Configuration

Abstract

CloudStack is an orchestration platform that supports different plugins to support different storage, network, hypervisor technologies. Often, these plugins provide configurations that system admins can tune. The package cloud-framework-config provides this functionality. This page describes how cloud-framework-config works and how to add configuration.

Requirements

The package cloud-framework-config must support the following:

- Dynamic reloading of configuration parameters that can be dynamic reloaded.
- Give priority to scoped overrides of configuration parameters. Currently, the scopes are Zone, Pod, Cluster, StoragePool, and Account.
- Allow different components in CloudStack to provide configuration parameters without modifying the CloudStack code
- Easily figure out where the configuration parameter was declared and used.

How to Use

Declaring a configuration parameter

cloud-framework-config provides a class, ConfigKey<T>, for configuration parameter declaration. In order to use it, declare it as follows:

```
public interface ClusterManager {
    final ConfigKey<String> ManagementHostIPAdr = new ConfigKey<String>("Advanced", String.class, "host",
"localhost", "The ip address of management server", true);
    ....
}
```

In the above declaration, the final statement makes the key definition a constant.

Each key must be declared with the following information. Please give careful thought to what you enter.

- category - A category from which an admin can retrieve related configuration parameters
- type - Type for the value. Supported values are Short, Long, Integer, String, Boolean, Date
- key - Name of the parameter when stored.
- default - Default value if no override is found.
- description - A description of what the configuration parameter changes and what are the correct values to input.
- dynamic - Can the configuration value be changed without restarting the server.

Keys can optionally contain the following information. They are specified a in a different constructor.

- scope - Scope at which overrides can be found for that specific scope. Currently, the scopes are Zone, Pod, Cluster, StoragePool, and Account. When a configuration parameter has a scope other than Global, override is checked first. If an override is not found, then the global value is returned.
- mulitiplier - A convenience value for number valued configuration parameter to multiple the set value with this multiplier. The most useful case for this is if the configurable range is in seconds but actual usage must be in milliseconds.

Retrieving a configurable value

In order to retrieve the value from the configuration parameter, perform the following. Note that if the configuration parameter is meant to be used by multiple components, it should probably be declared in a common interface or class. **Note that you should never save the return value if you want to dynamically change if the configurable value is changed**.

```
// Use case 1: To retrieve the global value
ManagementHostIPAdr.value();


// Use case 2: To retrieve the parameter but first look for the override in zone.
// Note that this will give an error for this parameter as it was not declared as a scoped parameter.
ManagementHostIPAdr.valueIn(zoneId);


// Use case 3: To use configuration that's declared by a different component.
public class NotChildOfClusterManager {
    ClusterManager.ManagementHostIPAddr.value();
}


// Use case 4: To use a configuration that you know it's there but you can't compile against.
// I strongly advise against this.  If a developer did not put their configuration parameter in
// an interface for you to use, it's best to communicate with that developer and make sure you
// agree on where to place it.
public class NoCompilationAccessToClusterManager {
  @Inject ConfigDepot _depot;
  ConfigKey<?> mgmtAddr = _depot.get("host");
}
```

Managing the configuration parameter

To manage the configuration parameter, cloud-framework-config provides a Configurable interface. By implementing this interface, the component declares itself to be a provider of configuration keys. The keys and the default values are saved into the configuration table.

```
public interface Configurable {

    /**
     * @return The name of the component that provided this configuration
     * parameter.  This value is saved in the database so someone can easily
     * identify who provides this parameter.
     **/
    String getConfigComponentName();

    /**
     * @return The list of config keys provided by this configuable.
     */
    ConfigKey<?>[] getConfigKeys();
}
```

How do I find which component declared a parameter I want to use?

Sharing configuration parameter should be done after some thought. You want to make sure the original implementer intended the configuration parameter to be shared. A clear indication would be if they defined it on an interface. If not, it might have been just because they didn't think about it.

To find out which component defined the parameter, look in the component field of the configuration table. It will show which component defined the parameter that you might want to use.

How does it work?

On startup, ConfigDepotImpl gathers all of the Configurable interfaces and constructs a set of keys. It checks to see if the key is already in the database. If not, then it populates the database with the default values. If a key that is in the database but is no longer found in the key list, the updated field is changed to null. At which point, an admin can decide to delete the obsolete key. Going forward, any caching or special implementation will go into ConfigDepotImpl and no other components need to modify the core CloudStack code to add their own configuration parameters.

What to do on CloudStack upgrades?

On every CloudStack upgrade, it will go through the list of configuration keys. If a key is obsolete, it changes the update field of the row to null. That key can be deleted or just left alone. If the update field has been changed with the latest timestamp, it means some field in that row has changed and required updating. The upgrades never changes the value field as that belongs to the administrator but it might change the default value field to show our defaults have changed. If you didnot make changes to the value of the configuration parameter, you might want to update it to be the same as the new default value.

Todo Items

Todo items are found in the comments for ConfigDepotImpl. Please add/remove on it if you take up different items.