Active System Manager Release 8.2
SDK Reference Guide

Notes, cautions, and warnings

NOTE: A NOTE indicates important information that helps you make better use of your computer.

CAUTION: A CAUTION indicates either potential damage to hardware or loss of data and tells you how to avoid the problem.

WARNING: A WARNING indicates a potential for property damage, personal injury, or death.

2016 - 04

Rev. A00

Contents

1

Overview

ASM is intended to automate the processes involved in provisioning, configuring, and managing bare metal hardware: chassis, servers, switches, and networked storage devices. ASM helps in automating processes related to hardware discovery, operating system installation, virtual network creation, and firmware management.

At the highest level of the deployment stack, ASM has the ability to install certain applications on a provisioned machine as part of the deployment process. Only a handful of native applications are available in ASM and thus the application SDK provides a mechanism to extend the ability of ASM to deploy additional applications.

ASM utilizes Puppet to install applications. Thus for any application, one needs to create or utilize an existing Puppet module or modules. The Application SDK is then a way to make ASM act as a Puppet *external node classifier* and to map Puppet class and resource parameters to the ASM UI.

NOTE: This document is intended for the technical audience. It is assumed the reader is familiar with Puppet, and concepts within Puppet like modules and resources.

Other Documents You May Need

See **http://www.dell.com/asmdocs** for additional supporting documents such as:

- *Active System Manager Release 8.2 User's Guide*
- *Active System Manager Release 8.2 Release Notes*
- *Active System Manager Release 8.2 Installation Guide*
- *Active System Manager Release 8.2 API Reference Guide*
- *Active System Manager Release 8.2 Compatibility Matrix Guide*
- *Active System Manager Integration for VMware vRealize Orchestrator User's Guide*

You can also see **http://www.dell.com/asmtechcenter** for how-to videos, white papers, blogs, and support forums.

Uploading an Application Module

Application modules are uploaded to ASM from the **Settings** → **Add-On Modules** page.

Supported formats for module uploads are .zip files.

NOTE: If you attempt to upload a standard Puppet module, you receive an error message that the required file asm_input.json is missing.

3 Content of an Application Module

An **Application Module** is typically a Puppet module plus one additional metadata file called asm_input.json. Technically, the minimal Application module could include only an asm_input.json file with nothing else, acting simply as a manifest or partial manifest. In that case, any required classes should already have been included in another application module or modules. Here is an example layout of the application module directory for the more typical case.

```
my_module /
   |-- asm_input.json
   |-- files
   |-- Gemfile
   |-- manifests
   |    `-- init.pp
   |-- metadata.json
   |-- Rakefile
   |-- README.md
   `-- tests
```

Key files of interest are:

asm_input.json is a JSON file containing metadata describing the puppet classes and types that should be exposed in the ASM UI.

metadata.json is a JSON file containing metadata about the puppet module itself. It contains data such as `name`, `version`, `author`, and the version of Puppet the module is expected to function with, operating systems it supports, and ruby gems that it requires. A full description of this file can be found in the Puppet documentation.

NOTE: In fact, ASM does not require a `metadata.json` file at all, but if one is present and contains values for these fields, they can be omitted in the `asm_input.json` and the values from the `metadata.json` file is used.

Structure of ASM_INPUT.JSON

A basic example of asm_input.json with the fields, classes, and types. A detailed explanation of this follows in the next subsection.

```
{
      "name": "my_module",
      "version": "0.1.0",
      "operatingSystemSupport": [
        {
          "operatingSystem": "RedHat",
          "releaseVersions": [ "5", "6", "7" ]
        },
      ],
      "requirements": [
        {
          "name": "pe",
          "versionRequirement": ">= 3.0.0 & 2015.3.0"
        },
        {
          "name": "puppet",
          "versionRequirement": ">= 3.0.0 & 5.0.0"
        }
      ],
      "classes": [ ... ],
      "types": [ ... ]
    }
```

This is almost a direct copy of a Puppet metadata.json file, except for renaming of some keys. A simple script to generate this structure from a Puppet module can be downloaded from the Dell techcenter at: http://en.community.dell.com/techcenter/converged-infrastructure/w/wiki/4318.dell-active-system-manager.

The mapping from Puppet manifest keys to ASM module keys is:

```
'operatingsystem_support' => 'operatingSystemSupport'
'operatingsystem'         => 'operatingSystem'
'operatingsystemrelease'  => 'releaseVersions'
'version_requirement'     => 'versionRequirement'
'requirements'            => 'requirements'
'version_range'           => 'versionRequirement'
```

The `name`, `version`, `operatingSystemSupport` and `requirements` fields are semantically identical to the fields of the analogous names contained in the `metadata.json` file.

An explanation of each field is as follows.

- **name**: The name of the module. By convention puppet module names are generally of the form <maintainer>-<module>. For example, the name of the Puppet Labs PostgreSQL module is 'puppetlabs-postgresql'. If present, the <maintainer> portion is removed to determine the module directory name that the module is copied into. For example, the puppetlabs-postgresql module would be copied into a directory called postgresql in the ASM appliance modules directory.
- **version**: The version of the module. Currently ASM only allows one version of a module to be uploaded at a time.
- **operatingSystemSupport**: A list of hashes containing `operatingSystem` and `releaseVersions`. This describes the operating systems and versions that are supported by the module. See the Puppet documentation for more detail.
- **requirements**: A list of hashes containing name and versionrequirements of the puppet the module is to function with. The `name` "pe" refers to supported Puppet Enterprise versions and the name "puppet" refers to open-source puppet. If these are present ASM validates that the module is supported by the version of Puppet that ASM ships with. That is Puppet Enterprise 3.3 which is roughly equivalent to open-source Puppet version 3.6.

NOTE: Because of the semantic version of library used, the puppet intersection expression, >=3.4.0<5.0.0, should be expressed differently as, >=3.4.0 & 5.0.0.

- **classes**: A list of Puppet classes that the Application Module wants to instantiate and expose through the ASM UI. The format is discussed in more detail following. Classes in Puppet are singletons, so only one instance of any class component can be attached to a given ASM server or VM component.
- **types**: A list of Puppet resources that the Application Module wants to instantiate and expose through the ASM UI. The data format for `types` is identical to the `classes`. Unlike classes, types are not singletons and more than one "type" component could be attached to the same ASM server or VM component.

The contents of the classes and types arrays are explained next.

5 Class and Type Structure

Here is an example of a class or type element inside the classes or tyes arrays:

```
{
      "id": "component-linux_postinstall-1",
      "componentValid": {
        "valid": true
      },
      "name": "linux_postinstall",
      "type": "SERVICE",
      "teardown": false,
      "resources": [
        {
          "id": "linux_postinstall",
          "displayName": "Application Settings",
          "parameters": [
            {
              "id": "install_packages",
              "type": "STRING",
              "displayName": "Install Packages",
              "required": true,
              "requiredAtDeployment": false,
              "hideFromTemplate": false,
              "readOnly": false,
              "generated": false,
              "infoIcon": false,
              "maxLength": 256,
            },
         …
          ]
        }
      ]
    }
```

This represents a single class or type element of the classes or types array. It exposes just a single attribute through ASM (for readability). But additional parameters would be added the same way.

The declaration above, if used in `classes`, when filled out with user values would be equivalent to a puppet manifest file such as:

```
class { 'linux_postinstall':
      install_packages     => '',
      upload_share         => '',
      upload_file          => 'test.sh',
      upload_recursive     => 'false',
      execute_file_command => 'bash test.sh',
      yum_proxy            => ''
    }
```

While the declaration above, if used in `types`, when filled out with user values would be equivalent to a puppet manifest file such as:

```
linux_postinstall { 'unique_title':
      install_packages     => '',
      upload_share         => '',
      upload_file          => 'test.sh',
      upload_recursive     => 'false',
      execute_file_command => 'bash test.sh',
  yum_proxy              => ''
}
```

Module Dependencies

Access to the ASM appliance could allow one to load Puppet modules on the Puppet master. But barring login access, any Puppet module dependencies must be uploaded to ASM in the same way as the application module that depends on them. That is to say, for each dependency module we must add an asm_input.json file to the Puppet module, zip it, and then upload it to ASM in order to satisfy the dependency.

For example, puppetlabs-apache depends on the module puppetlabs-concat. So, to utilize puppetlabs-apache, we would have to first upload the puppetlabs-concat module.

Typically the dependency modules will not be used explicitly, so they can be simplified by leaving the classes and types arrays *mostly* empty. However, at least one class or resource must be declared in one of the classes or types arrays. A convenient way to satisfy this requirement is to declare a *notify* resource in the types array.

In fact, the generator script will do just this, by supplying the –*-dependency* argument to the generator script:

```
generate_asm_module.rb --dependency
```

A

Appendix A Example code

```
{
  "requirements": [
    {
      "name": "pe",
      "versionRequirement": ">= 3.0.0 & < 2015.4.0"
    },
    {
      "name": "puppet",
      "versionRequirement": ">= 3.0.0 & < 5.0.0"
    }
  ],
  "operatingSystemSupport": [
    {
      "operatingSystem": "RedHat",
      "releaseVersions": [
        "6",
        "7"
      ]
    },
    {
      "operatingSystem": "Debian",
      "releaseVersions": [
        "7.8",
        "8"
      ]
    },
    {
      "operatingSystem": "Ubuntu",
      "releaseVersions": [
        "12.04",
        "14.04"
      ]
    }
  ],
  "classes": [
    {
      "id": "asm_test",
      "componentValid": {
        "valid": true
      },
      "name": "asm_test",
      "type": "SERVICE",
      "teardown": false,
      "resources": [
        {
          "id": "asm_test",
          "displayName": "asm_test",
          "parameters": []
        }
      ]
    }
  ],"types": [
    {
      "id": "component-say_something",
      "componentValid": {
        "valid": true
      },
      "name": "say_something",
      "type": "SERVICE",
      "teardown": false,
      "resources": [
        {
          "id": "say_something",
          "displayName": "say_something",
          "parameters": [
            {
              "id": "title",
              "type": "STRING",
              "displayName": "Content",
              "required": true
            },
            {
              "id": "ensure",
              "type": "STRING",
              "value": "present",
              "displayName": "Ensure",
              "required": true
            }
          ]
        }
      ]
    },
  ]
}
```