目 录

1 snapshot 的流程

1.1 数据结构

1.1.1 req

```
# nova/api/openstack/wsgi.py Request
```

1.1.2 body

```
{u'createImage': {u'name': u'snap1', u'metadata': {}}}
```

1.2 nova-api 部分

```
# nova/api/openstack/compute/servers.py Controller._action_create_image()
def _action_create_image(self, req, id, body):
    # id是instance id
    """Snapshot a server instance."""
    context = req.environ['nova.context']
    entity = body.get("createImage", {})

    image_name = entity.get("name")

    ...

    props = {}
    metadata = entity.get('metadata', {})
    common.check_img_metadata_properties_quota(context, metadata)
    try:
        props.update(metadata)
    except ValueError:
        msg = _("Invalid metadata")
        raise exc.HTTPBadRequest(explanation=msg)

    instance = self._get_server(context, req, id)

    bdms = objects.BlockDeviceMappingList.get_by_instance_uuid(
                context, instance.uuid)

    try:
        if self.compute_api.is_volume_backed_instance(context, instance,
                                                      bdms):
            img = instance['image_ref']
            if not img:
                properties = bdms.root_metadata(
                        context, self.compute_api.image_api,
                        self.compute_api.volume_api)
```

```
                    image_meta = {'properties': properties}
                else:
                    image_meta = self.compute_api.image_api.get(context, img)

                image = self.compute_api.snapshot_volume_backed(
                                                       context,
                                                       instance,
                                                       image_meta,
                                                       image_name,
                                                       extra_properties=props)
            else:
                image = self.compute_api.snapshot(context,
                                                  instance,
                                                  image_name,
                                                  extra_properties=props)
        except exception.InstanceInvalidState as state_error:
            common.raise_http_conflict_for_instance_invalid_state(state_error,
                    'createImage')
        except exception.Invalid as err:
            raise exc.HTTPBadRequest(explanation=err.format_message())

        # build location of newly-created image entity
        image_id = str(image['id'])
        url_prefix = self._view_builder._update_glance_link_prefix(
                req.application_url)
        image_ref = os.path.join(url_prefix,
                                 context.project_id,
                                 'images',
                                 image_id)

        resp = webob.Response(status_int=202)
        resp.headers['Location'] = image_ref
        return resp
```

```
    # nova/compute/api.py API.snapshot()
    def snapshot(self, context, instance, name, extra_properties=None):
        """Snapshot the given instance.

        :param instance: nova.db.sqlalchemy.models.Instance
        :param name: name of the snapshot
        :param extra_properties: dict of extra image properties to include
                                 when creating the image.
        :returns: A dict containing image metadata
        """
        image_meta = self._create_image(context, instance, name,
                                        'snapshot',
                                        extra_properties=extra_properties)

        # NOTE(comstud): Any changes to this method should also be made
        # to the snapshot_instance() method in nova/cells/messaging.py
        instance.task_state = task_states.IMAGE_SNAPSHOT_PENDING
        instance.save(expected_task_state=[None])

        self.compute_rpcapi.snapshot_instance(context, instance,
                                              image_meta['id'])
```

```
22
23          return image_meta
```

```
1   # nova/compute/rpcapi.py ComputeAPI.snapshot_instance()
2   def snapshot_instance(self, ctxt, instance, image_id):
3       version = '3.0'
4       cctxt = self.client.prepare(server=_compute_host(None, instance),
5               version=version)
6       cctxt.cast(ctxt, 'snapshot_instance',
7                  instance=instance,
8                  image_id=image_id)
```

1.3 nova-compute 部分

```
1   # nova/compute/manager.py ComputeManager._snapshot_instance()
2   def snapshot_instance(self, context, image_id, instance):
3       """Snapshot an instance on this host.
4
5       :param context: security context
6       :param instance: a nova.objects.instance.Instance object
7       :param image_id: glance.db.sqlalchemy.models.Image.Id
8       """
9       # NOTE(dave-mcnally) the task state will already be set by the api
10      # but if the compute manager has crashed/been restarted prior to the
11      # request getting here the task state may have been cleared so we set
12      # it again and things continue normally
13      try:
14          instance.task_state = task_states.IMAGE_SNAPSHOT
15          instance.save(
16                      expected_task_state=task_states.IMAGE_SNAPSHOT_PENDING)
17      except exception.InstanceNotFound:
18          # possibility instance no longer exists, no point in continuing
19          LOG.debug("Instance not found, could not set state %s "
20                    "for instance.",
21                    task_states.IMAGE_SNAPSHOT, instance=instance)
22          return
23
24      except exception.UnexpectedDeletingTaskStateError:
25          LOG.debug("Instance being deleted, snapshot cannot continue",
26                    instance=instance)
27          return
28
29      self._snapshot_instance(context, image_id, instance,
30                              task_states.IMAGE_SNAPSHOT)
```

```
1   def _snapshot_instance(self, context, image_id, instance,
2                          expected_task_state):
3       context = context.elevated()
4
5       current_power_state = self._get_power_state(context, instance)
6       try:
```

```
7            instance.power_state = current_power_state
8            instance.save()
9
10           LOG.audit(_('instance snapshotting'), context=context,
11                 instance=instance)
12
13           if instance.power_state != power_state.RUNNING:
14               state = instance.power_state
15               running = power_state.RUNNING
16               LOG.warn(_('trying to snapshot a non-running instance: '
17                     '(state: %(state)s expected: %(running)s)'),
18                   {'state': state, 'running': running},
19                   instance=instance)
20
21           self._notify_about_instance_usage(
22               context, instance, "snapshot.start")
23
24           def update_task_state(task_state,
25                                 expected_state=expected_task_state):
26               instance.task_state = task_state
27               instance.save(expected_task_state=expected_state)
28
29           self.driver.snapshot(context, instance, image_id,
30                                update_task_state)
31
32           instance.task_state = None
33           instance.save(expected_task_state=task_states.IMAGE_UPLOADING)
34
35           self._notify_about_instance_usage(context, instance,
36                                             "snapshot.end")
37       except (exception.InstanceNotFound,
38               exception.UnexpectedDeletingTaskStateError):
39           # the instance got deleted during the snapshot
40           # Quickly bail out of here
41           msg = 'Instance disappeared during snapshot'
42           LOG.debug(msg, instance=instance)
43           try:
44               image_service = glance.get_default_image_service()
45               image = image_service.show(context, image_id)
46               if image['status'] != 'active':
47                   image_service.delete(context, image_id)
48           except Exception:
49               LOG.warning(_("Error while trying to clean up image %s"),
50                           image_id, instance=instance)
51       except exception.ImageNotFound:
52           instance.task_state = None
53           instance.save()
54           msg = _("Image not found during snapshot")
55           LOG.warn(msg, instance=instance)
```

```
1    def snapshot(self, context, instance, image_id, update_task_state):
2        """Create snapshot from a running VM instance.
3
4        This command only works with qemu 0.14+
5        """
```

```
6           try:
7               virt_dom = self._lookup_by_name(instance['name'])
8           except exception.InstanceNotFound:
9               raise exception.InstanceNotRunning(instance_id=instance['uuid'])
10
11          base_image_ref = instance['image_ref']
12
13          base = compute_utils.get_image_metadata(
14              context, self._image_api, base_image_ref, instance)
15
16          snapshot = self._image_api.get(context, image_id)
17
18          disk_path = libvirt_utils.find_disk(virt_dom)
19          source_format = libvirt_utils.get_disk_type(disk_path)
20
21          image_format = CONF.libvirt.snapshot_image_format or source_format
22
23          # NOTE(bfilippov): save lvm and rbd as raw
24          if image_format == 'lvm' or image_format == 'rbd':
25              image_format = 'raw'
26
27          metadata = self._create_snapshot_metadata(base,
28                                                    instance,
29                                                    image_format,
30                                                    snapshot['name'])
31
32          snapshot_name = uuid.uuid4().hex
33
34          state = LIBVIRT_POWER_STATE[virt_dom.info()[0]]
35
36          # NOTE(rmk): Live snapshots require QEMU 1.3 and Libvirt 1.0.0.
37          #            These restrictions can be relaxed as other configurations
38          #            can be validated.
39          # NOTE(dgenin): Instances with LVM encrypted ephemeral storage require
40          #               cold snapshots. Currently, checking for encryption is
41          #               redundant because LVM supports only cold snapshots.
42          #               It is necessary in case this situation changes in the
43          #               future.
44          if (self._has_min_version(MIN_LIBVIRT_LIVESNAPSHOT_VERSION,
45                                    MIN_QEMU_LIVESNAPSHOT_VERSION,
46                                    REQ_HYPERVISOR_LIVESNAPSHOT)
47               and source_format not in ('lvm', 'rbd')
48               and not CONF.ephemeral_storage_encryption.enabled):
49              live_snapshot = True
50              # Abort is an idempotent operation, so make sure any block
51              # jobs which may have failed are ended. This operation also
52              # confirms the running instance, as opposed to the system as a
53              # whole, has a new enough version of the hypervisor (bug 1193146).
54              try:
55                  virt_dom.blockJobAbort(disk_path, 0)
56              except libvirt.libvirtError as ex:
57                  error_code = ex.get_error_code()
58                  if error_code == libvirt.VIR_ERR_CONFIG_UNSUPPORTED:
59                      live_snapshot = False
60                  else:
61                      pass
```

```
62          else:
63              live_snapshot = False
64
65          # NOTE(rmk): We cannot perform live snapshots when a managedSave
66          #            file is present, so we will use the cold/legacy method
67          #            for instances which are shutdown.
68          if state == power_state.SHUTDOWN:
69              live_snapshot = False
70
71          # NOTE(dkang): managedSave does not work for LXC
72          if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
73              if state == power_state.RUNNING or state == power_state.PAUSED:
74                  self._detach_pci_devices(virt_dom,
75                      pci_manager.get_instance_pci_devs(instance))
76                  self._detach_sriov_ports(context, instance, virt_dom)
77                  virt_dom.managedSave(0)
78
79          snapshot_backend = self.image_backend.snapshot(instance,
80                  disk_path,
81                  image_type=source_format)
82
83          if live_snapshot:
84              LOG.info(_LI("Beginning live snapshot process"),
85                       instance=instance)
86          else:
87              LOG.info(_LI("Beginning cold snapshot process"),
88                       instance=instance)
89
90          update_task_state(task_state=task_states.IMAGE_PENDING_UPLOAD)
91          snapshot_directory = CONF.libvirt.snapshots_directory
92          fileutils.ensure_tree(snapshot_directory)
93          with utils.tempdir(dir=snapshot_directory) as tmpdir:
94              try:
95                  out_path = os.path.join(tmpdir, snapshot_name)
96                  if live_snapshot:
97                      # NOTE(xqueralt): libvirt needs o+x in the temp directory
98                      os.chmod(tmpdir, 0o701)
99                      self._live_snapshot(virt_dom, disk_path, out_path,
100                                          image_format)
101                 else:
102                     snapshot_backend.snapshot_extract(out_path, image_format)
103             finally:
104                 new_dom = None
105                 # NOTE(dkang): because previous managedSave is not called
106                 #              for LXC, _create_domain must not be called.
107                 if CONF.libvirt.virt_type != 'lxc' and not live_snapshot:
108                     if state == power_state.RUNNING:
109                         new_dom = self._create_domain(domain=virt_dom)
110                     elif state == power_state.PAUSED:
111                         new_dom = self._create_domain(domain=virt_dom,
112                                 launch_flags=libvirt.VIR_DOMAIN_START_PAUSED)
113                     if new_dom is not None:
114                         self._attach_pci_devices(new_dom,
115                             pci_manager.get_instance_pci_devs(instance))
116                         self._attach_sriov_ports(context, instance, new_dom)
117             LOG.info(_LI("Snapshot extracted, beginning image upload"),
```

```
118                         instance=instance)
119
120             # Upload that image to the image service
121
122             update_task_state(task_state=task_states.IMAGE_UPLOADING,
123                     expected_state=task_states.IMAGE_PENDING_UPLOAD)
124             with libvirt_utils.file_open(out_path) as image_file:
125                 self._image_api.update(context,
126                                        image_id,
127                                        metadata,
128                                        image_file)
129                 LOG.info(_LI("Snapshot image upload complete"),
130                          instance=instance)
```