oVirt Extension API

The first step for fully modular oVirt

Martin Peřina
Software Engineer at Red Hat
Agenda

- Introduction
- oVirt Engine Extension API
- Extension API for AAA
- Extension API for Logging
- Extension API Tools
- Future plans
Introduction
What is oVirt?

- Large scale, centralized management system for server and desktop virtualization based on Linux/KVM
Extending oVirt prior to version 3.5

• Extensibility of oVirt < 3.4 was limited, the project consisted of two monolithic parts: Engine and VDSM

• **Engine UI Plugins**
  – enables to create a plugin for UI that is able to communicate with Engine using REST API

• **VDSM Hooks**
  – enable to execute custom script/command at certain predefined points in the flow
What is AAA?

- **Authentication**
  - Verification of identity that is trying to access the system

- **Authorization**
  - Verification of resources that identity is allowed to access

- **Accounting**
  - Statistics of resource usage by identity
AAA status in oVirt <= 3.4

- Complex implementation using Kerberos
- Insecure (no SSL/TLS, no SSO)
- No proper support for multi-domain setup
- No customization (monolithic module, logic and schema hard-coded)
- Not optimized (always recursive, sub-optimal LDAP queries)
Monolithic or extension based?

- Monolithic methodology is never flexible enough for customization not considered during initial design

- “Easy” to extend without breaking backward compatibility

- Not specific to AAA, but usable for whole oVirt Engine

- Possibility to write extensions in other languages (provided by JVM)

- Ability for extension to extension interaction
oVirt Engine Extension API
Engine Extension API

- Introduced in oVirt 3.5
- Currently used only for AAA and logging, but available for all parts of engine
- JBoss Modules is used to load extensions
- Extension's configuration is stored in properties files
Engine Extension API

- Primitive invoke-based interface, parameters are passed as maps

```java
package org.ovirt.engine.api.extensions;

public interface Extension {
    void invoke(ExtMap input, ExtMap output);
}
```
Map keys contain meaningful name, UUID and type:

```java
public static final ExtKey COMMAND = new ExtKey(
    "EXTENSION_INVOKE_COMMAND",
    ExtUUID.class,
    "485778ab-bede-4f1a-b823-77b262a2f28d"
);

public static final ExtKey RESULT = new ExtKey(
    "EXTENSION_INVOKE_RESULT",
    Integer.class,
    "0909d91d-8bde-40fb-b6c0-099c772ddd4e"
);
```
Common types for all extensions are placed in org.ovirt.engine.api.extensions package:

- **ExtUUID**
  - contains UUID and descriptive name

- **ExtKey**
  - consists of ExtUUID and type

- **ExtMap**
  - defined as Map<ExtKey, Object>
  - contains run-time type enforcement to value with key type information
Engine Extension API

- **Base**
  - contains common constants for all extensions:

- **InvokeKeys**
  - keys of input/output maps for invoke() method

- **InvokeCommands**
  - available commands:
    - LOAD, INITIALIZE, TERMINATE

- **InvokeResult**
  - result of invoke() method execution:
    - SUCCESS, UNSUPPORTED, FAILED
Engine Extension Configuration

- Extension configuration is stored in a property file which has to contain some mandatory options and may contain other extension specific options.

- Configuration files should be placed under one of those directories:
  - `/etc/ovirt-engine/extensions.d`
  - `/usr/share/ovirt-engine/extensions.d`

- Configured extensions are loaded during engine start-up.
ovirt.engine.extension.name = myextension

ovirt.engine.extension.bindings.method = jbossmodule

ovirt.engine.extension.binding.jbossmodule.module = org.ovirt.engineengineextensions.myext

ovirt.engine.extension.binding.jbossmodule.class = org.ovirt.engineengineextensions.myext.MyExtension

ovirt.engine.extension.provides = org.ovirt.engine.api.extensions.Extension
Extension Start-up Flow

1. Engine startup process
2. EngineExtensionsManager
3. Extensions Configuration Directory
4. Extension Instance

- engineInitialize()
- Get all config files
- Invoke LOAD
- Invoke INITIALIZE
ExtensionsManager class

- Provides internal API for engine to access extensions
- Uses Observer pattern to notify about extension updates
- It provides methods to access extensions:
  - `List<ExtensionProxy> getExtensionsByService(String provides)`
  - `ExtensionProxy getExtensionByName(String name)`
  - `List<ExtensionProxy> getLoadedExtensions()`
  - `List<ExtensionProxy> getExtensions()`
ExtensionProxy class

- Each loaded extensions is decorated with **ExtensionProxy** instance

- **ExtensionProxy** simplifies invoke() method execution:
  - Returns output map
  - Catches exceptions in case of a failure
  - Makes problem determination easier
Engine Extension API for Logging
Logger

- Constants related to logger extensions are stored in `org.ovirt.engine.api.extensions.logger.Logger`

- Provides ability to:
  - Publish log record to logger extension
  - Flush log records in logger extension
  - Close logger extension
Logger extensions

- **logger-log4j**
  - Provided by ovirt-engine-extension-logger-log4j package
  - Provides log4j appenders for oVirt Engine
  - Can be used for example to pass oVirt Engine log records to syslog
Code sample
Engine Extension API for AAA
AAA – Authentication (aka authn)

- Constants related to authentication extensions are stored in `org.ovirt.engine.api.extensions.aaa.Authn`

- Goal:
  - Verify the user that tries to access system

- Input:
  - User name and password or
  - HTTP negotiation

- Output:
  - Authentication record which contains principal and validity time interval
AAA – Authorization (aka authz)

• Constants related to authorization extensions are stored in `org.ovirt.engine.api.extensions.aaa.Authz`

• Goal:
  – Provide details about user

• Input:
  – Principal

• Output:
  – Authentication record with additional information (user details, set of groups, etc.)
AAA – Accounting (aka acct)

- Constants related to accounting extensions are stored in `org.ovirt.engine.api.extensions.aaa.Acct`
- Provides framework for security related events (successful/unsuccessful login, logout, etc.)
- It will provide full auditing capability in future
AAA – Mapping

- Constants related to mapping extensions are stored in `org.ovirt.engine.api.extensions.aaa.Mapping`
- Provides:
  - mapping of user name before authn
  - mapping of principal before authz
- Examples:
  - removing Kerberos suffix from user name before SSO
  - removing domain name from principal before accessing LDAP
AAA – Filters

- Set of servlet filters to handle authentication:
  - Supports negotiation using authz extensions.
  - Supports basic authentication.
  - HTTP session management.
Existing AAA extensions

- **internal**
  - Built-in into Engine
  - Provides only admin user to login to oVirt
  - Mostly used only in development environment, in production oVirt 3.6+ it's replaced with **aaa-jdbc**

- **kerberosldap**
  - Built-in into Engine
  - The legacy mixed kerberos/ldap implementation
  - It's deprecated in 3.6 (may be removed in 4.0) and should be replaced with **aaa-ldap**
Existing AAA extensions

- **aaa-ldap**
  - Provided by `ovirt-engine-extension-aaa-ldap` package
  - Interface for users/groups stored in LDAP server
  - Supports most of LDAP servers
  - Can be fully customized using configuration files

- **aaa-misc**
  - Provided by `ovirt-engine-extension-aaa-misc` package
  - Contains miscellaneous utilities for AAA
  - Can be use to configure kerberos support for oVirt using Apache mod_auth_kerb
Existing AAA extensions

- **aaa-jdbc**
  - New in oVirt 3.6
  - Provided by `ovirt-engine-extension-aaa-jdbc` package
  - Interface for users/groups stored in PostgreSQL database
  - Provides command line tool `ovirt-aaa-jdbc-tool` to manage users/groups
  - Replaces *internal* extension in production environment
Engine Extension API Tools
Engine Extension API Tools

- **ovirt-engine-extensions-tool**
  - New in oVirt 3.6
  - Provides ability to show information about installed extensions (list, show configuration, ...)
  - Enables testing of the extension functionality without running engine
Future plans
AAA – Future plans

- ovirt-engine-extension-aaa-sssd
  - sssd support.

- SSO service for oVirt applications
  - Move Authn to its own application, modify userportal, webadmin, reports to trust AAA application.

- We are currently planning what other parts of engine will expose its features via Extension API in oVirt 4.0
THANK YOU!

http://www.ovirt.org
mperina@redhat.com
mperina at #ovirt (irc.oftc.net)