Handbuch: GenericAPI

Silvan Adrian
Fabian Binna

1 Änderungshistorie

Datum	Version	Änderung	Autor
14.12.15	1.00	Erstellung des Dokuments	Fabian Binna

Inhaltsverzeichnis

2 Einleitung

In diesem Dokument soll beschrieben werden wie ein ComputeController für die LibVirt Library implementiert werden kann.

3 Package

Jede Implementation einer Library gehört in ein eigenes Package. In diesem Fall in das Package sddc.genericapi.libvirt. Dies dient hauptsächlich der Übersicht.

4 Provider

Damit der neue Provider (LibVirt) als solches erkannt werden kann, muss im Enumerator sddc.services.domain.Provider ein Eintrag "LibVirt"hinzugefügt werden.

5 Category

Es existieren bereits die Kategorien Compute, Storage und Network. Es währen auch andere Kategorien, wie Database, denkbar. In unserem Fall benötigen wir also keine neue Kategorie.

6 LibVirtController

Der LibVirtController ist eine abstrakte Klasse, die den Konstruktor definiert. Es können auch noch Hilfsmethoden implementiert werden, falls nötig.

6.1 LibVirtController Code

```
1  public abstract class LibVirtController
       extends ResourceController {
3
       protected Connect connect;
5
       public LibVirtController(
7          Category category, Connect connect) {

9          super(category, Provider.LibVirt);
           this.connect = connect;
11     }

13     protected String replaceUUID(String config) {
           UUID uuid = UUID.randomUUID();
15         return ConfigUtil.
           changeValue(config, "{{UUID}}", "" + uuid);
17     }

19 }
```

Im Konstruktor werden Libraryabhängige Klassen (Connect), sowie die Kategory übergeben. Die Kategory wird später von den konkreten Implementationen an den LibVirtController weitergeleitet. Der Kostruktor übergibt die Kategory dann nochmals eine Stufe weiter an den ResourceController und definiert mit Provider.LibVirt die eigene Zugehörigkeit. Die Hilfsmethode ReplaceUUID wird für die Bearbeitung der Configfiles benötigt.

7 LibVirtComputeController

Der LibVirtComputeController ist die konkrete Implementation des ResourceController.

7.1 ResourceController Code

```
public abstract class ResourceController {

  private final Category category;
  private final Provider provider;

  public ResourceController(Category category,
    Provider provider) {

    this.category = category;
    this.provider = provider;
  }

  /**
   * Creates the resource described in the ServiceModule
   * and returns an identifier
   * @param module A ServiceModule describing a
   *  Resource/Module
   * @return An Identifier that stores information
   *  about the created Resource
   */
  public abstract Identifier create(ServiceModule module);

  /**
   * Deletes an existing resource
   * @param identifier
   */
  public abstract void delete(Identifier identifier);

  /**
   * Gets information about an existing Resource/Module
   * @param identifier
   * @return
   */
  public abstract Map<String, String>
    getInformations(Identifier identifier);

  public Category getCategory() {
    return category;
  }

  public Provider getProvider() {
    return provider;
  }
}
```

Es müssen also die abstrakten Methoden create, delete und getInformations implementiert werden. Dabei ist die Art der Implementation unwichtig. Wichtig ist nur der Contract, der Methoden.

create
Anhand der Informationen im ServiceModule wird die Resource erstellt. Als Rückgabe muss ein Identifier erstellt werden, der zum Beispiel eine UUID beinhaltet.

delete
Mit dem Identifier kann die Resource erreicht, und somit auch gelöscht werden.

getInformations
Die Methode getInformations ist undefiniert. Das heisst es ist unklar welche Informationen sich in der Map befinden. Je nach Library können mehr oder weniger Informationen abgerufen werden. Es ist also praktisch unmöglich eine klare Definition über den Inhalt zu machen.

7.2 LibVirtComputeController Code

```
public class LibVirtComputeController
      extends LibVirtController {

  public LibVirtComputeController(Connect connect) {
    super(Category.Compute, connect);
  }

  @Override
  public Identifier create(ServiceModule module) {

    if(module == null)
      return null;

    String config = replaceUUID(module.getConfig());


    try {
    Domain domain = connect.domainDefineXML(config);
    domain.create();
    return new Identifier(  module.getName(),
    domain.getUUIDString(),
    module.getCategory(),
    module.getSize(),
    module.getProvider());

    } catch(LibvirtException libvirtException) {
      return null;
    }
  }
```

```
  @Override
  public void delete(Identifier identifier) {

    if(identifier == null)
      return;

    try {
      Domain domain =
      connect.
      domainLookupByUUIDString(
      identifier.getUuid());

      domain.destroy();
      domain.undefine();
    } catch(LibvirtException libvirtException) {
    }
  }

  @Override
  public Map<String, String>
  getInformations(Identifier identifier) {

    Map<String, String> infos = new HashMap<>();

    if(identifier == null)
      return infos;

    DomainInfo domainInfo;
    String domainName;

    try {
      domainInfo = connect.
      domainLookupByUUIDString(
        identifier.getUuid()).getInfo();

      domainName = connect.
      domainLookupByUUIDString(
        identifier.getUuid()).getName();

    } catch(LibvirtException libvirtException) {
      return infos;
    }
    infos.put("name", domainName);
    infos.put("memory", String.
      valueOf(domainInfo.memory));
    infos.put("vcpu", String.
      valueOf(domainInfo.nrVirtCpu));
    //...

    return infos;
  }

}
```

Im Konstruktor wird die Klasse Connect weitergegeben und die Kategory des Controllers definiert (Compute). Die Implementationen der ResourceController Methoden sind nicht immer leicht, da jede Library einen anderen Ansatz hat. Die Exceptions helfen hier nur wenig, da sie nicht in die höheren Schichten weitergeleitet werden dürfen und ohnehin für den Benutzer keine relevanten Informationen beinhalten. Das Logging der Exceptions wurde hier aus Gründen der Übersicht entfernt. Falls eine Resource nicht erstellt (create) werden konnte wird einfach null zurückgegeben. Der Workflow entscheidet dann was geschehen soll. Im Moment wird bei einem Fehler ein Rollback gemacht.

8 Dependency Injection

Die GenericAPI wird schlussendlich mit Dependency Injection zusammengesetzt. Die Konfiguration befindet sich im File Config.xml, welches nach dem ausrollen der Software im Rootverzeichnis liegt.

8.1 Config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans

    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <bean id="LibVirtConnect" class="org.libvirt.Connect">
  <constructor-arg value="test:///default" type="String">
  </constructor-arg>
  <constructor-arg value="false" type="boolean">
  </constructor-arg>
  </bean>

  <bean id="LibVirtComputeController"
 class="sddc.genericapi.libvirt.LibVirtComputeController">
  <constructor-arg><ref bean="LibVirtConnect" />
  </constructor-arg>
  </bean>

  <bean id="LibVirtNetworkController"
```

```
  class="sddc.genericapi.libvirt.LibVirtNetworkController">
   <constructor-arg><ref bean="LibVirtConnect" />
   </constructor-arg>
   </bean>

   <bean id="LibVirtStorageController"
  class="sddc.genericapi.libvirt.LibVirtStorageController">
   <constructor-arg><ref bean="LibVirtConnect" />
   </constructor-arg>
   </bean>

   <bean id="LibVirtServiceModuleHandler"
  class="sddc.genericapi.ServiceModuleHandler">
   <constructor-arg>
     <list>
       <ref bean="LibVirtComputeController" />
       <ref bean="LibVirtNetworkController" />
       <ref bean="LibVirtStorageController" />
     </list>
   </constructor-arg>
   </bean>

</beans>
```

Als erstes wird die Klasse Connect erstellt. Wie schon erwähnt handelt es sich hier um eine Klasse aus der LibVirt Library. Bei anderen Libraries kann die Instanziierung also komplett anders ausfallen. Wichtig ist hier, dass die Connect Klasse mit test:///default instanziiert wird, ansonsten funktionieren die Tests nicht. Danach können die konkreten Controller (LibVirtComputeController) definiert werden. Hier muss dann nur noch die Connect Klasse übergeben werden. Am Ende wird noch der LibVirtServiceModuleHandler benötigt, der alle Controller bündelt.