Plugin Development Guide

v.0.9

Table of Contents

Introduction

ConVirt team is pleased to announce the availability of Platform Plugin Development guide with this release. This allows developers to add a plugin for a specific virtualization platform. Once the plugin is deployed, ConVirt users can use the same user interface and features for the new platform. This document describes architecture of ConVirt followed by details on developing such a plugin. The Xen plugin implementation provides as a good reference.

Philosophy

The over all philosophy for the project is to allow for platform specific configuration while maintaining reasonable standardization to provide a unified view of VM life cycle management.

ConVirt Overview

For developing a plugin, it is necessary to have a good understanding of ConVirt model and some inner-workings. This section provides necessary details to achieve it.

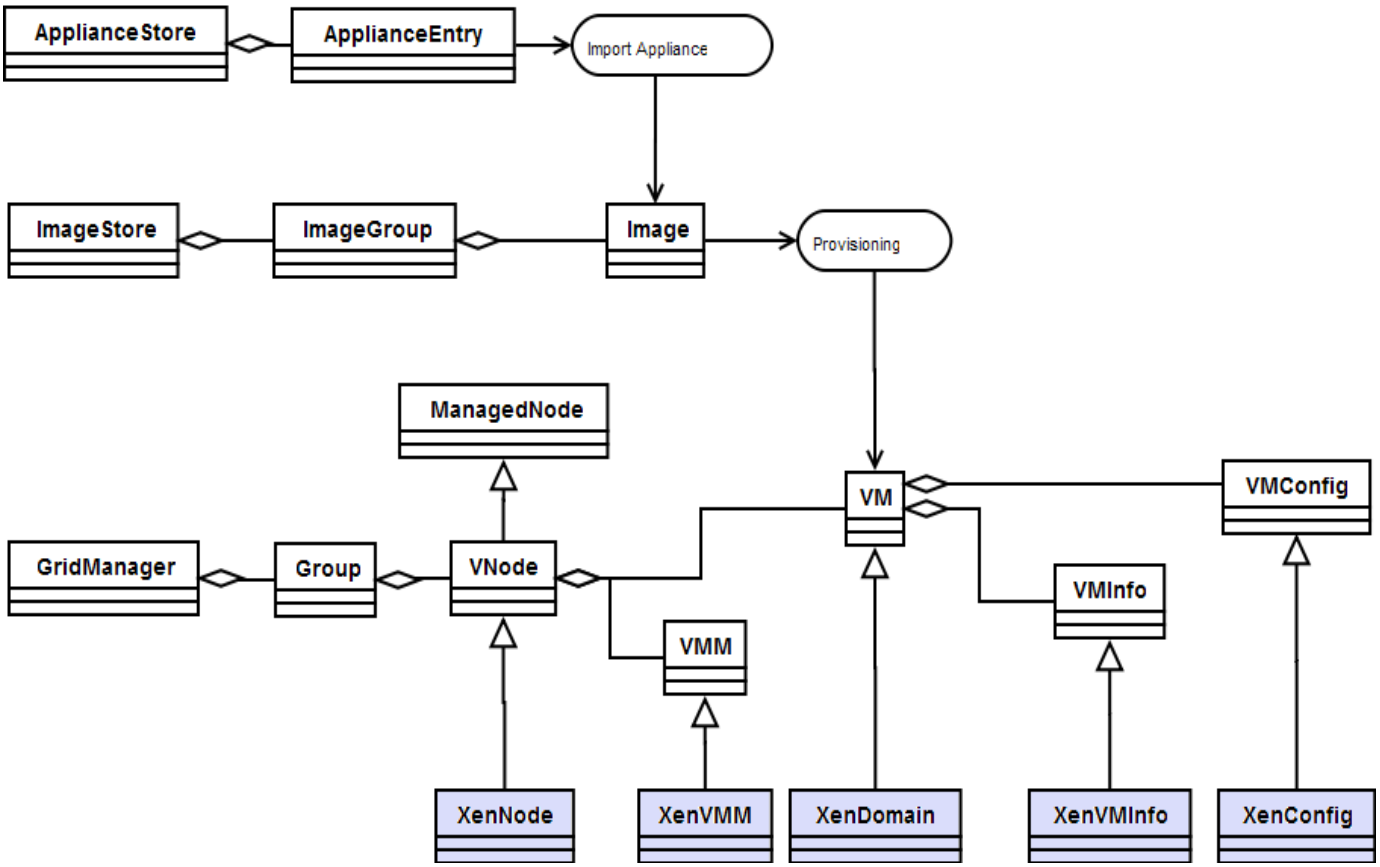
Packages

The following table outlines some of the packages in ConVirt.

Package	Description
core/model	All core classes representing model. For example, VM representing a Virtual Machine, ManagedNode and VNode representing managed server and managed virtualized server respectively etc. This package also contains GridManager, ImageStore, ApplianceStore etc.
core/appliances	Contains model for representing appliance catalog, and classes needed for transforming an appliance in to a ConVirt image. It also contains sub packages for appliance vendor integration.
core/utils	Contains utility classes and constants. Notably NodeProxy and Paramiko helper for doing remote operations via ssh.
core/repos	Reserved for having ConVirt repository related details
core/platforms	Platform specific model implementation.
client/	The client package contains the code for GTK base ConVirt client, utilities, glade files, as well as platform specific classes for UI.
server/	Reserved for ConVirt server
web/	Reserved for ConVirt web application
cli/	Reserved for CLI

Model

This section describes important entities and their relationships. Here is a UML diagram for the core components.



GridManager : Manages Groups (Server groups) and Managed Nodes (Servers).
Managed Node : A server being managed.

VNode : Virtualized Node, a node with Virtualized Platform / VMM running on it.

VMM : The Virtual Machine manager. It implements all interactions with the VMM for a specific platform.

VM : A Virtual Machine. A VM contains runtime information (VMInfo) and the configuration information (VMConfig).

ImageStore : Manages image groups and images.

ApplianceStore : Catalog of appliances provided by Appliance Vendors like rPath and JumpBox.

PlatformRegistry : Configuration information about available virtualization platforms.

Platform : Main Integration point for a platform plugin. Provides platform specific factories and related information about a plugin implementation.

PlatformUIHelper : Interface to be implemented by a platform plugin to customize the UI specific to the platform.

VMSettings : UI for editing images, VMConfigs as well as provisioning UI. Plugin developers can customize this for their platforms.

VMInfoHelper : A helper class that implements display of VM information in ConVirt UI.

Overall Flow and UI

This section describes program flow for initialization and few other important user work flows.

Initialization : On startup ConVirt initializes UI artifacts and important model items. The GridManager initialization loads the list of Server Pools and Servers while the Image Store initialization reads the image groups and images within them. The GridManager uses the Platform specific NodeFactory to create VNodes. The left navigation tree in ConVirt UI gets populated with the information from the GridManager and the Image Store. The Summary tab of the Notebook also gets initialized depending on the selection. ConVirt refreshes the Summary information with current CPU and memory information periodically.

Provisioning : The provisioning operation opens up the Settings Dialog (VMSettings or derived class) and allows the user to customize the image information. It then sets up the environment and kicks of the provisioning script.

Edit VM Settings : It reads in the VM configuration and allows the user to change it. If the VM is running, the user can also make changes to it. For example, memory allocated to a running VM can be changed.

Edit Image : It reads in the image configuration as well as VM configuration template and allows the user to customize it.

Appliance Import : The Appliance import begins with presenting appliance catalog using ApplianceList. On selection, The Appliance Import dialog shown and then the import is kicked off. The import operation typically opens the appliance package, create necessary VM template and image config files. The selection of templates (meta-template) is done in a platform specific fashion.

VM Lifecycle Ops : The VM operations are initiated using menu or button bar and the implementation in the VM/VMM gets invoked. The VNode acts as a factory to create VMs of specific platform type. While the VM acts as a Factory to create VMConfig and VMInfo in a similar fashion.

Plugin Development

Armed with this background, lets see what is involved in developing the actual plugin. ConVirt has a registry mechanism to get required integration points from the plugin. The Platform and PlatformUIHelper are the two interfaces which needs to be implemented by the plugin developer. The Platform interface helps tie in the platform specific core model, while the PlatformUIHelper helps with UI integration points. Overall the following steps needs to be taken. In the following sections each one of these are addressed in detail.

- Develop Platform specific model classes
- Develop Platform specific UI classes
- Specify Integration files
- Platform specific images
- Packaging

Platform Specific Core Model

This section specifies steps to implement platform specific core classes that are required to be implemented by a platform plugin.

- Implement platform specific VMM derived class. This is responsible for communicating with the actual VMM for the platform. In case of Xen the XenVMM class communicates with the Xen VMM using the xml-rpc API.
- Implement the VMInfo that encapsulates the required runtime VM information using dynamic_map. VMM would return instances of these when the runtime information is requested by the client. XenInfo is implemented in XenVMM.py which takes care of giving map like interface to the information received in sxp format.
- Implement platform specific VMConfig class. This would represent the configuration for the VM. It is recommended to use the VMConfig derived class for ease of use. If the platform requires specific config format, override the read and write methods for custom read/write. It is required that the VMConfig allows the configuration parameters to be accessed similar to a map. In case of Xen the XenConfig implementation takes care of this.
- Implement the platform specific VM class. Typically contains references to the platform config and platform's runtime VM information and keeps track of the state.
- Implement the VNode : Implement a platform specific VNode class. This typically handles creating platform specific VMM. Apart from this is also responsible for implementing platform specific migration checks, and returning a snapshot of all VMs. The XenNode class implements all these for the Xen platform.
- Implement the VNodeFactory that implements creating instance of platform specific VNode and other methods to read and write node information to the repository. This allows ConVirt to create correct type of platform node.
- Implement the Platform class. This class would be used by ConVirt to interact with model specific classes mentioned in the previous steps. In addition, the platform detection code and prerequisites checks needs to be implemented. Check the XenPlatform class for the Xen platform.

Platform specific UI Helpers

This section specifies steps to implement various UI artifacts that are required to be implemented by a platform plugin.

- Implement the UI helper class for the platform. This would require some of the following classes to be implemented.
- Implement AddNode dialog for the platform. This should return an instance of platform specific

node. And allow an existing node to be edited.

- Use the VMSettings dialog as the settings dialog for the platform. As a second pass, you might want to customize the VMSettings.

  The VMSettings dialog is used for different operations based on mode.
  - Editing VM config (mode = EDIT_VM_CONFIG)
  - Editing runtime VMInfo (mode = EDIT_VM_INFO)
  - Editing an Image (mode = EDIT_IMAGE)
  - Provisioning an Image (mode = PROVISION_IMAGE)

  The VMSettings dialog follows a pattern of

  - get context from the caller
  - initialize widgets based on mode
  - scatter information from the context to the UI element
  - let the user make changes
  - validate information from UI elements
  - gather information from UI elements
  - perform operation (saving/provisioning) on the gathered information.

  You can extend the dialog for customizing the UI. XenSettings dialog is implemented, that adds the bootloader, ramdisk and kernel parameters to the VMSettings dialog.

Integration

This section specifies the steps to populate proper registry entries for ConVirt to find the plugin information.

- Specify the registry file for the platform at the following location.
  ConVirt`/core/platforms/<platform>/registry.`
- Update the master platform registry to include the platform. This is located at
  ConVirt`/core/platforms/registry`

Platform specific Images

This section describes steps to add a new platform specific image to the image store.

- Under the image store create a directory containing image. Typically consisting of
- a template for the VM configuration
- image.conf which takes care of providing additional context for the provisioning script
- the provisioning script
- any additional data / disks that are required for the provisioning operation.
  You may customize or write your own provisioning script. ConVirt would invoke the script by giving it the vm_config and image_config context at the time of provisioning.
- Use the add_image.py utility in the CLI area to add register the image with the image store.
  `$ src/convirt/cli/add_image.py <image_name> <platform> <image_group>`
- You may also want to specify the platform specific meta-templates under the src/ConVirt/appliance/<platform>/. These would be used when an appliance of for this platform is being imported using catalog or location specified manually.

Packaging

For now simply provide the plugin as a tar ball which would overlay existing ConVirt installation. Include a readme and other documents too.

Feedback

Drop us an email to our mailing list or write post details on our forums at www.convirt.net.

FAQ

This section has few related FAQ entries.

-- How does ConVirt and Platform plugin work together ?

All the context required by the ConVirt is provided via the the platform registry. The platform registry allows the plugin developer to specify implementations required. In particular, implementations for Platform interface and PlatformUIHelper interfaces are expected.

Refer to : registry, Platform, PlatformUIHelper

-- How does ConVirt create appropriate VNode instance corresponding to the platform ?

The integration class implements the get_factory method that returns the appropriate factory class for creating VNode instances. The plugin developer also need to implement a dialog to add/edit node for its particular platform. Each node when persisted saves the platform along with it.

Refer to : VNode, VNodeFactory, XenNode, XenNodeFactory

-- How to get list of VMs managed by VMM on a particular Server (node). ?

The Platform Node is expected to return a list of managed VMs on the node. The VNode class implements most of these functionality with some assumptions. It assumes that VM configuration files would be present in some specific directory and the VMM class would return the list of running VMs. If a specific platform has different way of managing list of VM configuration, implement the get_dom_list method for the VNode. Note : The VNode acts as a factory for the platform specific VM instance.

Refer to : VNode, XenNode

-- How to get stats for all VMs running on a Server ?

The platform specific VNode is also responsible for getting a snapshot of state and stats (cpu, memory, disk i/o, network i/o) for all VMs. This is done by implementing the get_metric_snapshot method. The base class provides polling and caching facilities.

Refer to : XenNode

-- How to do VM ops, i.e. Start/Stop/Pause/Save/Restore/Connect to Console etc. ?

These VM operations are delegated to the VMM class. See XenVMM for more information. For getting to the console of a running VM, is_graphical, get_vnc_port and get_console_cmd methods need to be implemented by the VM class. The is_graphical method returns true, if the VM's console is graphical and can be reached via vnc. The get_console_cmd is invoked to get the command line. This command line is used to get to the console of the VM shown in the Terminal.

Refer to : VM, VMM, XenDomain, XenVMM

-- How is configuration for a particular VM is stored. ?

The base VMConfig class represents the configuration in the python config format. For complex configuration, one can derive a specific class from VMConfig and overwrite the read and write method. It is required to keep the map like access to the various elements of the configuration.

Refer to : PyConfig, VMConfig, DomListHelper, XenConfig

-- How should the VMInfo information be presented in the UI ?

ConVirt gives flexibility for a plugin to control how VM information is displayed. This is done via customizing the VMInfoHelper class.

Refer to : VMInfoHelper, XenInfoHelper

-- How should I display Platform specific information about a Server/Node ?
The platform specific VNode can implement the get_platform_info method to return the platform specific information. This would be displayed in a separate category with Server information.
Refer to : VMNode, XenNode