Extensions

Discovery and Ordering

Extensions are automatically discovered based on the interfaces they implement and which module is their parent. For example, if you place a storage extension in a child module of the network module, it will not be discovered. Additionally, depending on the extension, the ordering may be important in how they extension is used. For the extensions that ordering is important there is also a "Global Configuration" setting for the ordering. The value of the setting is a comma seperated list like SHA256SALT,MD5,LDAP,PLAINTEXT. The values are the names of the extension. The name will be the result of getName(), if the bean implements the Named interface, or the short class name (ie getClass().getSimpleName()).

If the order parameter is configured as A,B,C but the extensions that truly exist are A,C,D, the effective order will be A,C,D. Specifically configured items in the list that do not exist are ignored, if items are found that are not in the list, they are appended to the end of the list. If you do not want D to be loaded, you must add D to the exclude configuration property, which is always the same name as the order configuration, but ending with .exclude. For example, the order setting for com.cloud.server.auth.UserAuthenticator is user.authenticators.order and the exclude setting user.authenticators.exclude.

Inteface	Parent Module	Order Configuration	Description	Additional Info
com.cloud.agent.manager.allocator.HostAllocator	allocator	none		
com.cloud.consoleproxy.ConsoleProxyAllocator	allocator	none		
com.cloud.server.auth.UserAuthenticator	api	user.authenticators.order		
org.apache.cloudstack.acl.SecurityChecker	api	security.checkers.order		
org.apache.cloudstack.acl.APIChecker	api	api.checkers.exclude		
com.cloud.ha.Investigator	compute	ha.investigators.order		
com.cloud.ha.FenceBuilder	compute	none		
com.cloud.hypervisor.HypervisorGuru	compute	none		
com.cloud.utils.component.PluggableService	core	none		
com.cloud.resource.Discoverer	discoverer	none		
com.cloud.network.element.IpDeployer	network	none		
com.cloud.network.element.DhcpServiceProvider	network	none		
com.cloud.network.guru.NetworkGuru	network	none		
com.cloud.network.element.NetworkElement	network	none		
com.cloud.deploy.DeploymentPlanner	planner	deployment.planners.order		
org.apache.cloudstack.affinity.AffinityGroupProcessor	planner	affinity.processors.order		
org.apache.cloudstack.engine.subsystem.api.storage.DataStoreProvider	storage	none		
org.apache.cloudstack.engine.subsystem.api.storage.StoragePoolAllocator	storage	storage.pool.allocators.order		
com.cloud.storage.secondary.SecondaryStorageVmAllocator	storage	none		
com.cloud.template.TemplateAdapter	storage	none		
com.cloud.agent.manager.allocator.PodAllocator	storage	none		
org.apache.cloudstack.engine.subsystem.api.storage.SnapshotStrategy	storage	none		
org.apache.cloudstack.engine.subsystem.api.storage.DataMotionStrategy	storage	none		
org.apache.cloudstack.region.gslb.GslbServiceProvider	network	none		

Configuration File Based Exclusion

Discovery of modules is based on classpath scanning. In some situations it may not be possible or desirable to change the classpath. To force a module to not load you can create a file called modules.properties on the classpath that can exclude specific modules from loading. Additionally this same file can be used to exclude a specific extension. Extension loading is typically done through global configuration. If you want to setup an environment and you don't even want the extension/module loaded on the first start, then using the config file is appropriate.

Example: modules.properties

```
modules.exclude=storage-image-s3,storage-volume-solidfire
extensions.exclude=ClusterScopeStoragePoolAllocator,ZoneWideStoragePoolAllocator
```

Typically you would want to place this file in /etc/cloudstack/management