|  |  |  |
| --- | --- | --- |
| arrislogo_2013 | | asizearris.dot | 595328-001 Rev C |
|  | | 3 |
| This document is the property of ARRIS Group, Inc. This document may only be distributed to: (i) an ARRIS party having a legitimate business need for the information contained herein, or (ii) a non-ARRIS party having a legitimate business need for the information contained herein. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise without the prior written permission of ARRIS Group, Inc. (See Document Security Standard, 320190-000 for details.)  The ARRIS Logo and all other trademarks indicated as such herein are trademarks of ARRIS Group, Inc. All other product or service names are the property of their respective owners.  Copyright ARRIS Group Inc. 2016 All rights reserved.  Distribution of this document must adhere to the guidelines contained in the Document Control Process (365-095-0273). | | |
| Document Title | ARRIS Cloud Platform (ACP) Code Controller (CDC) User Guide | |
| Number | ENB-16843 | |
| Revision | X.2 | |
| Revision Date | 2/9/2015 | |
| Revision Author(s) | Tom Bahnck | |

ARRIS Internal

Level 2

Revision History – see Workflow History for approvers and approval dates, and Notice for release dates

| Rev | Notice # | Description | Author | Revision Date |
| --- | --- | --- | --- | --- |
| x.1 | 1 | Initial draft. | Tom Bahnck | 9/14/15 |
| x.1 | 2 | Edited some example manifest file data, corrected lexical errors in the JSON, and changed manifest.version property to 2 digits. | Tom Bahnck | 9/16/15 |
| x.1 | 3 | Embedded example release file, added cert to example release file, updated path to cert file in caFile property, embellished description of servicePort, noted that all containers specified in the manifest file are co-deployed on the same host, corrected second example container's version number and corresponding imageFile name. | Tom Bahnck | 9/16/15 |
| x.1 | 4 | Reordered manifest objects and properties to have (1) properties at the top of objects and in alphabetical order, and (2) objects to follow properties in alphabetical order; edited descriptions of some manifest data, renamed advertisedVolumes to advertisedDataVolumes and volumesRequiredFrom to dataVolumesRequiredFrom in order to distinguish from host volume mounts, added application to discovery object, renamed haProxyPort to proxyPort, added envVars to support environment variable passing to running containers, updated example release file, added note on Docker tags to semantic versioning description. | Tom Bahnck | 9/21/15 |
| x.1 | 5 | Added exposedPorts and publishedPorts to manifest file and updated example release file accordingly; changed path of reserved location for release files. | Tom Bahnck | 9/28/15 |
| x.1 | 6 | Corrected resource and scale structures, updated example release file, added Docker image considerations and explanation of container naming. | Tom Bahnck | 10/19/15 |
| x.1 | 7 | Changed 'docker export' to 'docker save' in Appendix B. | Tom Bahnck | 10/23/15 |
| x.1 | 8 | Capping this revision for distribution with the ACP 2.0.2 release. Changed title to user guide. Added notes on known issues regarding network links and scale instances. Removed cpu and storage resources properties. | Tom Bahnck | 10/26/15 |
| x.2 | 1 | Corrected typo in manifest file example and updated work flows diagram. | Tom Bahnck | 11/10/15 |
| x.2 | 2 | Added Import & Use section. Made editorial changes. Removed properties from the manifest file which are not currently supported. | Tom Bahnck | 12/23/15 |
| x.2 | 3 | Updated format of container and host fields inside publishedPorts to be strings, in order to accommodate port ranges. | Tom Bahnck | 1/15/16 |
| x.2 | 4 | Removed network links from manifest file. | Tom Bahnck | 2/9/16 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

Section Title Page Number

1. Introduction 4

1.1. Abbreviations and Acronyms 4

1.2. Applicable Documents 4

2. Overview 5

2.1. Workflows 5

3. Terminology 6

4. Release Package Requirements 7

4.1. Release Package Format 7

4.2. Manifest File Contents 8

4.3. License Metadata File 14

4.4. Example Release File 15

5. Appendix A: Semantic Versioning 15

6. Appendix B: Docker Image Considerations 15

List of Figures

Section Title Page Number

Figure 1 - Code Controller Workflows 5

Figure 2 - Product Topology 6

Figure 3 - Code Lifecycle 6

Figure 4 - Release Package Structure 7

Figure 5 - Manifest File Contents 8

Figure 6 - Attribution File Contents 14

List of Tables

Section Title Page Number

**No table of figures entries found.**

# Introduction

This document describes the way to deploy code in ARRIS Cloud Platform (ACP). Application deployment functions are provided by ACP's Code Controller (CDC) service, otherwise known as CDCaaS. It is a release artifact distributed with the rest of ACP 2.0, and installed via the process in the ACP installation guide. Refer to the ACP Core release for a copy of the installation guide, or contact the ACP team to obtain a copy through Stash or email.

## Abbreviations and Acronyms

| Acronym | Description |
| --- | --- |
| ACP | ARRIS Cloud Platform |
| CA | Certificate Authority |
| CDC | Code Controller |
| CI/CD | Continuous Integration / Continuous Deployment |
| CLI | Command Line Interface |
| OSS | Open Source Software |
| REST | REpresentational State Transfer |

## Applicable Documents

365-095-29362 ARRIS Cloud Platform (ACP) Code Controller REST API Specification

# Overview

CDC does the following:

* Automatically import a standard release file (format described below);
* Aggregate OSS license attributions from all software running in the ACP environment;
* Install and uninstall releases via simple REST commands;
* Deploy and undeploy applications using simple REST commands;
* Allocate containers to VMs in the ACP environment in accordance with applications' compute resource requirements;
* Manage the firewall on the VMs hosting application containers such that all required ports are accessible;
* Automatically integrate with ACP's Service Registration and Discovery (SR&D) platform, such that applications deployed using CDC need only advertise/declare their services of interest and allow ACP to instantiate, link, and configure supporting containers which drive the SR&D platform.  
    
  *Note: The SR&D platform is essentially a distributed load balancer for HTTP traffic. ACP also provides a separate, central load balancer service based on NGINX which applications may use as well. It is neither related nor dependent on the SR&D platform.*
* Automatically pass miscellaneous data into applications' containers using environment variables.

As CDC evolves, other core functions will be added as well, such as zero-downtime upgrades, resource affinity and anti-affinity, and application-level dependencies resolution.

## Workflows

CDC was required to support the different workflows illustrated below, in Figure 1. Each workflow is supported by use of a REST API as the control interface to CDC. Via CLI, an administrator can use cURL. Via AJAX or similar remoting, a user interface can populate its screens with CDC data. A customer, if desired, can also directly integrate exiting back office systems with CDC. Development teams may also directly integrate CDC with build servers, to enable DevOps processes.

This REST API is specified in [365-095-29362]. It is downloadable from DigitalCM and also included in the ACP Code Controller release. It may be shipped outside of ARRIS as well.

Figure 1 - Workflows



# Terminology

It is necessary to understand the following terminology, in CDC vernacular, in order to follow CDC theory of operations.

* A **SOLUTION** is an end-to-end system which delivers value to a customer. It may be comprised of appliance-based and third-party products, in addition to ACP cloud applications. CDC does not operate at the solution level.
* An **APPLICATION** ("app") is the unit of release for a product team. It is a single file, i.e. release package, which includes a manifest as well as container images used together to serve a broad function in the overall solution (as described above). This is the top-level scope of CDC.
* A **CONTAINER** is a unit of deployment, rather than release, on ACP. It performs a narrower function for the overall application, and may started and run independently from all other containers which comprise the application. A container generally exposes one or more interfaces, referred to as services, on one or more TCP port(s).
* A **SERVICE** is a fundamental unit of work. It is fulfilled by a process running in a container and generally listening on a TCP port.

Figure 2 summarizes the topology of entities described above.

Figure 2 - Product Topology



The terms that follow explain the functions or processes that CDC fulfills, rather than the artifacts upon which it operates:

* To **INSTALL** a release refers to the ingest and preparation of an application's release file for deployment in an ACP environment. Files are extracted, manifest file is validated, container images are loaded to ACP Docker registry, etc.
* To **CUSTOMIZE** a release refers to following application-specific instructions to tailor the release artifacts (e.g. config files) to a specific environment. Specific ports are exposed, certificates are generated, environment variables are defined, etc.
* To **DEPLOY** an app refers to the execution of an app in the ACP environment. Compute resource and scale levels are selected, container images are downloaded to host VMs, firewall is modified, containers are started, ACP services (e.g. database, load balancer) are configured, etc.
* To **UNDEPLOY** an app refers to the intentional and normal stopping of an application and freeing of the compute resources it was consuming in the ACP environment. Containers are stopped, container instances are deleted, container images are removed from host VMs, remaining deployment artifacts are deleted, etc.
* To **UNINSTALL** a release refers to the removal of all release artifacts from the ACP environment and Code Controller. Extracted files are deleted. Currently container images are not removed from the Docker registry.

Figure 3 summarizes the standard procedure, i.e. application lifecycle, formed by the functions explained above.

Figure 3 - Standard Procedure



# Release Package Requirements

In order for CDC to operate consistently/reliably/successfully, release files must contain certain artifacts. There must be a JSON-based manifest file that describes the application, as well as the relationships among its containers its dependencies on the outside world. The application may optionally embed its containers as well.

## Release File Format

The release file must be a .zip file. It should be the same file published to DigitalCM for distribution to customers.

At the root of the zip file must be:

* A text file named *manifest.json* which contains the configuration parameters described below (in section 4.2). Some parameters in the manifest file point to container images, which may be included in the zip file itself, or located on a Docker registry within network reach of CDC. If a container image is located on another registry which uses HTTPS, a CA certificate to trust the remote registry must also be included in the zip file.
* A directory named *attribution* which contains a subdirectory named *licenses*, and a text file named *<app id>.json*, where *<app id>* is a canonical ID string for the application. *<app id>.json* must contain the license metadata described below (in section 4.3). The *licenses* directory should contain one or more text files, each with the full text of a given OSS license used by the application.

Figure 4 illustrates the release file contents.

Figure 4 - Release File Structure



## Import & Use

To import a release file into the ACP environment, first identify the VM on which CDC is running. The CDC VM is selected as part of the ACP installation process, as documented in the install guide provided in the ACP Core release file. If the DNS server setup instructions in the guide were followed properly, the CDC VM can be identified by resolving the host name "cdcaas".

nslookup cdcaas

Server: 10.0.0.7

Address: 10.0.0.7#53

Name: cdcaas.internal.acp.arris.com

Address: 10.0.0.122

Upload the application release file to the releases directory on the CDC VM, at */var/opt/code\_controller/releases*.

Then, via the CDC REST API (from [365-095-29362]), use an *Install-Release-File* command to install the release, by passing the release file name as a parameter.

POST /cdcaas/v1.0/install?file=app-a-1.3.4.zip

Example response:

{

"$schema": "/schemas/App/v1.0",

"App": {

"id": "app-a",

"version": "1.3.4",

"scale": [

"test",

"production"

],

"resources": [

"thin",

"preferred"

]

}

}

CDC responds immediately with the application ID, version, and a list of resource and scale options with which the application can be deployed. However, installation is not complete until CDC reports *status* of *COMPLETE\_INSTALL* with *result* of *OK* upon subsequent *Retrieve-Installation-Status* command:

GET /cdcaas/v1.0/install/app-a/1.3.4

Example response:

{

"$schema": "/schemas/App/v1.0",

"App": {

"Task": {

"type": "INSTALL",

"started": "2015-08-18T12:00:00Z",

"status": "COMPLETE\_INSTALL",

"result": "OK",

"lastChange": "2015-08-18T12:03:00Z"

}

}

}

Once *status* is *COMPLETE\_INSTALL* and *result* is *OK*, select a *resource* and *scale* string from the response above, and use them as parameters in a subsequent *Deploy-App* command in order to deploy the application:

PUT /cdcaas/v1.0/deploy/app-a/1.3.4

CDC returns *201/Created* status, indicating that deployment is underway. To check on deployment progress, use the *Retrieve-Deployment-Status* command:

GET /cdcaas/v1.0/deploy/app-a/1.3.4

Example response:

{

"$schema": "/schemas/App/v1.0",

"App": {

"Task": {

"type": "DEPLOY",

"started": "2015-08-18T12:00:00Z",

"status": "COMPLETE\_DEPLOY",

"result": "OK",

"lastChange": "2015-08-18T12:03:00Z"

}

}

}

As with installation, deployment is complete when the *status* returned is *COMPLETE\_DEPLOY* with *result* of *OK*. From there, application-specific steps can be taken to verify the application's operations.

The *Undeploy-App* and *Uninstall-Release* commands can be subsequently used to later undeploy and uninstall the application, respectively.

DELETE /cdcaas/v1.0/deploy/app-a/1.3.4

DELETE /cdcaas/v1.0/install/app-a/1.3.4

*Note: Undeployment must precede uninstallation.*

'Each command returns 200/OK. Like initial installation and deployment, the progress of the undeploy and uninstall operations can be monitored using *Retrieve-Deployment-Status* and *Retrieve-Installation-Status* commands, respectively.

GET /cdcaas/v1.0/deploy/app-a/1.3.4

GET /cdcaas/v1.0/install/app-a/1.3.4

Operations are complete when the *status* returned is *COMPLETE\_UNDEPLOY* or *COMPLETE\_UNINSTALL*, respectively, and *result* is *OK*.

Example responses:

{

"$schema": "/schemas/App/v1.0",

"App": {

"Task": {

"type": "UNDEPLOY",

"started": "2015-08-19T12:00:00Z",

"status": "COMPLETE\_UNDEPLOY",

"result": "OK",

"lastChange": "2015-08-19T12:03:00Z"

}

}

}

{

"$schema": "/schemas/App/v1.0",

"App": {

"Task": {

"type": "UNINSTALL",

"started": "2015-08-19T12:00:00Z",

"status": "COMPLETE\_UNINSTALL",

"result": "OK",

"lastChange": "2015-08-19T12:03:00Z"

}

}

}

## Manifest File Contents

The manifest file is a JSON-formatted text file located at the root of the release file. Its contents are described in the example below, in Figure 5. **Required data items are colored red**. Optional data items may be omitted entirely from the manifest file. If an optional data item is omitted, ACP may infer a default value for it, as described below.

Note: For help constructing a valid manifest file, use the Manifest File Builder included in ACP's DevOps repo, and hosted on Areopagus (http://areopagus/manifest).

Figure 5 - Manifest File

**{**

**"manifest": {**

**"version": "1.0",**

*-- This is the Manifest file protocol version. It is presently 1.0.*

**"app": {**

*-- This is an object that represents the application.*

**"displayName": "Application A**",

*-- This should be a human-readable display name for the application, suitable for use by a UI.*

**"id": "app-a",**

*-- This is a canonical ID string for the application. ACP uses this in a variety of ways to keep resources for each application in the ACP environment separate. It is user-defined (i.e. the development team should choose an app ID at build time, which has a large probability of uniqueness across the company. Going forward, the ACP team may provide an app ID registry where product teams can disclose the name(s) they have chosen and also discover if a desired app ID is already reserved.)*

**"version": "1.3.4",**

*-- This is the version number of this application release. It order to take advantage of ACP's compatibility logic during deployment, upgrade, and downgrade in the future, the version number string should use semantic versioning per Appendix A (section 5).*

**"containers": [**

*-- This is a list of container images in the release file.*

**{**

*-- This is the first container image in the release file.*

**"imageName": "arrs/app-a-component-x",**

*-- This is a name which ACP will give the container image when it is loaded to the private ACP Docker registry. If ACP is pulling the image from a remote registry via the "remoteRegistry" data items below, it must also match the name of the image stored in the remote registry.*

**"version": "1.3.4.1",**

*-- This is the tag to assign to this image when it is loaded to the private ACP Docker registry. Like "imageName", if is pulling the image from a remote registry via the "remoteRegistry" data items below, it must also match the tag associated with the image stored in the remote registry. Docker image tags should use 4-digit semantic versioning per the description in Appendix A (section 5).*

**"advertisedDataVolumes": [**

*-- This is an array of data volumes which this container makes available to other containers in the application which request them via the "dataVolumesRequiredFrom" object below.*

**{**

**"permissions": "ro",**

*-- This is a permissions level to enforce on the files located at "volumePath" below. "ro" means read-only.*

**"volumePath": "/public/outgoing"**

*-- This is a path in this container to expose to other containers which request access to it via the "dataVolumesRequiredFrom" object below.*

**},**

**{**

**"permissions": "rw",**

*-- This is also a permissions level to enforce on the files located at "volumePath" below. "rw" means read-write.*

**"volumePath": "/public/incoming"**

**}**

**],**

**"dataVolumesRequiredFrom": [**

*-- This is an array of references to other container images, also defined in the "containers" array in this manifest file, which require access to the volumes enumerated in this container's "advertisedDataVolumes" paths.*

**{**

**"imageRef": "arrs/app-a-component-y"**

*-- This is the name of another container in this manifest file whose "advertisedDataVolumes" this container needs to access. It must match the "imageName" property of the container having volumes of interest.*

**}**

**],**

**"discovery": [**

*-- This array, as well as the "registration" array, configure ACP's Service Registration & Discovery (SR&D) service, which load-balances traffic among multiple instances of a container without use of a central load balancer.*

**{**

**"application": "app-b",**

*-- This is the name of an application providing a service which this container must access. Its value must match the manifest.app.id property in the manifest file of the target application when the target application was deployed.*

**"proxyPort": 10000,**

*-- This is the port number through which the application in this manifest file will access the target service. It need not be the same port number that the target application registered the service with in the "registration" object in its own manifest file. The target service will be accessed by the application in \*this\* manifest file using the host name "arrs-synapse:<proxyPort>". Note that there is no need to list this port in either the exposedPorts or publishedPorts sections below.*

**"serviceName": "svc2"**

*-- This is the name which the target application provided for the target service in the "serviceName" property of the "registration" object in the target application's manifest file. It is effectively a handle for the service which binds the "discovery" and "registration" data structures.*

**}**

**],**

**"envVars": [**

*-- This is an array of static environment variables to pass into this container at runtime.*

**{**

**"name": "VAR\_1",**

*-- This is an environment variable's name.*

**"value": "value1"**

*-- This is an environment variable's value.*

**}**

**],**

**"exposedPorts": [**

*-- This is a list of port numbers within this container which will be exposed to the network. In order to open a port to the outside world (i.e. outside of the container), it must be published via "publishedPorts" below. There is no need to expose ports which are also published; enumerating ports in the exposed ports section is primarily for documentation purposes.*

**"9000",**

*-- This is a single exposed port.*

**"9005-9010"**

*-- This is a range of exposed ports.*

**],**

**"imageSource": {**

*-- This is the location of this container image. It is described by either a "remoteRegistry" data set below, or an "imageFile" data set as shown in the next container definition. Either "remoteRegistry" or "imageFile" must be present in this "imageSource" object.*

**"remoteRegistry": {**

*-- This data set is required when this container image is stored in a remote Docker registry.*

**"caFile": "/certs/appteamCA.crt",**

*-- This is the path to a certificate file in the release file which establishes trust between CDC and the remote server. The certificate file must be PEM-encoded and unencrypted. This data item is only required when "secureRegistry", as described below, is set to true.*

**"location": "appteam-repo.arrs.arrisi.com:5000",**

*-- This is the host name or IP address, and port number, of the remote registry server from which this container image can be downloaded.*

**"secureRegistry": true**

*-- This is a flag which indicates whether or not the remote registry server requires HTTPS. If so, "caFile", as described above, must contain the path to a certificate which CDC can use to establish trust with the remote server.*

**}**

**},**

**"publishedPorts": {**

*-- This enumerates all ports within this container which should be published to the host (i.e. the outside world). Each port defined in this section may optionally be enumerated in the "exposedPorts" section as well.*

**"list": [**

*-- This is an array of mappings from container-side port numbers to host-side port numbers. Not all exposed ports in the "exposedPorts" data set need to have mappings in the "publishedPorts" data set.*

**{**

**"container": "9000",**

*-- This is a single container-side port number.*

**"host": "9000"**

*-- This is a single host-side port number to map to the above container-side port number in the "container" property. This field is optional. If there is a "container" property with no corresponding "host" property, the container port number is mapped to a random port on the host within the ephemeral port range (e.g. 32768 - 61000).*

**},**

**{**

**"container": "9005-9010",**

*-- This is a range of container-side port numbers to publish to the host.*

**"host": "9005-9010"**

*-- This is a corresponding range of host-side port numbers to map to the above container-side port number range in the "container" property. Like the previous example, this field is optional. If there is a "container" property which contains a port range, with no corresponding "host" property, the container port numbers in the range are mapped to random port numbers on the host within the ephemeral port range.*

**}**

**]**

**},**

**"registration": [**

*-- This is an array of services to make available via ACP's Service Registration & Discovery (SR&D) platform. SR&D is a way of distributing traffic to multiple instances of a container without using a central load balancer.*

**{**

**"healthCheckUrl": "/health",**

*-- This is a URI which ACP can use to check health of this service (i.e. the process running on "servicePort" below in this container). A 2XX response status is considered healthy, while 4XX and 5XX as well as timeouts and non-standard responses are considered not healthy. Unhealthy instances are removed from the backend server pool until they report back healthy status again.*

**"serviceName": "svc1",**

*-- This is a name for this service which other containers, either within or outside of this application, can use to request access to this service. It should be unique within the scope of this application only.*

**"servicePort": 9050**

*-- This is the port number in this container on which this service is accessed. It is the container-side port number and may not necessarily be the same as the port number*

*published to the host. Note that there is no need to list this port in either the exposedPorts or publishedPorts sections above.*

**}**

**],**

**"resources": [**

*-- This is an array of different resource level sets that can be assigned to this container at deployment time. Different level sets can be defined for different target deployment environments, e.g. 250 MB of memory for test versus 1 GB memory for production.*

**{**

**"level": "min",**

*-- This is a label for this resource level data set. This string value assigned to "level" is passed to CDC as an HTTP parameter at deployment time to allocate this level set to this container (see [*365-095-29362*]). There can be any number of levels defined in the "resources" data set, to fit any deployment scenario(s) the application product team sees fit.*

**"memory": 512,**

*-- This is the number of MB of memory to allocate to this container at deployment time. Fractional (i.e. decimal) values are not permitted.*

**"vcpus": 1**

*-- This is the number of CPUs (i.e. virtual processors) to allocate to this container at deployment time. Fractional (i.e. decimal) values are not permitted.*

**}**

**],**

**"scale": [**

*-- This is an array of different scale levels that can be assigned to this container at deployment time. Scale refers to the number of instances of this container to start at deployment time. It is expressed as a whole number. Different scale levels can be defined for different workloads, e.g. 1 instance for development versus 10 for production.*

**{**

**"level": "small"**,

*-- This is a label for this scale level. The string value assigned to "level" is passed to CDC as an HTTP parameter at deployment time, to allocate this scale level to this container (see [*365-095-29362*]). It need not match any resource level (i.e. resource level and scale level are independently selectable at deployment time). There can be any number of levels defined in the "scale" data set, to fit any deployment scenario(s) the application product team sees fit.*

**"instances": 1**

*-- This is the number of instances of this container to start when a scale level of "small" (per example above) is selected at deployment time.*

*Note: In ACP release 2.0.2, an integration issue with Docker Swarm causes the number of instances to always be 1 regardless of this value.*

**}**

**]**

**},**

**{**

*-- This is an example of a second container within the application. It is shown here to simply draw contrast with the data items used in the first container.*

**"imageName": "arrs/app-a-component-y",**

**"version": "1.3.4.5",**

**"imageSource": {**

**"imageFile": "/images/arrs-app-a-component-y\_1.3.4.5.tgz"**

*-- Note use of "imageFile" rather than "remoteRegistry" in the "imageSource" data set, to describe a container image provided within the release file. The "imageFile" path points to a Docker image provided in the same release file as this manifest file.*

**},**

**"advertisedDataVolumes": [**

*-- Note that the data volumes in this data set are made accessible to the previously defined container because the previous container cited this container's image name in an "imageRef" property in its "dataVolumesRequiredFrom" data set.*

**{**

**"permissions": "ro",**

**"volumePath": "/public/outgoing"**

**},**

**{**

**"permissions": "rw",**

**"volumePath": "/public/incoming"**

**}**

**],**

**"dataVolumesRequiredFrom": [**

*-- Note that the volume paths in the previously defined container's "advertisedDataVolumes" data set are made available to this container because the previous container's image name is cited in an "imageRef" property below.*

**{**

**"imageRef": "arrs/app-a-component-x"**

**}**

**],**

**"discovery": [**

*-- Note that this "discovery" object allows this container to access the service named "svc1", defined in the previous container's "registration" section, via port number 6000 (i.e. host name arrs-synapse and port number 6000). The ACP SR&D platform will automatically load balance all traffic to "svc1" among all running instances of the previous container.*

**{**

**"application": "app-a",**

**"proxyPort": 6000,**

**"serviceName": "svc1"**

**}**

**],**

**"resources": [**

*-- Note that the property values of this resource set differ from the property values of the same resource set in the previously defined container; however, the same resource level must be defined in every container in this manifest file (e.g., in this manifest file example, both containers must have a resource object with a "level" of "min".)*

**{**

**"level": "min",**

**"memory": 256,**

**"vcpus": 2,**

**}**

**],**

**"scale": [**

*-- Note that the number of instances for a given scale level in this container can differ from the number of instances for the same scale level in the previously defined container; however, the same scale levels must be defined for every container in this manifest file (e.g., in this manifest file example, both containers must have a scale object with a "level" of "small".)*

**{**

**"level": "small",**

**"instances": 2**

**}**

**]**

**}**

**]**

**}**

**}**

**}**

## License Metadata File

There must be a license metadata file named *<app id>.json* in a directory named *attribution* at the root of the release file. Its contents are described in the example that follows.

Figure 6 - Attribution File Contents

**{**

**"ossAttribution": {**

**"product": "app-a",**

*-- This is the application ID string. It should match the manifest.app.id property value in the release manifest file, as described above.*

**"productVersion": "1.3.4",**

*-- This is the version number of this application release. It should match the manifest.app.version property value in the release manifest file, as described above.*

**"entries": [**

*-- This array enumerates all of the OSS components in the release which require license attribution.*

**{**

**"component": "etcd",**

*-- This is the name of an OSS component.*

**"componentVersion": "2.0",**

*-- This is the version of the OSS component.*

**"componentUrl": "https://coreos.com/",**

*-- This is a URL to an entity on the internet which describes or distributes the OSS component.*

**"license": "Apache 2.0",**

*-- This is the name of the license under which the OSS component is distributed.*

**"licenseUrl": "licenses/apache-license-2.0.txt",**

*-- This is the path to the full text of the license for the OSS component. It is relative to the "attribution" directory in the release file.*

**"notice": ""**

*-- This is the text of the informational NOTICE required by the OSS license entity, if necessary.*

**}**

**]**

**}**

**}**

## Example Release File

Embedded below is an example release file which matches the examples used in this document. Note that the container image file(s) do not contain actual images. The intent is solely to demonstrate the release file structure and how it relates to the manifest file and license metadata file.



# Appendix A: Semantic Versioning

ACP uses semantic version numbers in order to determine whether or not different versions of applications are compatible with one another. The *manifest.app.version* value in the release manifest file is used for this purpose. Use cases for semantic versioning include:

* Determining whether an installed version of an application's dependency can support an application's deployment;
* Determining whether an application upgrade or downgrade will or will not impede the operation of other applications running in the same ACP environment.

Applications should follow the practices described in <http://semver.org>, noting the following:

* Three-digit version numbers should be used for application-level versioning. This is the construct upon which ACP applies compatibility logic.
* Four-digit version numbers should be used for Docker containers. The fourth digit (e.g. 1.0.0.#) should represent build number. This allows a development or test environment to hold multiple builds of the same container image in the same private registry. It also optimizes image transfer from the Docker registry, and allows a system administrator to correlate a running container to an application release when inspecting the ACP environment.  
    
  *Note: Application teams should not use container version values such as "latest" or "current". These values are not descriptive enough to support deployment environments. CDC performs software release artifact installation where if version values are re-used from release to release, the software images maintained in ACP's registry will be overwritten, and can cause subsequent confusion in resolving application defects in deployment environments, as a controlled software artifact maintained in ACP is now newer than a deployed software artifact.*

Applications which do not use semantic versioning are permitted, but ACP assumes that they are not compatible with any other application.

# Appendix B: Docker Image Considerations

The following is a list of best practices which guarantee interoperability with CDC and foster predictable behavior within the ACP environment.

* **One container image per TAR File.** CDC presently does not support multiple images in a single TAR file, as cited in an *imageFile* property in the manifest file below. An example of a Docker export command which places a single image in a TAR file is as follows:  
  docker save myapp:1.3.4 > myapp-1.3.4.tar
* **One name and one version tag per image.** Multiple names and tags for a given image are not preserved when the image is extracted from the TAR file and pushed to ACP's Docker registry. Only the name and version tag stated in the release manifest file is preserved.
* **The name and version tag in the manifest file is canonical.** CDC uses the *imageName* and *version* properties in the manifest file to name and tag a container image when it is loaded to the private Docker registry in the ACP environment. If the name or tag embedded in the image differs from the name or image in the manifest file, the information in the manifest file is preserved.
* **Use the *exec* form of *ENTRYPOINT*.** This is the most flexible and predictable way to begin processing in a container. An example of the *exec* form of ENTRYPOINT in a Dockerfile is as follows:  
  ENTRYPOINT ["/container\_start.sh", “arg1”, “arg2”]
* **Handle SIGTERM in the entrypoint.** In the future, ACP may adjust the scale of an application by triggering additional containers to start, or removing running containers from a deployment environment. When running containers are scheduled for termination, the Docker engine signals the containers using SIGTERM. An application can handle this by trapping SIGTERM in its defined ENTRYPOINT, and subsequently performing any necessary clean-up functions before the program is exited. "Normal" entrypoint termination as a result of handling SIGTERM should return a zero value. "Abnormal" entrypoint termination as a result of handling SIGTERM should return a non-zero value. Containers which do not handle SIGTERM are force-terminated after a fixed timer expires. As a consequence, the remaining container instances could encounter incorrect, incomplete, or corrupt state data. If the entrypoint for a container is a shell script, e.g. to spawn multiple other executables, care should be taken to remember all spawned PID values. Such spawned processes and their associated PID values can also receive the SIGTERM signal for graceful shut-down; however, this requires support in the entrypoint script to forward SIGTERM to the spawned programs.

Note also that when Docker containers are started, ACP names them using the convention *{app-id}-{#}*, where *app-id* is the value of the *manifest.app.id* property in the application's manifest file, and *#* is the ordinal position of the container's definition in the release manifest file.