Connector Development Guide

- Metadata
    - compute-center-types XML Schema
    - Element Definitions
        - compute-center-type
        - Connector-impl-class-name
        - Property-definition
        - Machine-descriptor-definition
    - image-repository-types
    - Image-types
- Implementing the IConnector Interface
    - void setControllerServices(IControllerServices controllerServices)
    - int getAgentPort()
    - createMachine(IComputer computeCenter, IImage image, IMachineDescriptor machineDescriptor)
    - refreshMachineState(IMachine)
    - restartMachine(IMachine)
    - terminateMachine(IMachine)
    - void deleteImage(IImage image)
    - void refreshImageState(IImage image)
    - void validate(IComputeCenter computeCenter)
    - void configure(IComputeCenter computeCenter)
    - void unconfigure(IComputeCenter computeCenter)
    - void validate(IImageStore imageStore)
    - void configure(IImageStore imageStore)
    - void unconfigure(IImageStore imageStore)
    - void validate(IImage image)
    - void configure(IImage image)
    - void unconfigure(IImage image)
- Deploying a Connector

The AppDynamics IConnector interface allows custom implementations of orchestration functionalities such as creating, destroying, restarting, configuring, and validating machine and image instances directly from the AppDynamics Controller user interface.

To create an AppDynamics Connector, the user must implement the IConnector interface and define three sets of xml metadata. The AppDynamics Controller has the ability to dynamically register any new connector every time it starts.

In addition to this topic see the Cloud Connector API Javadoc.

Metadata

The Metadata provides the Controller with the necessary information to register the new compute center, image, and image repository. The Controller uses the metadata to dynamically form the user interface and link the IConnector implementation to the Controller. The three XML files must be named as the following compute-center-types, image-repository-types, image-types.

compute-center-types XML Schema

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="compute-center-types">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="compute-center-type" maxOccurs="unbounded"
minOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="name"/>
              <xs:element type="xs:string" name="description"/>
              <xs:element type="xs:string" name="connector-impl-class-name"/>
              <xs:element name="property-definitions">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="property-definition"
maxOccurs="unbounded" minOccurs="1">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="name"/>
                          <xs:element type="xs:string" name="description"/>
                          <xs:element type="xs:string" name="required"/>
                          <xs:element type="xs:string" name="type"/>
                          <xs:element type="xs:string"
name="default-string-value"/>
                          <xs:element type="xs:string"
name="string-max-length"/>
                          <xs:element type="xs:string"
name="allowed-string-values"/>
                          <xs:element type="xs:string"
name="default-file-value"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="machine-descriptor-definitions">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="property-definition"
maxOccurs="unbounded" minOccurs="1">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="name"/>
                          <xs:element type="xs:string" name="description"/>
                          <xs:element type="xs:string" name="required"/>
                          <xs:element type="xs:string" name="type"/>
                          <xs:element type="xs:string"
name="default-string-value"/>
                          <xs:element type="xs:string"
name="string-max-length"/>
                          <xs:element type="xs:string"
name="allowed-string-values"/>
                          <xs:element type="xs:string"
name="default-file-value"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
```

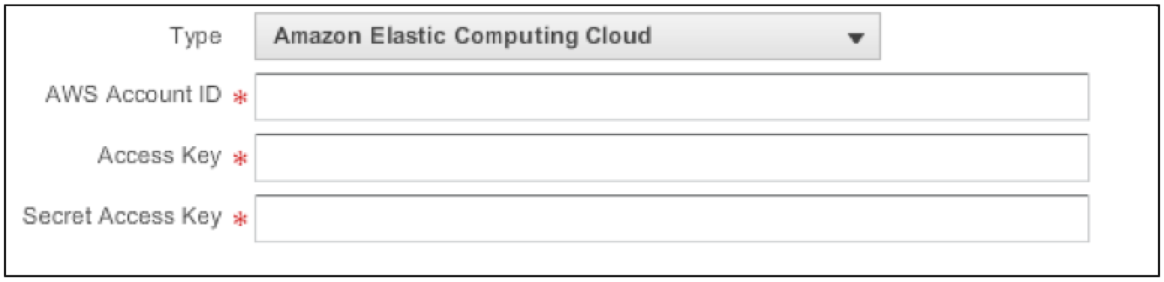
```
</xs:complexType>
```

```
  </xs:element>
</xs:schema>
```

Element Definitions

compute-center-type

List of compute-center-type associated with the IConnector implementation. Each compute-center-type will be registered under the Compute Clouds tab in the UI, for example:



Type	Amazon Elastic Computing Cloud
AWS Account ID *	
Access Key *	
Secret Access Key *	

- **Name**: Compute Center Type name.
- **Description**: Compute Center Type name shown on the Controller GUI.

For example:

```
<description>Amazon Elastic Computing Cloud</description>
```

Connector-impl-class-name

Full name of the IConnector implementation class.

Property-definition

List of property definitions for the compute center. These definitions are used to dynamically generate the property text fields such as AWS Account ID, Access Key, and Secret Access Key.

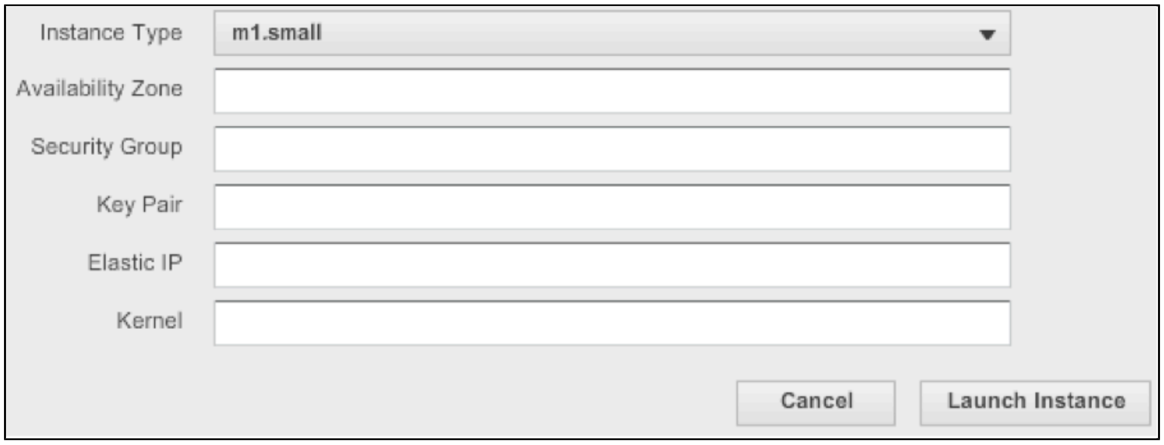
- **Name**: Name of the property field. Such as “AWS Account ID” in Figure 1.
- **Description**: Description of the property.
- **Required**: Checks if the property field cannot be empty. State true/false.
- **Type**: Type of the property field. STRING/FILE
- **Default-string-value**: Default initialization value of the STRING property.
- **String-max-length**: Maximum number of characters allowed to be stored in the property field
- **Allowed-string-values**: List of allowed string values, delimited by comma. If specified this property will be displayed as a drop down list of all the allowed string values.
- **Default-file-value**: Default FILE type property value

For example, using the “AWS Account” property:

```
<property-definition>
    <name>AWS Account ID</name>
    <description>AWS Account ID</description>
    <required>true</required>
    <type>STRING</type>
    <default-string-value></default-string-value>
    <string-max-length>80</string-max-length>
    <allowed-string-values></allowed-string-values>
    <default-file-value></default-file-value>
</property-definition>
```

Machine-descriptor-definition

Contains list of property definitions associated with a machine instance object of that Compute Center type. These properties can be seen in Launch Instance window under the Images tab. For example, an EC2 Launch Instance Window showing Amazon Elastic Computing Cloud Compute Type MachineDescriptor Definitions:



Instance Type	m1.small
Availability Zone	
Security Group	
Key Pair	
Elastic IP	
Kernel	

image-repository-types

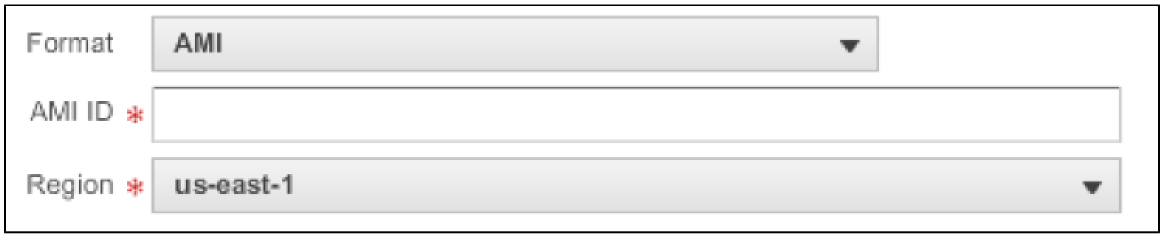
When each compute cloud is registered, a corresponding imagestore object is created. The imagestore name, description, connector-impl-class-name, and property-definitions should mirror the Compute Center property-definitions.

The image-repository-types XML schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="image-repository-types">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="image-repository-type" maxOccurs="unbounded"
minOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="name"/>
              <xs:element type="xs:string" name="description"/>
              <xs:element type="xs:string" name="connector-impl-class-name"/>
              <xs:element name="property-definitions">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="property-definition"
maxOccurs="unbounded" minOccurs="1">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="name"/>
                          <xs:element type="xs:string" name="description"/>
                          <xs:element type="xs:string" name="required"/>
                          <xs:element type="xs:string" name="type"/>
                          <xs:element type="xs:string"
name="default-string-value"/>
                          <xs:element type="xs:byte" name="string-max-length"/>
                          <xs:element type="xs:string"
name="allowed-string-values"/>
                          <xs:element type="xs:string"
name="default-file-value"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Image-types

Each image type defines an image type found under the Images tab. For example:



Format	AMI
AMI ID *	
Region *	us-east-1

The image-types XML schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="image-types">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="image-type" maxOccurs="unbounded" minOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="name"/>
              <xs:element type="xs:string" name="description"/>
              <xs:element name="property-definitions">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="property-definition">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="name"/>
                          <xs:element type="xs:string" name="description"/>
                          <xs:element type="xs:string" name="required"/>
                          <xs:element type="xs:string" name="type"/>
                          <xs:element type="xs:string"
name="default-string-value"/>
                          <xs:element type="xs:byte" name="string-max-length"/>
                          <xs:element type="xs:string"
name="allowed-string-values"/>
                          <xs:element type="xs:string"
name="default-file-value"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="supported-compute-center-types">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="compute-center-type"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Implementing the IConnector Interface

void setControllerServices(IControllerServices controllerServices)

When the connector implementation is created it is passed a handle to the Controller services. The implementor must maintain a reference to this object if it is needed when invoking other connector operations. This method will be most likely implemented as the following:

```
private IControllerServices controllerServices;
public void setControllerServices(IControllerServices controllerServices)
{
 this.controllerServices = controllerServices;
}
```

int getAgentPort()

Retrieve the agent port for machine instances managed through this connector. Most likely it is the value returned by calling the controllerServices.getDefaultAgentPort() method. For example:

```
controllerServices.getDefaultAgentPort();
```

createMachine(IComputer computeCenter, IImage image, IMachineDescriptor machineDescriptor)

Creates a new machine instance in the specified compute center, using the specified image. The type of machine instance and the hardware configuration etc, are specified in the machine instance descriptor. The implementation should try to not block the call until the machine instance is fully started. If possible, it should start the machine instance creation and return a Machine handle with the state set to MachineState.STARTING. The refreshMachineState(IMachine) method can then be called to check when machine instance startup is complete. If the Machine handle is created through the IControllerServices interface then its state will automatically be set to MachineState.STARTING. For example:

```
Public IMachine createMachine(IComputeCenter computeCenter, IImage image,
IMachineDescriptor machineDescriptor) throws InvalidObjectException,
ConnectorException
{
//retrieve property definitions stored in IComputeCenter, IImage, and
IMachineDescriptor //objects
String accountId =
controllerServices.getStringPropertyValueByName(computeCenter.getProperties(),
“Account ID”);
String accessKey =
controllerServices.getStringPropertyValueByName(computeCenter.getProperties(),
“Access Key”);
String imageId =
controllerServices.getStringPropertyValueByName(image.getProperties(), “Image
ID”);
String machineSize =
controllerServices.getStringPropertyValueByName(machineDescriptor.getProperties(),
“Image Size”);
//To grab a file property, the code must iterate through the corresponding
property //array. For example:
Byte[] bytes = null;
For(IProperty i: image.getProperties)
{
 If (i.getDefinition.getType == PropertyType.FILE)
{
 If (i.getDefinition.getName().equals(“Property Name”);
       {
         bytes = ((IFileProperty)i).getFileBytes();
  Break;
       }
}
}

/*
Code to create a new machine instance on the specified compute center
*/
int agentPort = controllerServices.getDefaultAgentPort();
IMachine machine = controllerServices.createMachineInstance(“machine instance
id”, “unique internal name”, computeCenter, machineDescriptor, image,
agentPort);
return machine;
}
```

refreshMachineState(IMachine)

The refreshMachineState method is used by the controller to poll the status of a particular IMachine object. If the MachineState of an IMachine object is not marked as STARTED, the controller will poll its status at an interval. If the MachineState is set as STARTED, the controller will assume the machine instance is running and will not invoke this method. For example:

```
public void refreshMachineState(IMachine machine) throws
InvalidObjectException, ConnectorException
{

//get the IMachine instance name, usually stores the corresponding machine
instance id
String machineId = machine.getName();

//Each IMachine object has access to its IComputeCenter, IMachineDescriptor,
and IImage //objects
String zone =
controllerServices.getStringPropertyValueByName(machine.getMachineDescriptor().get
"Zone");

/*
Code to grab machine instance status
Invoke any post build actions
*/
/*
The IMachine object Machine state must be set accordingly.

MachineState.STOPPED will mark the IMachine as stopped and delete the IMachine
object from the controller. Machinestate can be set manually to STOPPED if the
machine instance no longer exists
MachineState.STARTED will set the IMachine as started. The controller will no
longer poll the machine instance status in STARTED state.
*/

machine.setState(MachineState.STARTING);
}
```

restartMachine(IMachine)

Restarts the specified machine instance. The implementation should try to not block the call until the machine instance is fully restarted. If possible it should begin the machine restart and return immediately. Afterwards the Machine handle state will be set automatically to MachineState.STARTING. The refreshMachineState(IMachine) method can then be called to check when machine instance restart is complete.

terminateMachine(IMachine)

Terminates the specified machine instance. The implementation should try to not block the call until the machine instance is fully terminated. If possible, it should start the machine termination and return immediately. Afterwards the Machine handle state will be set automatically to MachineState.STOPPING. The refreshMachineState(IMachine) method can then be called to check when machine instance termination is complete.

void deleteImage(IImage image)

Delete the image from the underlying dynamic capacity provider.

void refreshImageState(IImage image)

Refreshes the image state of the specified image. This callback is needed since the image save and image copy can take a considerable amount of time to complete. Instead of blocking on the image save and image copy operations, the connector implementation should return quickly. Afterwards, the controller will poll the image state through this operation to see when the image save or image copy is complete. When complete, it is the responsibility of the connector implementation to set the image state to ImageState.READY.

void validate(IComputeCenter computeCenter)

Validates if a Compute Center has the valid Properties and rest of the fields. For example:

```
public void validate(IComputeCenter computeCenter) throws
InvalidObjectException, ConnectorException
{

String accountId =
controllerServices.getStringPropertyValueByName(computeCenter.getProperties(),
“AWS Account ID”);

String accessKey =
controllerServices.getStringPropertyValueByName(computeCenter .getProperties(),
“Access Key”);

String secretKey =
controllerServices.getStringPropertyValueByName(computeCenter .getProperties(),
“Secret Access Key”);


/*
Validate if the user credentials are correct. Throw exception if properties are
invalid.
*/

}
```

void configure(IComputeCenter computeCenter)

Does any configuration that needs to be done for the Compute Center to be able to be used by Singularity Dynamic Provisioning Manager. This may include creating access accounts, creating permissions, etc.

void unconfigure(IComputeCenter computeCenter)

Un-does any configuration that was done for the Compute Center to be able to be used by Singularity Dynamic Provisioning Manager. This may include deleting access accounts, deleting permissions, etc.

void validate(IImageStore imageStore)

Validates if an Image Store has the valid Properties and rest of the fields.

void configure(IImageStore imageStore)

Does any configuration that needs to be done for the Image Store to be able to be used by Singularity Dynamic Provisioning Manager. This may include creating access accounts, creating permissions, etc.

void unconfigure(IImageStore imageStore)

Un-does any configuration that was done for the Image Store to be able to be used by Singularity Dynamic Provisioning Manager. This may include deleting access accounts, deleting permissions, etc.

void validate(IImage image)

Validates if an Image has the valid Properties and rest of the fields.

void configure(IImage image)

Does any configuration that needs to be done for the Image to be able to be used by Singularity Dynamic Provisioning Manager. This may include creating access accounts, creating permissions, etc.

void unconfigure(IImage image)

Un-does any configuration that was done for the Image to be able to be used by Singularity Dynamic Provisioning Manager. This may include deleting access accounts, deleting permissions, etc.

For more information regarding the IController interface and its related classes, see the Cloud Connector API Javadoc.

Deploying a Connector

To deploy a connector
1. Compile the IConnector implementation into a Jar file.
2. Package the compiled jar and the xml metadata into one directory.
3. Put the directory in: <controller-install-dir>/lib/connectors.
4. Restart the Controller by running the stopController and startController scripts under <controller-install-dir>/bin.

You should see the new connector in the Compute Clouds and Images tab under Systems.