http://wso2.com/

Stratos 2.0 Cartridge Development Guide

Date: 05 April 2013
Email: support@wso2.com

Stratos2.0 Cartridge Development Guide

Introduction

Stratos2 ship with some built in cartridge types namely WSO2 Product cartridges, PHP and MySQL. If you want to deploy your own cartridge type in Stratos2 you need to develop your cartridge first. As the definition goes Stratos2 cartridge is a Virtual Machine running in the Stratos2 underlying IaaS + some Stratos2 specific configuration. So developing a cartridge means creating a virtual machine image for the underlying IaaS, installed with the necessary software and configurations and creating/updating configuration in Stratos2 Cloud Controller(CC).

A new cartridge can be added to Stratos2 dynamically. What this mean is that you can add your own cartridge Stratos2 without bringing down the Cloud Controller.

Developing a cartridge involve two steps

1. For each IaaS configured in Stratos2 CC, create a virtual machine image included with the necessary software and configurations.

2. Add the cartridge specific configuration to the Stratos2 CC. This is just copying the cartridge configuration file to the right place. The cartridge configuration file is a xml file and include the image id of the VM image you created for that IaaS. For WSO2 Carbon cartridges it is also necessary to create a service.xml as well in order to maintain domain information in CC.

Currently this process is automated with Puppet . In Stratos2 Paas setup a separate Iaas instance would be used to maintain these puppet scripts and the automation process. This is called the puppet master instance. The configuration changes of WSO2 products will be kept on this instance.

Conventions used in this document

- Configuration file entries are written as in the following example.

```
This is a configuration file entry
```

- Commands typed into linux shell and linux bash scripts are shown as in the following example.

$ example_command parameters

-**Important:** This is very important fact

When something need to be highlighted within a sentence the text is written in **bold**

Creating a Cartridge Image

For each IaaS on which you plan to run Stratos2 cartridges, you need to create an OS image of the cartridge. In this guide we describe how to do this for Openstack and Amazon EC2. However the process is similar to other IaaS's that Stratos2 may support in furure. If you read this guide and understand it and also familiar with other IaaSs then you can easily adapt the process for that IaaS.

Openstack as the IaaS

You can create a cartridge from an already running VM instance on Openstack. This guide assume that the Cartridge image is created by an operations person of an organization who has permission into the Openstack Controller machine. We designate the person who create a cartridge image, as a cartridge developer.

Pre-requisites:

Download the Stratos2 release pack **wso2s2-openstack-1.0.0.zip**. Unarchive it into a folder of your choice. All the scripts required for creating cartridge images are included in **wso2s2-openstack-1.0.0/tools/cartridge_create** folder. From now on we call that folder **CARTRIDGE_HOME**.

cartridge_home folder contain utilities to create customized images from the already running openstack instances. The main utility is a CLI tool called **stratos-image**.

For **kvm** virtual machines download the qcow2 ubuntu base image
precise-server-cloudimg-amd64-disk1.img

For **lxc** containers don't download the qcow2 images, but download the tar.gz file from the download site.

For **lxc** containers download the image precise-server-cloudimg-amd64.tar.gz from
http://cloud-images.ubuntu.com/precise/current/

You need a username and path to private key to access the Openstack controller.

You need the public(floating)ip of the openstack instance from which you need to create the image. Also you need a username for that instance and a private key to access that instance. In addition you need the instance id of the virtual machine from which you need to create the cartridge image.

Steps:

Copy the cartridge_home folder to the Openstack controller. Log into the Openstack Controller.

$ cd cartridge_home

init_scripts/hello is a sample where you will find some files needed to create a simple custom cartridge image for demonstration purposes. The files are

```
hello
hello.ctrg
openstack/get-launch-params.rb
openstack/wso2-cartridge-init.sh
```

hello is the name of our new cartridge. Note that hello folder name, hello file and hello.ctrg should match. Following are the brief description of each file.

hello - This file copy necessary files to the virtual machine instance. You may edit this to include files that you need copied to your cartridge image.

hello.ctrg - This is the script that will be executed to install software in the virtual machine. You may edit this file to install necessary software for your cartridge. This file will be executed after the virtual machine loaded. Since this is a simple demo cartridge we are just creating a file in the /home/ubuntu folder called hello_world.

openstack/get-launch-params.rb - This is the file which process payload passed to your cartridge instance at load time. You may not edit this file.

openstack/wso2-cartridge-init.sh - This file will get executed at the time of your virtual machine load(rc.local). This file contain code to talk to Openstack metadata server and get necessary information like publid ip. It also contain code to join the Elastic load balancer. The necessary code to pull cartridge application source files from a git repository is also included. You may edit this file to customize behaviour of your cartridge at load time.

You can have a look at a more real cartridge like php(init_scripts/php) to see actual details of a cartridge.

You need a file with the following information regarding the Openstack environment and copy that file into the current folder. Let's name that as **demorc**. Normally when you install the Openstack you get this file generated. Please refer to your Openstack installation documents on how to get the information in this file.

```
NOVA_API_HOST=192.168.16.20
GLANCE_API_HOST=192.168.16.20
KEYSTONE_API_HOST=192.168.16.20

NOVA_REGION="nova"

export NOVA_USERNAME=demo
export NOVA_PROJECT_ID=demo
export NOVA_PASSWORD=openstack
export NOVA_API_KEY=openstack
export NOVA_REGION_NAME=$NOVA_REGION
export NOVA_URL="http://$NOVA_API_HOST:5000/v2.0/"
export NOVA_VERSION="1.1"

export OS_USERNAME=demo
export OS_PASSWORD=openstack
export OS_TENANT_ID=7434d33c18e04e60a6751922773fbe2d
export OS_AUTH_URL="http://$KEYSTONE_API_HOST:5000/v2.0/"
export OS_AUTH_STRATEGY="keystone"

export EC2_URL="http://$NOVA_API_HOST:8773/services/Cloud"
export EC2_ACCESS_KEY="8b742fee68c6464198517242828adb82"
export EC2_SECRET_KEY="279a7f28c8a54d5db8e27524f648a7d3"
export S3_URL="http://$GLANCE_API_HOST:3333"
```

Now add the above variables to the environment

$ source ./demorc

First we need to upload the base image to the glance server.

$ glance -I admin -K openstack -T demo -N http://192.168.1.2:5000/v2.0 add name="precise-server-cloudimg-amd64" disk_format=ami container_format=ami distro="ubuntu 12.04" is_public=true < /home/wso2/precise-server-cloudimg-amd64-disk1.img

where /home/wso2/precise-server-cloudimg-amd64-disk1.img is the ubuntu base image you downloaded.

Now execute following command and see whether your base image is uploaded correctly.

$ nova image-list

You will need the image id of the image you just uploaded to create a virtual machine instance.
Now create a key pair to which will be used to access the virtual machine instance you will create.

$nova keypair-add wso2 > wso2.pem
$chmod 0600 wso2.pem

Copy the VM instance private key(wso2.pem) to **init_scripts/<template>** (e.g., i**nit_scripts/hello**) folder.

You also need to add security rules to access the virtual machine instance.

$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0

Now you can create a virtual machine instance

$ nova boot --key_name=wso2 --flavor=1 --image=<image id> <instance name>

Where wso2 is the keypair name you just created and <instance name> is the name you will give to your instance. <image id> is the one you get executing **nova image-list** command for the image you uploaded.

Now you can see your instance by executing

$ nova list

You need instance ip and id for creating cartridge image from this instance in the following commands.
For help on how to execute stratos-image command execute.

$ ./stratos-image help

Using following commands we create and upload the image

$ ./stratos-image 192.168.17.129 hello ubuntu wso2.pem

Here **192.168.17.129** is private ip of the instance you spawned, **hello** is the name of the cartridge

tempalte, **ubuntu** is the virtual machine user, and **wso2.pem** is the private key to access the instance.

$ nova image-create <vm instance id> hello-cartridge-amd64

Here **<vm instance id>** is the instance id of the instance you spawned,
`hello-cartridge-amd64` `is` the name you give to your cartridge image.

Now execute **nova image-list** command and see whether your new cartridge image is listed. You can create an instance from this new cartridge image and ssh into it using the wso2.pem key and verify that hello_world file is created in /home/ubuntu folder.

You will need the cartridge image id to register the cartridge in the WSO2 Cloud Controller.

EC2 as the IaaS

Cartridge creation for EC2 is automated using a puppet, defining a new cartridge type can be done as follow**.**

- A New cartridge type should be defined using puppet language [1].

- To setup puppet in EC2
  - Download and extract wso2s2-ec2-1.0.0.zip to the server.
  - Inside the wso2s2-ec2 you will get s2demo-single-node-installer-ec2.
  - To run the automated installer execute the following commands
    - chmod +x s2demo-single-node-installer-ec2
    - ./s2demo-single-node-installer-ec2
  - This will create /mnt/puppet/stratos2 folder

- First the user has to add definitions and configurations of the new cartridge type to puppet master as follows.
  - To define a new carbon cartridge type you have to define a new puppet class extending the stratos.pp (/mnt/puppet/stratos2/manifests/classes/stratos.pp). stratos.pp contains all the common functionalities related to Carbon servers.

  - For other Non Carbon cartridge types users has to define a new class file in /mnt/puppet/stratos2/manifests/classes/<new_class>.pp (refer an existing class to implement a new class).

- Create a modules and template directory in /mnt/puppet/stratos2/{modules,templates}
  - modules directory should contain the static files (static configuration files, external JARs.), in the folder location defined in the corresponding .pp class.
  - template directory should contain files with variables, which can submit to changes (for carbon servers axis2.xml, carbon.xml, etc.), in the folder location defined in the corresponding .pp class
- Then the user has to create a node definition in the nodes.pp file and add the new class to the node definition.

- Next to create the cartridge install puppet agent in a ec2 instance and create an AMI. Please refer Stratos 2.0 Installation Guide on how to install puppet agent.

Configuring Cloud Controller to support new cartridges

Once the cartridge developer create the cartridge image, the cartridge has to be registered with the Stratos2. This is normally done by a operation team person who maintain Stratos2 deployments.
For this to work one has to do following set of configurations.

What are the configuration files used by Cloud Controller and where do they reside?

In a fresh Cloud Controller pack, you have only the main configuration file which is named as **"cloud-controller.xml"** and resides in **${WSO2-CC}/repository/conf/**. This file mainly consists of 3 parts.

1. **BAM data publisher section**

   This section contains the information related to BAM data publisher of Cloud Controller, such as BAM server information, data publishing task interval etc. By default BAM data publisher is disabled (enable attribute is set to false). We recommend to keep it like that.

```
<dataPublisher enable="false">
        <!-- BAM Server Info - default values are 'admin' and 'admin'
                Optional element. -->
        <bamServer>
        <!-- BAM server URL should be specified in carbon.xml -->
```

```
                <adminUserName>admin</adminUserName>
                <adminPassword
svns:secretAlias="cloud.controller.bam.server.admin.password">admin</adminPassword>
            </bamServer>
            <!-- Default cron expression is '1 * * * * ? *' meaning 'first second of every minute'.
                 Optional element. -->
            <cron>1 * * * * ? *</cron>
</dataPublisher>
```

2. **Topology sync section**

This section contains the information related to topology synchronization, mainly WSO2 Message Broker server information.

```
<topologySync enable="true">
            <!-- MB server info -->
            <mbServerUrl>localhost:5673</mbServerUrl>
            <cron>1 * * * * ? *</cron>
</topologySync>
```

NOTE: Disabling this will result in not syncing up the topology and it is **NOT RECOMMENDED** to do so.

3. **IaaS providers section**

Here, you can define the information of IaaS providers, that may or may not dependent on the different service clusters. You can skip this section and provide IaaS information at cartridge configuration level. But then you need to repeat common IaaS related properties over and over again at each of your cartridge configuration level. This section is provided to make your life easy. You need just override sections in the cartridge configuration files as needed at cartridge level. **We recommend** you to utilize this feature.

```
<!-- Specify the properties that are common to an IaaS here. This element
        is not necessary [0..1]. But you can use this section to avoid specifying
        same property over and over again. -->
 <iaasProviders>
        <!-- type attribute is a must and should be unique. name attribute is optional, and you can add any string
value -->
        <iaasProvider type="ec2" name="ec2 specific details">
            <!-- full qualified name of the IaaS implementation  -->
            <className>org.wso2.carbon.stratos.cloud.controller.iaases.AWSEC2Iaas</className>
```

```
                <!-- provider string -->
                <provider>aws-ec2</provider>
                <!-- identity and credentials of your IaaS account. We use secure vault here. If we cannot find a
value via secure vault, we will look for the text value of this element. -->
                <identity svns:secretAlias="cloud.controller.ec2.identity"></identity>
                <credential svns:secretAlias="cloud.controller.ec2.credential"></credential>
                <!-- these orders will be used when scaling up and down instances. 1 has the highest precedence
        -->
                <scaleUpOrder>1</scaleUpOrder>
                <scaleDownOrder>2</scaleDownOrder>
                <!-- you can define any required property here. -->
                <property name="jclouds.ec2.ami-query"
value="owner-id=XXXX-XXX;state=available;image-type=machine"/>
                <property name="availabilityZone" value="us-east-1c"/>
                <property name="securityGroups" value="default"/>
                <property name="instanceType" value="m1.large"/>
                <property name="keyPair" value="nirmal-key"/>
                 <imageId>us-east-1/ami-XXXXX</imageId>
        </iaasProvider>
        <iaasProvider type="openstack" name="openstack specific details">
                <className>org.wso2.carbon.stratos.cloud.controller.iaases.OpenstackNovaIaas</className>
                 <provider>openstack-nova</provider>
                 <identity svns:secretAlias="cloud.controller.openstack.identity"></identity>
                 <credential svns:secretAlias="cloud.controller.openstack.credential"></credential>
                 <property name="jclouds.endpoint" value="http://192.168.16.20:5000/" />
                <property name="jclouds.openstack-nova.auto-create-floating-ips" value="false"/>
                 <property name="jclouds.api-version" value="2.0/" />
                 <scaleUpOrder>2</scaleUpOrder>
                 <scaleDownOrder>3</scaleDownOrder>
                 <property name="X" value="x" />
                 <property name="Y" value="y" />
                 <imageId>nova/dab37f0e-cf6f-4812-86fc-XXXXXXXXXXXX</imageId>
        </iaasProvider>
 </iaasProviders>
```

Registering your Cartridge with Cloud Controller

You should define the Cartridges that the system going to support, in Cloud Controller. This can be done easily, by **dropping a XML file which defines your Cartridge**, into the "**${WSO2-CC}/repository/deployment/server/cartridges/"** folder. You can do this either before starting the server or **while the server is running**.

Cartridge XML file should have a unique file name and should adhere to any of the following two XML schemas.

1. Cartridge schema - you can define only one Cartridge in a file.
2. Cartridges schema - you can define multiple Cartridges in a single file.

NOTE: There's a limitation in XML 1.0 specification and hence, you need to respect the sequential order of the elements of these configuration files. This limitation is rectified in XML 1.1.

Sample Cartridge XML file (comments in the file explains each element):

```
<!-- You can have 1..n cartridge elements. -->
<cartridge type="php" host="php.slive.com" provider="php">
        <!-- cartridge element can have 0..n properties, and they'll be overwritten by the  properties specified under iaasProvider child elements of cartridge element. -->
        <property name="ss" value="slsls"/>
        <!-- Cartridge description -->
        <description>a php cartridge</description>
        <!-- A cartridge element should add a reference to an existing IaaS provider (specified in the above &lt;iaasProviders&gt; section) or it can create a completely new IaaS Provider (which should have a unique "type" attribute. -->
        <iaasProvider type="openstack" >
            <imageId>nova/250cd0bb-96a3-4ce8-bec8-XXXXXXXXXX</imageId>
            <property name="keyPair" value="demo"/>
        </iaasProvider>
        <!-- This is required for the GIT repo creation. 'baseDir' attribute should point to a valid path in the service instance, and it's the place, where we will clone the GIT repo -->
        <deployment baseDir="xyz">
            <!-- Following directories will be created in the newly created GIT repo. -->
            <dir>abc</dir>
            <dir>abcdef</dir>
        </deployment>
        <!-- Following element is required, if this is a non-carbon cartridge. -->
        <portMapping>
            <http port="80" proxyPort="8280"/>
            <https port="443" proxyPort="8243"/>
        </portMapping>
        <!-- This element will be used to define application types of a cartridge that ca be given a domain mapping. E.g. In WSO2 AppServer cartridge, app types are Axis2Services, Webapps, JaxWebapps, and Jaggeryapps. -->
        <appTypes>
            <appType name="axis2services" />
            <appType name="services" />
            <appType name="webapps" appSpecificMapping="false"/>
        </appTypes>
</cartridge>
```

Note that in the cartridge configuration file you can even configure an IaaS provider. That means you can add a cartridge with a IaaS provider definition and then that IaaS provider is added to the Stratos2 if it is not already added. So this is a mean of dynamically adding an IaaS provider configuration to Stratos2.

This reveals very strong feature of Stratos2. Not only that it can support different IaaS's, but that support can be added dynamically to Stratos2.

Annex

{WSO2-CC}/repository/conf/etc/cartridge.xsd

```
<xs:element name="cartridge">
        <xs:annotation>
        <xs:documentation>You can have 1..n cartridge elements.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:choice maxOccurs="unbounded">
                <xs:element name="property" maxOccurs="unbounded"
                        minOccurs="0">
                        <xs:annotation>
                                <xs:documentation>
                                        cartridge element can have 0..n properties, and
                                        they'll be overwritten by the properties
                                        specified under iaasProvider child elements of
                                        cartridge element.
                                </xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                                <xs:simpleContent>
                                        <xs:extension base="xs:string">
                                                <xs:attribute type="xs:string" name="name" />
                                                <xs:attribute type="xs:string" name="value" />
                                        </xs:extension>
                                </xs:simpleContent>
                        </xs:complexType>
                </xs:element>
                <xs:element name="description" maxOccurs="1" minOccurs="0" />
                <xs:element name="iaasProvider" maxOccurs="unbounded"
```

```
                minOccurs="1">
                <xs:annotation>
                        <xs:documentation>
                                A cartridge element should add a reference to an
                                existing IaaS provider (specified in the above
                                &amp;lt;iaasProviders&amp;gt; section) or it can
                                create a completely new IaaS Provider (which
                                should have a unique "type" attribute.
                        </xs:documentation>
                </xs:annotation>
                <xs:complexType>
                        <xs:choice maxOccurs="unbounded">
                                <xs:element type="xs:string" name="imageId"
                                        minOccurs="0" maxOccurs="1" />
                                <xs:element name="property"
                                        maxOccurs="unbounded" minOccurs="0">
                                        <xs:complexType>
                                                <xs:simpleContent>
                                                        <xs:extension base="xs:string">
                                                                <xs:attribute type="xs:string"
                                                                        name="name" />
                                                                <xs:attribute type="xs:string"
                                                                        name="value" />
                                                        </xs:extension>
                                                </xs:simpleContent>
                                        </xs:complexType>
                                </xs:element>
                        </xs:choice>
                        <xs:attribute type="xs:string" name="type" />
                </xs:complexType>
        </xs:element>
        <xs:element name="deployment" maxOccurs="1" minOccurs="1">
                <xs:complexType>
                        <xs:choice maxOccurs="unbounded">
                                <xs:element name="dir" maxOccurs="unbounded"
                                        minOccurs="0" type="xs:string">
                                </xs:element>
                        </xs:choice>
                        <xs:attribute name="baseDir" type="xs:string">
                        </xs:attribute>
                </xs:complexType>
        </xs:element>
        <xs:element name="portMapping" maxOccurs="1"
                minOccurs="0">
                <xs:complexType>
                        <xs:choice maxOccurs="unbounded">
                                <xs:element name="http" maxOccurs="1"
                                        minOccurs="1">
```

```
                                        <xs:complexType>
                                                <xs:attribute name="port"
                                                        type="xs:string" />
                                                <xs:attribute name="proxyPort"
                                                        type="xs:string" />
                                        </xs:complexType>
                                </xs:element>
                                <xs:element name="https" maxOccurs="1"
                                        minOccurs="0">
                                        <xs:complexType>
                                                <xs:attribute name="port"
                                                        type="xs:string" />
                                                <xs:attribute name="proxyPort"
                                                        type="xs:string" />
                                        </xs:complexType>
                                </xs:element>
                        </xs:choice>
                </xs:complexType>
        </xs:element>
        <xs:element name="appTypes" maxOccurs="1"
                minOccurs="1">
                <xs:complexType>
                        <xs:choice maxOccurs="unbounded">
                                <xs:element name="appType" maxOccurs="unbounded"
                                        minOccurs="1">
                                        <xs:complexType>
        <xs:attribute name="name" type="xs:string" use="required" />
        <xs:attribute name="appSpecificMapping"
                                                        type="xs:string" />
                                        </xs:complexType>
                                </xs:element>
                        </xs:choice>
                </xs:complexType>
        </xs:element>
    </xs:choice>
    <xs:attribute type="xs:string" name="type" />
    <xs:attribute type="xs:string" name="host" />
    <xs:attribute type="xs:string" name="provider" />
  </xs:complexType>
</xs:element>
```

{WSO2-CC}/repository/conf/etc/cartridges.xsd

```
<xs:element name="cartridges" >
        <xs:annotation>
```

```
<xs:documentation>Use below section to specify properties that are needed in order to start Cartridges.</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:choice maxOccurs="unbounded">
<xs:element name="cartridge" maxOccurs="unbounded" minOccurs="1">
<xs:annotation>
<xs:documentation>You can have 1..n cartridge elements.</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:choice maxOccurs="unbounded">
        <xs:element name="property" maxOccurs="unbounded"
                minOccurs="0">
                <xs:annotation>
                        <xs:documentation>
                                cartridge element can have 0..n properties,
                                and they'll be overwritten by the properties
                                specified under iaasProvider child elements
                                of cartridge element.
                        </xs:documentation>
                </xs:annotation>
                <xs:complexType>
                        <xs:simpleContent>
                                <xs:extension base="xs:string">
                                        <xs:attribute type="xs:string"
                                                name="name" />
                                        <xs:attribute type="xs:string"
                                                name="value" />
                                </xs:extension>
                        </xs:simpleContent>
                </xs:complexType>
        </xs:element>
        <xs:element name="description" maxOccurs="1"
                minOccurs="0" />
        <xs:element name="iaasProvider" maxOccurs="unbounded"
                minOccurs="1">
                <xs:annotation>
                        <xs:documentation>
                                A cartridge element should add a reference
                                to an existing IaaS provider (specified in
                                the above &amp;lt;iaasProviders&amp;gt;
                                section) or it can create a completely new
                                IaaS Provider (which should have a unique
                                "type" attribute.
                        </xs:documentation>
                </xs:annotation>
                <xs:complexType>
```

```
                                <xs:choice maxOccurs="unbounded">
                                        <xs:element type="xs:string" name="imageId"
                                                minOccurs="0" maxOccurs="1" />
                                        <xs:element name="property"
                                                maxOccurs="unbounded" minOccurs="0">
                                                <xs:complexType>
                                                        <xs:simpleContent>
                                                                <xs:extension
                                                                        base="xs:string">
                                                                        <xs:attribute
                                                                                type="xs:string"
name="name" />
                                                                        <xs:attribute
                                                                                type="xs:string"
name="value" />
                                                                </xs:extension>
                                                        </xs:simpleContent>
                                                </xs:complexType>
                                        </xs:element>
                                </xs:choice>
                                <xs:attribute type="xs:string" name="type" />
                        </xs:complexType>
                </xs:element>
                <xs:element name="deployment" maxOccurs="1">
                        <xs:complexType>
                                <xs:choice maxOccurs="unbounded">
                                        <xs:element name="dir" maxOccurs="unbounded"
                                                minOccurs="0" type="xs:string">
                                        </xs:element>
                                </xs:choice>
                                <xs:attribute name="baseDir" type="xs:string">
                                </xs:attribute>
                        </xs:complexType>
                </xs:element>
                <xs:element name="portMapping" maxOccurs="1"
                        minOccurs="0">
                        <xs:complexType>
                                <xs:choice maxOccurs="unbounded">
                                        <xs:element name="http" maxOccurs="1"
                                                minOccurs="1">
                                                <xs:complexType>
                                                        <xs:attribute name="port"
                                                                type="xs:string" />
                                                        <xs:attribute name="proxyPort"
                                                                type="xs:string" />
                                                </xs:complexType>
                                        </xs:element>
                                        <xs:element name="https" maxOccurs="1"
```

```
                                        minOccurs="0">
                                        <xs:complexType>
                                                <xs:attribute name="port"
                                                        type="xs:string" />
                                                <xs:attribute name="proxyPort"
                                                        type="xs:string" />
                                        </xs:complexType>
                                </xs:element>
                        </xs:choice>
                </xs:complexType>
        </xs:element>
        <xs:element name="appTypes" maxOccurs="1"
                minOccurs="1">
                <xs:complexType>
                        <xs:choice maxOccurs="unbounded">
                                <xs:element name="appType" maxOccurs="unbounded"
                                        minOccurs="1">
                                        <xs:complexType>
        <xs:attribute name="name" type="xs:string" use="required" />
        <xs:attribute name="appSpecificMapping"
                                                        type="xs:string" />
                                        </xs:complexType>
                                </xs:element>
                        </xs:choice>
                </xs:complexType>
        </xs:element>
    </xs:choice>
    <xs:attribute type="xs:string" name="type"/>
    <xs:attribute type="xs:string" name="host" />
    <xs:attribute type="xs:string" name="provider" />
    </xs:complexType>
    </xs:element>
    </xs:choice>
    </xs:sequence>
    </xs:complexType>
</xs:element>
```

References

[1] http://docs.puppetlabs.com/puppet/2.7/reference/