Cloud.com CloudStack Adapter

Framework Guide

September 8, 2010

Contents

1 Overview

The CloudStack™ Adapter Framework allows you to further customize the CloudStack platform to suit your business needs. We'd like to think that our default adapter implementations should be sufficient enough to suit your needs but we also know that every business has unique requirements. This framework will allow you to build your own adapters and easily integrate within the Cloud.com CloudStack platform. We currently support custom implementation of the following:

- **UserAuthenticators**. Allows you to integrate custom authentication schemes that the Management Server can use to validate user credentials.
- **HostAllocators**. Allows you to create custom rules to determine which physical host to allocate the guest virtual machines on.
- **PodAllocators**. Allows you to create custom rules to determine which pod to allow the guest virtual machines on.
- **ConsoleProxyAllocators**. Allows you to create custom rules to determine which routing nodes to allow the console proxy on.
- **Investigators**. Allows you to create custom rules to determine when a virtual machine is detected as down for HA purposes.

1.1 Prerequisites

The CloudStack Adapter Framework is built entirely using Java as the programming language. To get started, you will need the following.

- The CloudStack Management Server installed and running (for more information, refer to the Cloud.com CloudStack Installation Guide)
- Understanding of XML (Refer to the W3C XML Tutorial)
- Understanding of the Java programming language
- Understanding of how the Cloud.com CloudStack works

2 UserAuthenticators

User authenticators are written by extending the com.cloud.server.auth.DefaultUserAuthenticator interface. The interface contains one method.

```
        /**
         * Authenticates the user by username and password.
         *
         * @param username
         * @param password
         * @param domainId
         * @return true if the user has been successfully authenticated, false otherwise
         **/

        public boolean authenticate(String username, String password, Long domainId);
```

A custom adapter can be written by implementing this one method. The following is an example custom authenticator.

2.1 Example Custom Authenticator

```
import javax.ejb.Local;
@Local(value={com.cloud.server.auth.UserAuthenticator.class})

public class MyAuthenticator extends com.cloud.server.auth.DefaultUserAuthenticator {
    public static final Logger s_logger = Logger.getLogger(MyAuthenticator.class);

    public boolean authenticate(String username, String password, Long domainId) {
        if (s_logger.isDebugEnabled()) {
            s_logger.debug("Retrieving user: " + username);
        }
        UserAccount user = _userAccountDao.getUserAccount(username, domainId);
        if (user == null) {
            s_logger.debug("Unable to find user with " + username + " in domain " + domainId);
            return false;
        }

        MessageDigest md5;
        try {
            md5 = MessageDigest.getInstance("MD5");
        } catch (NoSuchAlgorithmException e) {
            throw new CloudRuntimeException("Error", e);
        }
        md5.reset();
        BigInteger pwInt = new BigInteger(1, md5.digest(password.getBytes()));

        // make sure our MD5 hash value is 32 digits long...
        StringBuffer sb = new StringBuffer();
        String pwStr = pwInt.toString(16);
        int padding = 32 - pwStr.length();
        for (int i = 0; i < padding; i++) {
            sb.append('0');
        }
        sb.append(pwStr);

        if (!user.getPassword().equals(sb.toString())) {
            s_logger.debug("Password does not match");
            return false;
        }
        return true;
    }
}
```

2.2 Integrating a Custom Authenticator

1. Write a custom authenticator class, based on the example in the previous section.
2. Package the code into a JAR file.
3. Copy the JAR file to /usr/share/java.

4. Modify the components.xml and components-premium.xml files found in /usr/share/cloud/management/conf as follows.

   Look for the following in both files.

   ```
   <adapters key="com.cloud.server.auth.UserAuthenticator">
        <adapter name="MD5" class="com.cloud.server.auth.MD5UserAuthenticator"/>
   </adapters>
   ```

   Replace the MD5UserAuthenticator with your class name. Optionally, you can change the name of the adapter as well. You will also need to modify /usr/share/cloud/management/etc/tomcat6.conf and append the path and name of your jar to the end of the CLASSPATH.

5. Restart the Management Server by typing “service cloud-management restart”.