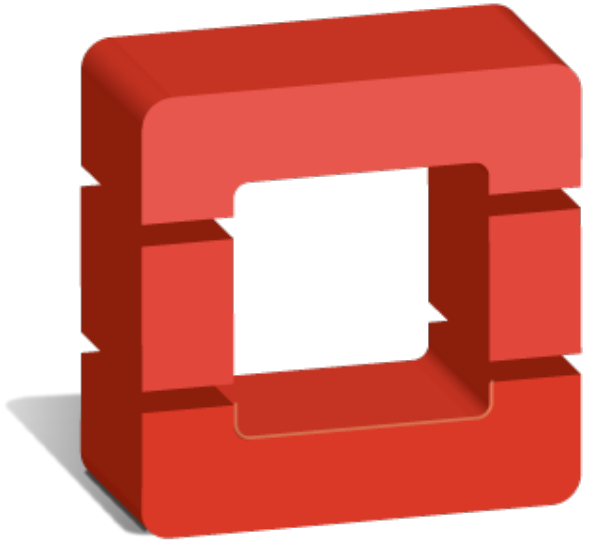


Solving problems, the cloud way!

Peeyush Gupta

What is it?

- OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface.(Got it!)

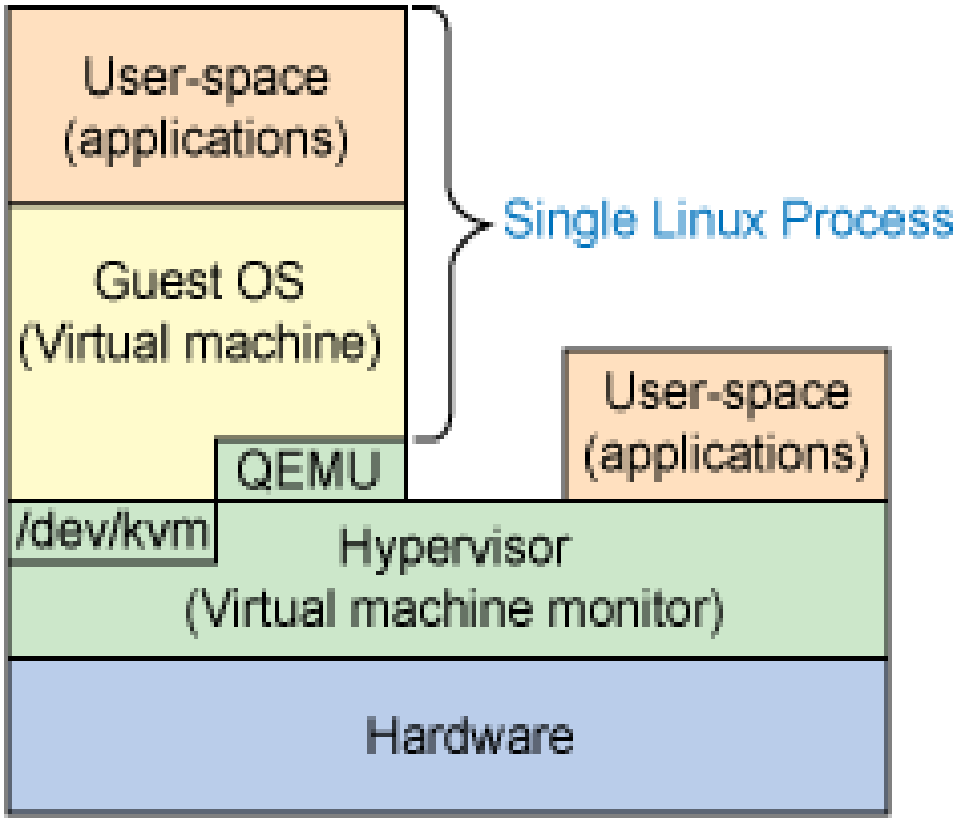
- Provides an Infrastructure as a Service (IaaS)
  - Elasticity
  - Virtualization

Virtualization

- Why?
  - Efficiency

- How?
  - -Hypervisor
  - KVM/QEMU/libvirt

Virtualization



7 Components

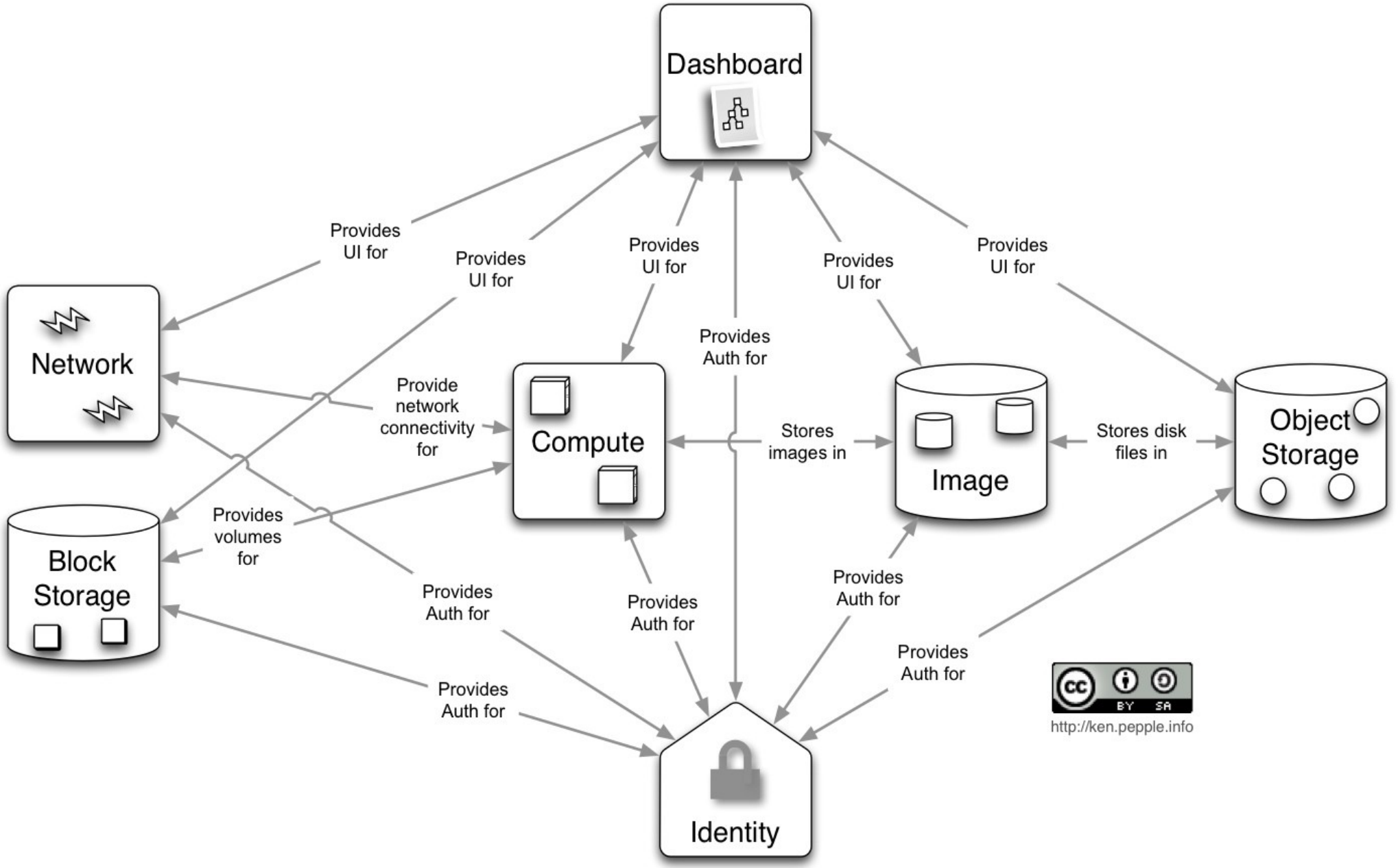
- Horizon – the Dashboard
- Keystone – the Authentication system
- Swift – the Object storage
- Glance – the Image repository
- Nova – the Core of the system
- Quantum – the Networking component
- Cinder – the Volume guy!

Communication

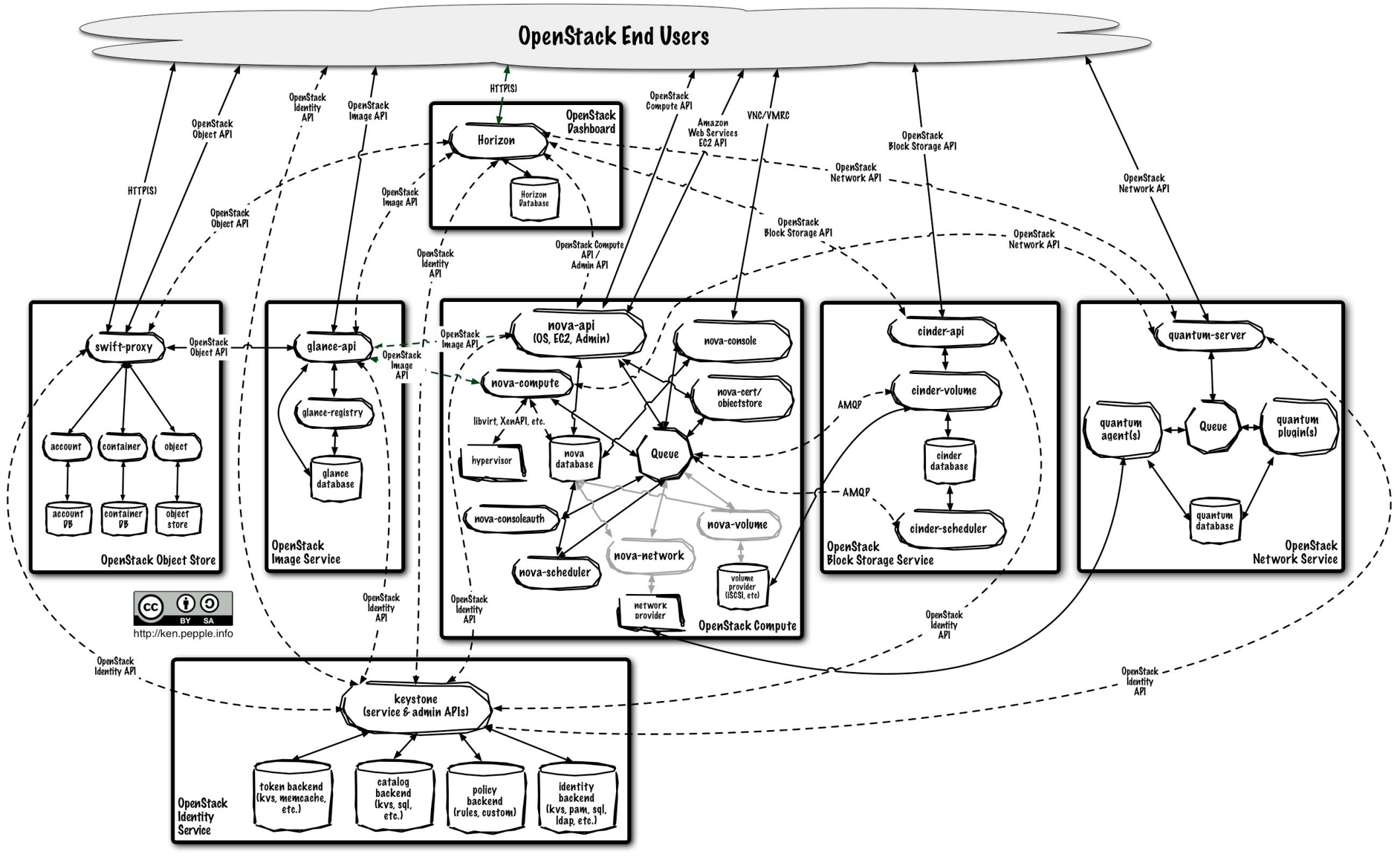
- Database

- Message Queue
  - uses AMQP
  - Implemented through rabbitmq server

Conceptual Architecture



Logical Architecture



Nova

- The big daddy!
- Responsible for storing, retrieving images from Glance and running the instances.
- Provides virtual instances on demand.

Nova – the Components

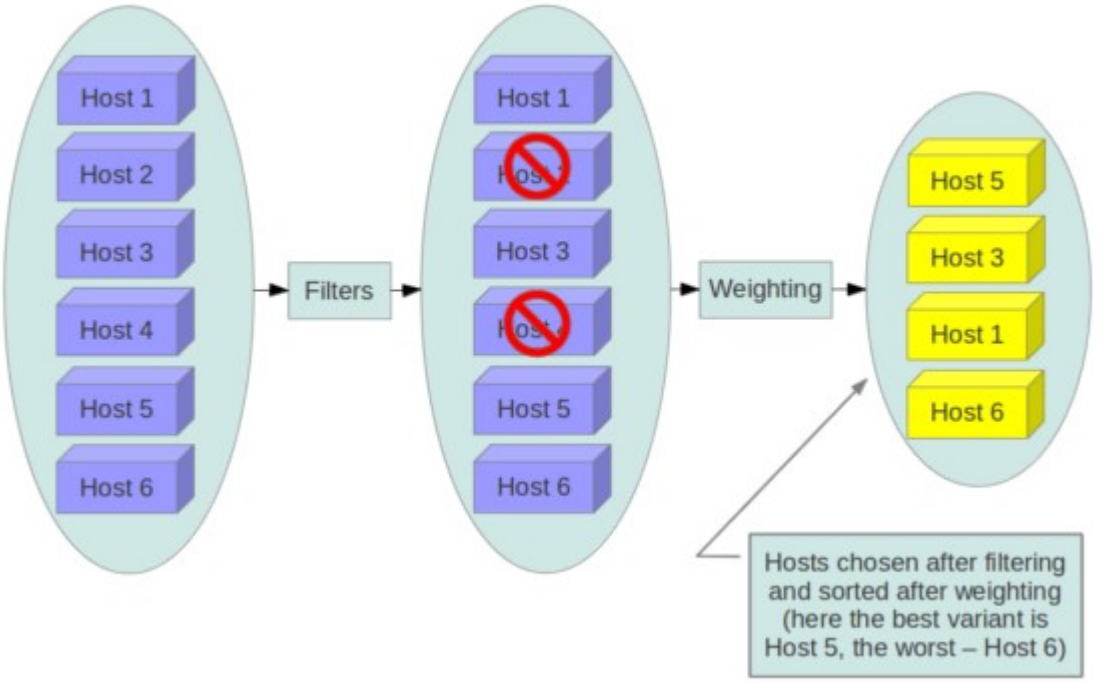
- Nova-api : Talks to the user
- Nova-compute : creates/terminates VM instances.
- Nova-volume : Volume guy, again!
- Nova-network : Performs networking tasks.
- Nova-scheduler : Determines on which host a particular instance should run.
- Console Services

Nova – the Components

- Nova-network
  - Floating and Fixed IPs
  - Supports three kind of networking
    - Flat
    - Flat DHCP
    - VLAN
- Nova-volume
  - Used to add more storage to the running instances.

Nova-scheduler

- Takes a VM instance and determine where it should run (on which host).
- 2 step process



Filtering

- 3 choices
  - Simple : Finds the least loaded host.
  - Random : Selects a random available host.
  - Zone : Selects a random host within an availability zone.
- Defaults
  - Volume Scheduler – Chance (Random)
  - Compute Scheduler – Filter (Availability Zone, RAM and capability)
- Need more?
  - Try hints!

RAM-Filter


```
def host_passes(self, host_state, filter_properties):
    """Only return hosts with sufficient available RAM."""
    pdb.set_trace();
    instance_type = filter_properties.get('instance_type')
    requested_ram = instance_type['memory_mb']
    free_ram_mb = host_state.free_ram_mb
    total_usable_ram_mb = host_state.total_usable_ram_mb

    memory_mb_limit = total_usable_ram_mb * CONF.ram_allocation_ratio
    used_ram_mb = total_usable_ram_mb - free_ram_mb
    usable_ram = memory_mb_limit - used_ram_mb
    if not usable_ram >= requested_ram:
        LOG.debug(_("%(host_state)s does not have %(requested_ram)s MB "
                    "usable ram, it only has %(usable_ram)s MB usable ram."),
                  {'host_state': host_state,
                   'requested_ram': requested_ram,
                   'usable_ram': usable_ram})
        return False
```

Availability Zone-Filter


```
def host_passes(self, host_state, filter_properties):
    spec = filter_properties.get('request_spec', {})
    props = spec.get('instance_properties', {})
    availability_zone = props.get('availability_zone')

    if availability_zone:
        context = filter_properties['context'].elevated()
        metadata = db.aggregate_metadata_get_by_host(
                    context, host_state.host, key='availability_zone')
        if 'availability_zone' in metadata:
            return availability_zone in metadata['availability_zone']
        else:
            return availability_zone == CONF.default_availability_zone

    return True
```

Compute-Filter


```
def host_passes(self, host_state, filter_properties):
    """Returns True for only active compute nodes."""
    capabilities = host_state.capabilities
    service = host_state.service

    alive = self.servicegroup_api.service_is_up(service)
    if not alive or service['disabled']:
        LOG.debug(_("%(host_state)s is disabled or has not been "
                    "heard from in a while"), {'host_state': host_state})
        return False
    if not capabilities.get("enabled", True):
        LOG.debug(_("%(host_state)s is disabled via capabilities"),
                  {'host_state': host_state})
        return False
    return True
```

Other Options

- Core Filter : Filters on the basis of available CPU cores.
- Isolated Hosts Filter : Allows to generate isolated sets of images and hosts.
- Disk Filter : Schedule instances if there are sufficient disk available for ephemeral storage.
- And so on...

Host Aggregates

- Partition the availability zone further.
- Only visible to admin, not to users.
- Each node can have multiple aggregates, each aggregate can have multiple key-value pairs, and the same key-value pair can be assigned to multiple aggregate.
- Uses aggregate_instance_extra_specs filter.

Weighing

- Scheduler applies cost functions on each host and calculates the weight.
- The least weighted host wins.

```
ram_weight_opts = [
        cfg.FloatOpt('ram_weight_multiplier',
                     default=1.0,
                     help='Multiplier used for weighing ram.  Negative '
                          'numbers mean to stack vs spread.'),
]
```

The Actors Behind The Scene

- Manager(gets the rpc call from API) calls up the driver to run the instance:

```
def run_instance(self, context, request_spec, admin_password,
        injected_files, requested_networks, is_first_time,
        filter_properties):
    """Tries to call schedule_run_instance on the driver.
    Sets instance vm_state to ERROR on exceptions
    """
    instance_uuids = request_spec['instance_uuids']
    with compute_utils.EventReporter(context, conductor_api.LocalAPI(),
                                     'schedule', *instance_uuids):
        try:
            return self.driver.schedule_run_instance(context,
                    request_spec, admin_password, injected_files,
                    requested_networks, is_first_time, filter_properties)

        except exception.NoValidHost as ex:
            # don't re-raise
            self._set_vm_state_and_notify('run_instance',
                                          {'vm_state': vm_states.ERROR,
                                          'task_state': None},
                                          context, ex, request_spec)
```

The Actors Behind The Scene

- The filter scheduler (default), calls up the its _scheduler function

```
def schedule_run_instance(self, context, request_spec,
                          admin_password, injected_files,
                          requested_networks, is_first_time,
                          filter_properties):
    """This method is called from nova.compute.api to provision
    an instance.  We first create a build plan (a list of WeightedHosts)
    and then provision.

    Returns a list of the instances created.
    """
    payload = dict(request_spec=request_spec)
    notifier.notify(context, notifier.publisher_id("scheduler"),
                    'scheduler.run_instance.start', notifier.INFO, payload)

    instance_uuids = request_spec.get('instance_uuids')
    LOG.info(_("Attempting to build %(num_instances)d instance(s) "
                "uuids: %(instance_uuids)s"),
              {'num_instances': len(instance_uuids),
               'instance_uuids': instance_uuids})
    LOG.debug(_("Request Spec: %s") % request_spec)

    weighed_hosts = self._schedule(context, request_spec,
                                   filter_properties, instance_uuids)
```

The Actors Behind The Scene

- _schedule function calls up get-filtered-host function and weight function of host manager

```
hosts = self.host_manager.get_all_host_states(elevated)

selected_hosts = []
if instance_uuids:
    num_instances = len(instance_uuids)
else:
    num_instances = request_spec.get('num_instances', 1)
for num in xrange(num_instances):
    # Filter local hosts based on requirements ...
    hosts = self.host_manager.get_filtered_hosts(hosts,
            filter_properties)
    if not hosts:
        # Can't get any more locally.
        break

    LOG.debug(_("Filtered %(hosts)s"), {'hosts': hosts})

    weighed_hosts = self.host_manager.get_weighed_hosts(hosts,
            filter_properties)
```

The Actors Behind The Scene

- Filtering happens here, finally!
- get_filtered_hosts() function of host_manager.py filters hosts and returns only rthe ones passing all filters.
- get_weighed_host() weigh the filtered hosts, according to the weighs set by weigh handler.

In the end...

- The scheduler manager sends an rpc call to the compute manager of selected hosts.
- The compute manager receives it from the queue and runs its create_instance method and we get a running VM instance.

Thank You