations. If the machine, on which a file is supposed to be located, doesn't have access to the public network, then the file will have to be downloaded in some way and placed in the relevant location.

### Prerequisites

- A VM containing the Cloudify CLI. This VM will be used to orchestrate the bootstrap process.
- A VM to host the manager.

### Assumptions

We will assume that the CLI VM is a CentOS 7.x.

For documentation purposes, we'll assume the following values:

| Value | Description |
|---|---|
| `<manager-private-ip>` | The private IP of the manager's VM |
| `<manager-public-ip>` | The public IP of the manager's VM. If the manager doesn't have a public IP, this value can be any IP address which the CLI machine can use in order to reach the manager |
| `<manager-ssh-key>` | Path to the SSH key with which to access the manager's VM. **The path should be available on the CLI machine**. |

### Step 1: Download the manager resources package

The manager resources package, for version 3.4.1, is located at http://repository.cloudifysource.org/org/cloudify3/3.4.1/sp-RELEASE/cloudify-manager-resources_3.4.1-sp-b410.tar.gz. Download it and place it on the manager's VM, as `/tmp/cloudify-manager-resources.tar.gz`.

### Step 2: Prepare the CLI machine

1. Create a new directory to be used as the root directory for our work (for example: `~/cloudify`).
2. Create a new directory to be used as a Cloudify working directory (for example: `~/cloudify/manager`).
3. Create a new directory to host offline resources (for example: `~/cloudify/offline`).
4. Ensure that you have access to the Cloudify manager blueprints.
    - If you had installed the CLI using the official CLI RPM, then the manager blueprints are available for you in `/opt/cloudify/cloudify-manager-blueprints`.

```
export
MANAGER_BLUEPRINTS_DIR=/opt/cloudify/cloudify-manager-blueprint
s
```

- Otherwise, obtain the manager blueprints (https://github.com/cloudify-cosmo/cloudify-manager-blueprints/archive/3.4.1.tar.gz) and extract them somewhere (for example: `~/cloudify-manager-blueprints`):

```
export MANAGER_BLUEPRINTS_DIR=~/cloudify/manager-blueprints
curl -L -o /tmp/cloudify-manager-blueprints.tar.gz
https://github.com/cloudify-cosmo/cloudify-manager-blueprints/a
rchive/3.4.1.tar.gz
mkdir -p $MANAGER_BLUEPRINTS_DIR
cd $MANAGER_BLUEPRINTS_DIR
tar -zxvf /tmp/cloudify-manager-blueprints.tar.gz
--strip-components=1
```

- (From here on, we will refer to the manager blueprints directory as `$MANAGER_BLUEPRINTS_DIR`.)

## Step 3: Prepare Python Virtual Environment

```
virtualenv ~/cloudify/env
source ~/cloudify/env/bin/activate
pip install cloudify==3.4.1
pip install wagon
```

## Step 4: Download YAML files & DSL resources

The Simple manager blueprint imports two YAML files:

- http://www.getcloudify.org/spec/cloudify/3.4.1/types.yaml
- http://www.getcloudify.org/spec/fabric-plugin/1.4.1/plugin.yaml

In addition, by default, it uploads the following DSL resources to the manager:

- http://www.getcloudify.org/spec/openstack-plugin/1.4/plugin.yaml
- http://www.getcloudify.org/spec/aws-plugin/1.4.1/plugin.yaml
- http://www.getcloudify.org/spec/tosca-vcloud-plugin/1.3.1/plugin.yaml
- http://www.getcloudify.org/spec/vsphere-plugin/2.0/plugin.yaml
- http://www.getcloudify.org/spec/diamond-plugin/1.3.3/plugin.yaml

Download all these files to the same base directory:

```
cd ~/cloudify/offline
mkdir dsl && cd dsl
declare -a yamls=("cloudify/3.4.1/types.yaml"
"fabric-plugin/1.4.1/plugin.yaml" "openstack-plugin/1.4/plugin.yaml"
"aws-plugin/1.4.1/plugin.yaml" \
  "tosca-vcloud-plugin/1.3.1/plugin.yaml"
"vsphere-plugin/2.0/plugin.yaml" "diamond-plugin/1.3.3/plugin.yaml")
for y in "${yamls[@]}"; do curl -L --create-dirs -o ${y}
http://www.getcloudify.org/spec/${y}; done
```

## Step 5: Download and install Wagon files

The Simple manager blueprint uses the Fabric plugin. We'll download it and install it into the Python virtualenv that the bootstrap is going to run from.

```
cd ~/cloudify/offline
mkdir plugins && cd plugins
curl -L -O
http://repository.cloudifysource.org/cloudify/wagons/cloudify-fabric-plu
gin/1.4.1/cloudify_fabric_plugin-1.4.1-py27-none-linux_x86_64-centos-Cor
e.wgn
wagon install
cloudify_fabric_plugin-1.4.1-py27-none-linux_x86_64-centos-Core.wgn
```

## Step 6: Prepare inputs file

```
cp $MANAGER_BLUEPRINTS_DIR/simple-manager-blueprints-inputs.yaml
~/cloudify/manager/manager-inputs.yaml
vi ~/cloudify/manager/manager-inputs.yaml
```

Populate the following inputs:

```
public_ip: <manager-public-ip>
private_ip: <manager-private-ip>
ssh_user: centos
ssh_key_filename: <manager-ssh-key>
manager_resources_package: file:///tmp/cloudify-manager-resources.tar.gz
dsl_resources:
  - {'source_path':
'/home/centos/cloudify/offline/dsl/openstack-plugin/1.4/plugin.yaml',
'destination_path': '/spec/openstack-plugin/1.4/plugin.yaml'}
  - {'source_path':
'/home/centos/cloudify/offline/dsl/aws-plugin/1.4.1/plugin.yaml',
'destination_path': '/spec/aws-plugin/1.4.1/plugin.yaml'}
  - {'source_path':
'/home/centos/cloudify/offline/dsl/tosca-vcloud-plugin/1.3.1/plugin.yaml
', 'destination_path': '/spec/tosca-vcloud-plugin/1.3.1/plugin.yaml'}
  - {'source_path':
'/home/centos/cloudify/offline/dsl/vsphere-plugin/2.0/plugin.yaml',
'destination_path': '/spec/vsphere-plugin/2.0/plugin.yaml'}
  - {'source_path':
'/home/centos/cloudify/offline/dsl/fabric-plugin/1.4.1/plugin.yaml',
'destination_path': '/spec/fabric-plugin/1.4.1/plugin.yaml'}
  - {'source_path':
'/home/centos/cloudify/offline/dsl/diamond-plugin/1.3.3/plugin.yaml',
'destination_path': '/spec/diamond-plugin/1.3.3/plugin.yaml'}
  - {'source_path':
'/home/centos/cloudify/offline/dsl/cloudify/3.4.1/types.yaml',
'destination_path': '/spec/cloudify/3.4.1/types.yaml'}
```

Also, make sure that the value of the `minimum_required_total_physical_memory_in_mb` is lower or equals to the amount of RAM the manager's VM has (in megabytes).

## Step 7: Initialize bootstrap directory

```
cd ~/cloudify/manager
cfy init -r
vi .cloudify/config.yaml
```

When editing `.cloudify/config.yaml`, add the following snippet to the end:

```
import_resolver:
  parameters:
    rules:
      - "http://www.getcloudify.org/spec":
"file:///home/centos/cloudify/offline/dsl"
```

### Step 8: Invoke bootstrap

```
cfy bootstrap -p $MANAGER_BLUEPRINTS_DIR/simple-manager-blueprint.yaml
-i manager-inputs.yaml
```

Or, to generate more logging for potential troubleshooting:

```
cfy bootstrap -p $MANAGER_BLUEPRINTS_DIR/simple-manager-blueprint.yaml
-i manager-inputs.yaml --debug | tee bootstrap.log
```

## Summary

As demonstrated, "working offline" with Cloudify requires certain additional effort to the Cloudify Manager setup process — depending on the level of separation required from the public networks (or any network, including internal ones). However, when weighed against the benefits involved in running with a self-contained Cloudify Manager, the scale clearly tips towards working offline, especially in mission-critical Cloudify Manager installations. At GigaSpaces, feedback we receive from our customers are in agreement: working offline helped reduce outages borne out of network instabilities impacting access to online resources, which is why we recommend this approach to our customers.