The Tag System

by Frank Zhang on Apr 4, 2015

Tags in ZStack not only help users aggregate resources but also control behaviors of the software. ZStack has a complete specification defining category, form, and usage of tags. Besides users, plugins can create their tags too, in order to record metadata and extend properties of existing resources; by all these means, tags can help plugins to introduce new features without changing ZStack's database schema, diminishing the need for database migration in software upgrade.

The motivation

With increasing resources in the cloud, users may want a way to group similar resources with human readable labels, for example, all VMs of web servers with a label 'web-tier-vm', in order to retrieve them as a group from UI or CLI. For IaaS itself, the pre-defined business logic may never satisfy users' requirements; cite creating VM as an example, the default algorithm for selecting target host may just randomly pick up one from a host pool, but users may want various algorithms that fit their scenarios; such as selecting a host with more than 8G memory, selecting a host having SR-IOV (http://en.wikipedia.org/wiki/X86_virtualization#PCI-SIG_Single_Root_I.2FO_Virtualization_.28SR-IOV.29) hardware, or selecting a host already having VMs of the same user running. The IaaS software can hardly provide individual API for all of the endless, unpredictable requirements, there must be a mechanism allowing base APIs(e.g. APICreateVmInstanceMsg) to carry extra information.

Depending on their business logic, plugins may choose to or not to create database schemas. For example, the Open vSwitch L2 network, as creating a new type of resource, may need a new schema; however, a plugin that allows reservation of host memory may not need to create a separate schema but attach a piece of data to hosts. If the IaaS software doesn't give a way for plugins to attach data, they will start either creating new, trivial schemas or modifying existing schemas to add their columns, which causes hairy situation in database migration of software upgrade.

Finally, for third-party software built upon ZStack, allowing them to store information into ZStack's database could avoid data integrity problem and enable them using ZStack's comprehensive query APIs(see details in The Query API (query.html)).

The problem

Most IaaS software have had the concept of tags. However, not all of them have defined an exhaustive specification of tags for different scenarios. For example, some of them may use tags for user aggregating resources while some of them may use tags for internal business logic. ZStack, instead, carefully designs a specification defining every aspect of tags for different scenarios.

The tag system

In ZStack, tags are essentially strings carrying small pieces of information that associate with resources. A tag is mainly made up of below fields:

FIELD	DESCRIPTION
uuid	tag's UUID
resourceUuid	resource UUID that the tag is associated with
resourceType	resource type that the tag is associated with
tag	a string that contains meaningful information
type	tag type: System or User

The fundamental difference between ZStack and other IaaS software for tags is ZStack categorizes tags into two classes: **User** and **System**.

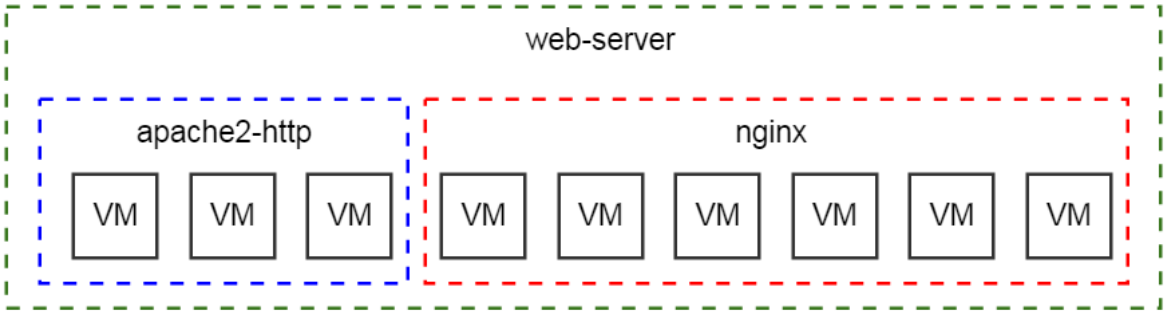
1. User tags

User tags, as implied by the name, are tags created by users for grouping resources. For example, grouping VMs that install Apache2 HTTP server by a tag 'apache2-http'; then users can retrieve those VMs using a query API with tag 'apache2-http' as a query condition:

```
QueryVmInstance __userTag__=apache2-http
```

> ***Note**: See The Query API (query.html) for details.*

This is the most common usage of tags. A resource can have multiple tags associated, grouped in different logic groups.



Accompanying with system tags, user tags can also be used to control behaviors of ZStack. For example, if a user tag `SSD` has been created on a primary storage, a system tag can instruct ZStack to create VM's root volume on only primary storage with user tag `SSD`. In this case, user tags are more like resources' metadata input by users. We will see soon that plugins can also create resources' metadata using system tags.

2. System tags

Unlike user tags that can be created by users at any time with any arbitrary values, system tags, which have fixed format of value, are pre-defined by ZStack orchestration services and plugins, and can be used in below scenarios:

2.1 Metadata

Plugins can use system tags to record resources' metadata. For example, host's database table doesn't have columns to capture metadata such as hypervisor version, hypervisor SDK version; however, the derived host plugin, for example, the KVM host plugin, may need those metadata in order to decide if some feature is available on current hypervisor; for instance, the supportability of live volume snapshot of KVM is determined by libvirt and QEMU version. In ZStack, the KVM host plugin saves OS version, libvirt version, QEMU version, and qemu-img tool version as system tags when connecting to the backend hosts.

```
QuerySystemTag fields=tag resourceUuid=d07066c4de02404a948772e131139eb4


{
  "inventories": [
      {
          "tag": "capability:liveSnapshot"
      },
      {
          "tag": "qemu-img::version::2.0.0"
      },
      {
          "tag": "os::version::14.04"
      },
      {
          "tag": "libvirt::version::1.2.2"
      },
      {
          "tag": "os::release::trusty"
      },
      {
          "tag": "os::distribution::Ubuntu"
      }
  ],
  "success": true
}
```

2.2 Resource properties

Plugins can also use system tags to add new properties to a resource. For example, the VM's database schema has no column recording what IP allocation algorithm to use when allocating VM nic. This sort of extra property can be implemented as system tags. There is no limit to number of system tags a plugin can create; auxiliary plugins can leverage this and avoid bothering with database schemas.

> ***Database Schema vs System Tags**: As both database schema and system tags can define a resource's properties, it's sometimes confusing whether a property should be a column of the database schema or a system tag in a separate table. Modifying an existing database schema to add new columns normally needs database migration that is a main pain of IaaS software upgrade so that developers may prefer system tags for all new properties. However, abusing system tags is a wrong programming way. By ZStack's convention, properties in the form of system tags should only be used to introduce non-inherent resource properties; system tags are not made for rescuing poorly-designed database schema. For example, if cluster UUID is missing from VM's database schema (though it won't), it must be added back even a database migration is required, but a department ID introduced by a plugin made by a user for private use should be implemented as a system tag. This kind of tradeoff is not easy to take sometimes, we will put strict eyes on any database schema changes.*

2.3 Metaprogramming

System tags can also annotate resources in order to influence ZStack's execution flows, which is similar to Metaprogramming (http://en.wikipedia.org/wiki/Metaprogramming) somehow. For example, administrator can create a system tag `reservedMemory::1G` on a KVM host, hinting ZStack host allocator to reserve 1G memory from available memory of that host; if the administrator changes his or her mind, he or she can remove the tag to return the 1G memory back then. There are lots of similar system tags; for example, in *User tags* section we mentioned the cooperation of a user tag `SSD` and a system tag to specify destination primary storage for VM's root volume; the system tag is called `primaryStorage::allocator::userTag::{tag}::required`; if `primaryStorage::allocator::userTag::SSD::required` is created on an instance offering, root volumes of VMs created from that instance offering will be allocated to only primary storage with user tag `SSD`. There are many so-called `interpreting points` that will seek for specific system tags during execution, which can change default behaviors of the code.

2.4 Thirdparty software integration

Thirdparty software built upon ZStack can use system tags to store information along with resources in ZStack's database; this is particularly useful to avoid data inconsistency between thirdparty software's database and ZStack's database. For example, a private software may record department ID of VM for auditing usage of IT resource from each department, which is usually done in a private database, and forces the private software to track VM lifecycle because it needs to update own database when a VM is created or destroyed. Otherwise, the data will incorrectly reflect the truth. With help of system tags, the private software could use a system tag, for example, `audit::departmentId::{id}`, to store the information in ZStack's database, transferring the responsibility for managing lifecycle of department ID to ZStack; when a VM is being destroyed, it's department ID(e.g. `audit::departmentId::1`) will be deleted automatically in the same database transaction of deleting the VM row. Furthermore, the private software can retrieve VMs by looking up their department ID using regular query API:

```
QueryVmInstance fields=uuid __sysTag__=audit::departmentId::1
```

> ***Note**: At this ZStack version (0.6), we haven't opened the interface that allows defining arbitrary system tags, all system tags are pre-defined. We plan to open this interface in next version, user-defined system tags can be created with some allowed prefix, for example, `3rd::`.*

Relationship to other components

The tag system is one of the core ZStack components; it not only has separate APIs and services, but also integrates with other core components seamlessly. Users can create tags at the time a resource is being created or after the resource is created. All ZStack creational APIs support two inherent parameters: `userTags` and `systemTags`, tags passed in them will be created along with resources. For example:

```
CreateVmInstance name=testTag systemTags=hostname::web-server-1 l3NetworkUuids=6572ce44c3f6422d8063b0fb262cbc62
instanceOfferingUuid=04b5419ca3134885be90a48e372d3895 imageUuid=f1205825ec405cd3f2d259730d47d1d8
```

If the resource has been existing, users can create or delete tags using tag API:

```
CreateUserTag resourceType=VmInstanceVO resourceUuid=613af3fe005914c1643a15c36fd578c6 tag=web

DeleteTag uuid=596070a6276746edbf0f54ef721f654e
```

Tags associated with a resource will be deleted automatically when the resource is being deleted.

Resources can be queried by tags using two special query conditions: `__userTag__` and `__systemTag__`:

```
QueryVmInstance __userTag__=web zoneUuid=04b5419ca3134885be90a48e372d3895

QueryHost __systemTag__=capability:liveSnapshot
```

There are also query APIs specific for tags:

```
QueryUserTag resourceUuid=0cd1ef8c9b9e0ba82e0cc9cc17226a26 tag~=web-server-%

QuerySystemTag resourceUuid=50fcc61947f7494db69436ebbbefda34
```

Summary

In this article, we demonstrated ZStack's tag system. With the system, users, plugins, and thirdparty software can use tags in various ways without code and database schema change. This is another basis that enables the potential of rapidly evolving ZStack into a mature, complete cloud solution while keeping the core orchestration robust.

Community

Mailing List (https://groups.google.com/forum/#!forum/zstack)
Community (http://www.zstack.org/community)
Gitter

Resources

Intallation (http://www.zstack.org/installation)
Tutorials (http://www.zstack.org/tutorials)
Blog (http://www.zstack.org/blog)
Documentation (http://www.zstack.org/documentation)

Connect Us

(https://twitter.com/zstack_org) (https://www.facebook.com/zstackorg) (https://github.com/zstackorg/zstack)
(../misc/wechat.html) (http://weibo.com/zstack)

ZStack is open source IaaS software provided under the Apache 2.0 license.

Your feedback is invaluable, please let us know your thoughts. (mailto:info@zstack.org)