Metadata Tagging in Cinder

OpenStack Summit Vancouver

Dave Chen
May 2015
Agenda

- Motivation
- Use-case scenarios
- Work in progress
  - Cinder CLI, API
  - Properties protection
  - New Dashboard
- Conclusion
- Blueprints and references
Motivation

- Reuse the existed block storage
- Cinder volume holds different kinds of metadata
- Image metadata is used in various scenarios
Motivation

- No way to update image metadata in bootable Volume

How to give the semantic meaning to each individual volume

Only support Volume metadata

**CLI Command:**

```
> cinder metadata 7d9db757-401a-4975-a481-33dbc33384ed set spl_creator_policy=1
> cinder metadata 7d9db757-401a-4975-a481-33dbc33384ed unset spl_creator_policy
```
Alternatives

Any other alternatives? besides the dependencies in Cinder's image metadata?
Alternatives

✓ **Nova flavors**
  - **Pros:** Nova needs, so Nova take care of it.
  - **Cons:** Is overridden by image metadata

✓ **control in the image**
  - **Pros:** Glance maintains image, let glance take care of image metadata
  - **Cons:** How to reuse the existed volume
Use-case scenarios

Where is the image metadata consumed?
Use cases - set driver options

- Watchdog

```python
watchdog_action = (flavor.extra_specs.get('hw_watchdog_action')
                  or flavor.extra_specs.get('hw:watchdog_action')
                  or 'disabled')

if (image_meta is not None and
   image_meta.get('properties', {}).get('hw_watchdog_action')):
    watchdog_action = image_meta['properties']['hw_watchdog_action']
```
Nova compute workflow overview with libvirt

```
def run_instance(self, context, instance, request_spec, filter_properties, requested_networks, injected_files, admin_password, is_first_time, node, legacy_bdm_in_spec):
    if filter_properties is None:
        filter_properties ={}
    @utils.synchronized(instance['uuid'])
    def do_run_instance():
        self._run_instance(context, request_spec, filter_properties, requested_networks, injected_files, admin_password, is_first_time, node, instance, legacy_bdm_in_spec)
        do_run_instance()
```

cmd: `run_instance`
kwargs: `{ ... }`
correlation_id: xxxx
context: xxxx

```
manager.run_instance()
manager._run_instance()
manager._build_instance()
manager._spawn()
driver.spawn()
domain = libvirt_conn.defineXML(xml)
domain.createWithFlags()
```

`public method`

`private methods`

`driver public method`
Use cases - Scheduling

- Host filtering

```python
class ImagePropertiesFilter(filters.BaseHostFilter):
    """Filter compute nodes that satisfy instance image properties.

    The ImagePropertiesFilter filters compute nodes that satisfy any architecture, hypervisor type, or virtual machine mode properties specified on the instance's image properties. Image properties are contained in the image dictionary in the request_spec.
    """

    # Image Properties and Compute Capabilities do not change within a request
    run_filter_once_per_request = True

def _instance_supported(self, host_state, image_props, hypervisor_version):
    img_arch = image_props.get('architecture', None)
    img_h_type = image_props.get('hypervisor_type', None)
    img_vm_mode = image_props.get('vm_mode', None)

def host_passes(self, host_state, filter_properties):
    """Check if host passes specified image properties.

    Returns True for compute nodes that satisfy image properties contained in the request_spec.
    """

    spec = filter_properties.get('request_spec', {})
    image_props = spec.get('image', {}).get('properties', {})
```
Use cases

- Shutdown behavior

```python
def _get_power_off_values(self, context, instance, clean_shutdown):
    """Get the timing configuration for powering down this instance."""
    if clean_shutdown:
        timeout = compute_utils.get_value_from_system_metadata(instance,
                     key='image_os_shutdown_timeout', type=int,
                     default=CONF.shutdown_timeout)
        retry_interval = self.SHUTDOWN_RETRY_INTERVAL
    else:
        timeout = 0
        retry_interval = 0

    return timeout, retry_interval
```
WIP - Overall Process

1. Create or update image metadata
2. Invoke CRUD ReST API on metadata
3. Check permission
4. Look up Tom’s role in the project
5. Trigger CRUD operation

Keystone
- ReST API
- Horizon
- Dashboard
- Cinder
- CLI
- Props Protection
- Loading Rules
- Role Based
- Policy Based
- Basic Policies
- Added Policies

Tom
WIP - Cinder Client

$ cinder help image-metadata
usage: cinder image-metadata <volume> <action> <key=value> [<key=value> ...]

Sets or deletes volume image metadata.

Positional arguments:
  <volume>     Name or ID of volume for which to update metadata.
  <action>     The action. Valid values are 'set' or 'unset.'
  <key=value>  Metadata key and value pair to set or unset. For unset, specify only the key.

CLI Command:
> cinder image-metadata 7d9db757-401a-4975-a481-33dbc33384ed set spl_creator_policy=1
> cinder image-metadata 7d9db757-401a-4975-a481-33dbc33384ed unset spl_creator_policy
WIP - Cinder API

**Request**

POST /v2/{project_id}/volumes/{volume_id}/action

Context-Type: application/json
Accept: application/json
X-Auth-Token: token_id

```
{
  "os-set_image_metadata": {
    "metadata": {
      "os_shutdown_timeout": 80
    }
  }
}
```

**Response**

HTTP/1.1 200 OK
x-compute-request-id: ***
x-openstack-request-id: ***
Date: Fri, 10 Apr 2015 02:56:53 GMT
Context-Type: application/json

```
{
  "metadata": {
    "os_shutdown_timeout": 80
  }
}
```
WIP - Cinder API

Request

POST /v2/{project_id}/volumes/{volume_id}/action
Context-Type: application/json
Accept: application/json
X-Auth-Token: token_id

{
   "os-del_image_metadata": {
      "key": "os_shutdown_timeout"
   }

Response

HTTP/1.1 200 OK
x-openstack-request-id: ***
Date: Fri, 10 Apr 2015 02:56:53 GMT
content-length: 0
Context-Type: application/json
Why is this needed in Cinder?
- Glance has role or policy-based access control to image properties
- Unauthorized users should not change properties such as license key

How to do it in Cinder?
- Provide a mechanism for protected properties similar to what Glance did
  - Role-based props protection
  - Policy-based props protection
Role-based property protection

[^x_billing_code_.*]

<table>
<thead>
<tr>
<th>Action</th>
<th>Role</th>
<th>DESC</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>admin</td>
<td>Billing cannot create the props</td>
</tr>
<tr>
<td>read</td>
<td>admin, billing</td>
<td>Grant read permission to admin or billing</td>
</tr>
<tr>
<td>update</td>
<td>admin, billing</td>
<td>Grant update permission to admin or billing</td>
</tr>
<tr>
<td>delete</td>
<td>admin</td>
<td>Billing cannot delete the props</td>
</tr>
</tbody>
</table>
How to: Role-based properties protection

**Step 1:** Configure the main configuration file `cinder.conf`
- `property_protection_file = property-protections.conf`
- `property_protection_rule_format = roles`
- Copy `property-protections.conf` to `/etc/cinder/`

**Step 2:** Create a desired role
- Create a user with the role as member or admin

**Step 3:** Run command to update or create metadata
- `cinder image-metadata volume_id set x_billing_code_1 = 12x888`

Only matched role **admin** or **billing** is allowed
### Policy-based property protection

<table>
<thead>
<tr>
<th>Action</th>
<th>Policy</th>
<th>DESC</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>cinder_creator</td>
<td>cinder_creator: role:admin or role:spl_role</td>
</tr>
<tr>
<td>read</td>
<td>cinder_creator</td>
<td>...</td>
</tr>
<tr>
<td>update</td>
<td>context_is_admin</td>
<td>context_is_admin: role:admin</td>
</tr>
<tr>
<td>delete</td>
<td>context_is_admin</td>
<td>...</td>
</tr>
</tbody>
</table>
How to: Policy-based properties protection

**Step 1**: Configure main configuration file `cinder.conf`
- `property_protection_file = property-protections-policies.conf`
- `property_protection_rule_format = policies`
- Copy `property-protections-policies.conf` to `/etc/cinder/`

**Step 2**: Create a policy
- Create user `tom` with the role as `member` or `admin`
- Create a new policy "cinder_creator"::"role:admin" or :"role:member"

**Step 3**: Run command to update or create metadata
- `cinder image-metadata volume_id set image_os_shutdown = 80`

`admin` or `tom` is allowed
## WIP - New Dashboard

### Volumes

<table>
<thead>
<tr>
<th>Project</th>
<th>Host</th>
<th>Name</th>
<th>Size</th>
<th>Status</th>
<th>Type</th>
<th>Attached To</th>
<th>Bootable</th>
<th>Encrypted</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>openstack-dev@lvmdriver-1#lvmdriver-1</td>
<td>a24f0974-99fc-4e5-95cb-0e18f981e19</td>
<td>2GB</td>
<td>In-use</td>
<td>lvmdriver-1</td>
<td>Attached to os_shutdown_timeout on /dev/vda</td>
<td>Yes</td>
<td>No</td>
<td>Update Volume Status</td>
</tr>
<tr>
<td>admin</td>
<td>openstack-dev@lvmdriver-1#lvmdriver-1</td>
<td>non-bootable-volume</td>
<td>2GB</td>
<td>Available</td>
<td>lvmdriver-1</td>
<td></td>
<td>No</td>
<td>No</td>
<td>Update Metadata</td>
</tr>
<tr>
<td>admin</td>
<td>openstack-dev@lvmdriver-1#lvmdriver-1</td>
<td>volume_from_image</td>
<td>2GB</td>
<td>Available</td>
<td>lvmdriver-1</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Delete Volume</td>
</tr>
</tbody>
</table>
WIP - New Dashboard

Update Metadata

You can specify resource metadata by moving items from the left column to the right column. In the left columns there are metadata definitions from the Glance Metadata Catalog. Use the "Other" option to add metadata with the key of your choice.

Available Metadata

- Custom
- hw_watchdog_action

No available metadata

Existing Metadata

- attached_mode
- rw

- os_shutdown_timeout
- 80

- readonly
- False

Filter

Filter
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk Allocation per Host</td>
<td>Properties related to the Nova scheduler filter AggregateDiskFilter. Filters aggregate hosts based on the available disk space compared to the requested disk space. Hosts in the aggregate with not ...</td>
</tr>
<tr>
<td>IO Ops per Host</td>
<td>Properties related to the Nova scheduler filter AggregateIoOpsFilter. Filters aggregate hosts based on the number of instances currently changing state. Hosts in the aggregate with too many instances will be...</td>
</tr>
<tr>
<td>Instances per Host</td>
<td>Properties related to the Nova scheduler filter AggregateNumInstancesFilter. Filters aggregate hosts by the number of running instances on it. Hosts in the aggregate with too many instances will be...</td>
</tr>
<tr>
<td>Shutdown Behavior</td>
<td>These properties allow modifying the shutdown behavior for stop, rescue, resize, and shelve operations.</td>
</tr>
<tr>
<td>Compute Host Capabilities</td>
<td>Hardware capabilities provided by the compute host. This provides the ability to fine tune the hardware specification required when an instance is requested. The ComputeCapabilitiesFilter should be...</td>
</tr>
<tr>
<td>Hypervisor Selection</td>
<td>OpenStack Compute supports many hypervisors, although most installations use only one hypervisor. For installations with multiple supported hypervisors, you can schedule different hypervisors using...</td>
</tr>
</tbody>
</table>
Conclusion - One concrete example

1. Set `os_shutdown_time = 80`
2. Call CRUD ReST API on metadata
3. Boot instance from volume
   - Copy from `volume_glance_metadata` to `instance_system_metadata`
4. Shutdown instance
5. Call Nova API

Give me your preferences

Wait!
References


2. Property protections in Glance:
   http://docs.openstack.org/developer/glance/property-protections.html
Q & A
Volume metadata (User metadata):
- Controlled by tenant and used by tenant to organize and label their volumes
- Arbitrary key - value pairs
- This should not be given semantic meaning by drivers.

Boot metadata (Image metadata):
- Passed to Nova during boot; serves the same purpose as Glance metadata in controlling the boot
- H/W parameters
- License keys, etc.

Admin metadata:
- Attached to each volume
- Contains some driver specific stuff
- Hidden from the tenant (might be exposed to admin)
- There to avoid growing the volume table indefinitely
- Semantic meaning, possibly just for a subset of drivers
Use cases

○ Virtual CPU topology

✓ Provides preferred socket/core/thread counts for the virtual CPU instance exposed to guests.

✓ Enables avoidance of limitation thresholds on vCPU topologies from OS vendors.
Use cases

- Minimum flavor Requirements
  - Minimum required CPU
  - Minimum RAM
- May be used by the Horizon UI to guide flavor selection.
- A bootable volume can be used to install new software and experiment with setting the minimum requirements before cloning to an image.

- Libvirt driver options
  - The libvirt compute driver options are properties specific to compute drivers.
  - Example: hw_vif_model: Specifies the model of virtual network interface device to use.
Disclaimer

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Include as footnote in appropriate slide where performance data is shown:

- § Configurations: [describe config + what test used + who did testing]
- § For more information go to http://www.intel.com/performance.

Intel, the Intel logo, {List the Intel trademarks in your document} are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2015 Intel Corporation.