

---

# CloudStack 2.2

## Allocator Implementation

short description



open source cloud computing

### Dude McPants

Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Apache CloudStack is an effort undergoing incubation at The Apache Software Foundation (ASF).

Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.

CloudStack® is a registered trademark of the Apache Software Foundation.

Apache CloudStack, the CloudStack word design, the Apache CloudStack word design, and the cloud monkey logo are trademarks of the Apache Software Foundation.

### Abstract

A short overview and summary of the book's subject and purpose, traditionally no more than one paragraph long. Note: the abstract will appear in the front matter of your book and will also be placed in the description field of the book's RPM spec file.

1. Implementing a custom HostAllocator .....	2
1.1. Host Allocator Interface .....	2
1.2. Reference HostAllocator Implementation .....	4

## Allocator Implementation

---

1.3. Loading a custom HostAllocator .....	4
2. Custom Storage Pool Allocator .....	4
2.1. Custom Storage Pool Allocator Interface .....	4
2.2. Reference StoragePoolAllocator Implementation .....	5
2.3. Loading a custom StoragePoolAllocator .....	6

Allocators are part of the CloudStack Adapters. There are following categories of allocators currently supported:

1. HostAllocators: Allows you to create custom rules to determine which physical host to allocate the guest virtual machines on.
2. StoragePoolAllocators: Allows you to create custom rules to determine which storage pool to allocate the guest virtual machines on.

## 1. Implementing a custom HostAllocator

### 1.1. Host Allocator Interface

HostAllocators are written by extending `com.cloud.agent.manager allocator.HostAllocator` interface. The interface defines following two methods.

```
/**
 * Checks if the VM can be upgraded to the specified ServiceOffering
 * @param UserVm vm
 * @param ServiceOffering offering
 * @return boolean true if the VM can be upgraded
 */
public boolean isVirtualMachineUpgradable(final UserVm vm, final ServiceOffering offering);

/**
 * Determines which physical hosts are suitable to allocate the guest virtual machines on
 *
 * @param VirtualMachineProfile vmProfile
 * @param DeploymentPlan plan
 * @param Type type
 * @param ExcludeList avoid
 * @param int returnUpTo
 * @return List<Host> List of hosts that are suitable for VM allocation
 */
public List<Host> allocateTo( VirtualMachineProfile<?extends VirtualMachine> vmProfile,
    DeploymentPlan plan, Type type, ExcludeList avoid, int returnUpTo);
```

A custom HostAllocator can be written by implementing the 'allocateTo' method

Input Parameters for the method 'HostAllocator :: allocateTo':

1. `com.cloud.vm.VirtualMachineProfile` *vmProfile*

VirtualMachineProfile describes one virtual machine. This allows the adapters like Allocators to process the information in the virtual machine and make determinations on what the virtual machine profile should look like before it is actually started on the hypervisor.

HostAllocators can make use of the following information present in the VirtualMachineProfile:

- a. The `ServiceOffering` that specifies configuration like requested CPU speed, RAM etc necessary for the guest VM.
  - b. The `VirtualMachineTemplate`, the template to be used to start the VM.
2. `com.cloud.deploy.DeploymentPlan`  
DeploymentPlan should specify:
  - a. `dataCenterId`: The data center the VM should deploy in.
  - b. `podId`: The pod the VM should deploy in; null if no preference.
  - c. `clusterId`: The cluster the VM should deploy in; null if no preference.
  - d. `poolId`: The storage pool the VM should be created in; null if no preference
3. `com.cloud.host.Host.Type` *type*  
Type of the Host needed for this guest VM. Currently `com.cloud.host.Host.Type` interface defines the following Host types:
  - a. Storage
  - b. Routing
  - c. SecondaryStorage
  - d. ConsoleProxy
  - e. ExternalFirewall
  - f. ExternalLoadBalancer
4. `com.cloud.deploy.DeploymentPlanner.ExcludeList` *avoid*  
The `ExcludeList` specifies what datacenters, pods, clusters, hosts, storagePools should not be considered for allocating this guest VM. HostAllocators should avoid the hosts that are mentioned in `ExcludeList.hostIds`
  - a. `SetLong` `dclIds`;
  - b. `SetLong` `podIds`;
  - c. `SetLong` `clusterIds`;
  - d. `SetLong` `hostIds`;
  - e. `SetLong` `poolIds`;
5. `int` `returnUpTo`  
This specifies return up to that many available hosts for this guest VM. To get all possible hosts, set this value to -1

### 1.2. Reference HostAllocator Implementation

Refer to `com.cloud.agent.manager allocator.impl.FirstFitAllocator` that implements the `HostAllocator` interface. This allocator checks available hosts in the specified datacenter, Pod, Cluster and considering the given Service Offcering requirements.

If `returnUpTo = 1`, this allocator would return the first Host that fits the requirements of the guest VM

### 1.3. Loading a custom HostAllocator

Write a custom `HostAllocator` class, implementing the interface described above.

Package the code into a JAR file and make the JAR available in the classpath of the Management Server/tomcat

Modify the `components.xml` and `components-premium.xml` files found in `/client/tomcatconf` as follows:

Search for 'HostAllocator' in these files.

```
<adapters key="com.cloud.agent.manager.allocator.HostAllocator">
  <adapter name="FirstFit" class="com.cloud.agent.manager.allocator.impl.FirstFitAllocator"/>
</adapters>
```

Replace the `FirstFitAllocator` with your clas name. Optionally, you can change the name of the adapter as well.

Restart the Management Server

## 2. Custom Storage Pool Allocator

`StoragePoolAllocators` are written by extending `com.cloud.storage.allocator.StoragePoolAllocator` interface.

### 2.1. Custom Storage Pool Allocator Interface

A custom `StoragePoolAllocator` can be written by implementing the 'allocateTo' method

```
/**
 * Determines which storage pools are suitable for the guest virtual machine
 * @param DiskProfile dskCh
 * @param VirtualMachineProfile vmProfile
 * @param DeploymentPlan plan
 * @param ExcludeList avoid
 * @param int returnUpTo
 * @return List<StoragePool> List of storage pools that are suitable for the VM
 */
public List<StoragePool> allocateToPool(DiskProfile dskCh, VirtualMachineProfile<? extends
VirtualMachine> vm, DeploymentPlan plan, ExcludeList avoid, int returnUpTo);
```

This interface also contains some other methods to support some legacy code. However your custom allocator can extend the existing `com.cloud.storage.allocator.AbstractStoragePoolAllocator`. This class provides default implementation for all the other interface methods.

Input Parameters for the method 'StoragePoolAllocator :: allocateTo':

1. `com.cloud.vm.DiskProfile` *dskCh*

`DiskCharacteristics` describes a disk and what functionality is required from it. It specifies the storage pool tags if any to be used while searching for a storage pool.

2. `om.cloud.vm.VirtualMachineProfile` *vmProfile*

`VirtualMachineProfile` describes one virtual machine. This allows the adapters like Allocators to process the information in the virtual machine and make determinations on what the virtual machine profile should look like before it is actually started on the hypervisor.

StoragePoolAllocators can make use of the following information present in the `VirtualMachineProfile`:

- a. The `VirtualMachine` instance that specifies the properties of the guest VM.
- b. The `VirtualMachineTemplate`, the template to be used to start the VM.

3. `com.cloud.deploy.DeploymentPlan` *plan*

`DeploymentPlan` should specify:

- a. `dataCenterId`: The datacenter the VM should deploy in.
- b. `podId`: The pod the VM should deploy in; null if no preference
- c. `clusterId`: The cluster the VM should deploy in; null if no preference.
- d. `poolId`: The storage pool the VM should be created in; null if no preference.

4. `com.cloud.deploy.DeploymentPlanner.ExcludeList` *avoid*

The `ExcludeList` specifies what datacenters, pods, clusters, hosts, storagePools should not be considered for allocating this guest VM. StoragePoolAllocators should avoid the pools that are mentioned in `ExcludeList.poolIds`:

- a. `SetLong` `dclIds`;
- b. `SetLong` `podIds`;
- c. `SetLong` `clusterIds`;
- d. `SetLong` `hostIds`;
- e. `SetLong` `poolIds`;

5. `int` `returnUpTo`

This specifies return up to that many available pools for this guest VM. To get all possible pools set this value to -1

## 2.2. Reference StoragePoolAllocator Implementation

Refer to `com.cloud.storage.allocator.FirstFitStoragePoolAllocator` that implements the `StoragePoolAllocator` interface. This allocator checks available pools in the specified datacenter, Pod, Cluster, and considers the given `DiskProfile` characteristics.

## Allocator Implementation

---

If `returnUpTo = 1`, this allocator would return the first Storage Pool that fits the requirements of the guest vm

### 2.3. Loading a custom StoragePoolAllocator

Write a custom `StoragePoolallocator` class, implementing the interface described above.

Package the code into a JAR file and make the JAR available in the classpath of the management server/tomcat.

Modify the `components.xml` and `components-premium.xml` files found in `/client/tomcatconf` as follows:

Search for 'StoragePoolAllocator' in these files.

Replace the `FirstFitStoragePoolAllocator` with your class name. Optionally, you can change the name of the adapter as well.

Restart the management server