CS 657: Database Project

Openstack Cloud Adminstration Interface

Using Ruby-Rails and Twitter Bootstrap

The purpose of the project is to provide an easy interface for viewing the Openstack Environment. The app uses administrative credentials to protect user and senstive information so that any user can view instances running on the Openstack Cloud.

Openstack

OpenStack Software delivers a massively scalable cloud operating system. The technology consists of a series of interrelated projects that control large pools of processing, storage, and networking resources throughout a datacenter.

For the purposes of this project Openstack functionality has been massively scaled, focusing on interfacing the Cloud app database with the databases used by Openstack: Nova and Glance. Here, our main focus will be on the Nova database and it's relation to the Cloud database.

The app establishes three separate connections to the databases, and defines models as sub-classes from these connections. In order to accomplish this we have overloaded the ActiveRecord::Base class to create and maintain the remote connections. The Nova and Glance databases are located on host with IP 10.0.0.0, while the Cloud database is located on 10.0.0.6. The database connections are managed through Ruby-Rails config files, using md5 authorization for remote connections, and peer authorization on localhost. When new instances are generated, they feed from a shared drive which hosts the code for the app, while running thier own server instances, which use their own cache to serve assets such as images and pre-loaded pages. Each instance connects to the database via the network so that any instance hosting the app can interact with itself and the Openstack environment. Haproxy load balancer is used to manage outside connections and distribute them to the virtual interfaces.

1 Tables

1.1 Cloud Database (postgresql)

```
                          List of relations
 Schema |                  Name                  |   Type   |  Owner
--------+------------------------------------------------+----------+----------
 public | Admin                                  | table    | postgres
```

```
 public | Admin_id_seq                                    | sequence | postgres
 public | Jordan                                         | table    | postgres
 public | Jordan_id_seq                                  | sequence | postgres
 public | cloud_users                                    | table    | postgres
 public | cloud_users_id_seq                             | sequence | postgres
 public | schema_migrations                              | table    | postgres
 public | user_cloud_user_associations                   | table    | postgres
 public | user_cloud_user_associations_id_seq            | sequence | postgres
 public | virtual_interface_cloud_user_associations      | table    | postgres
 public | virtual_interface_cloud_user_associations_id_seq | sequence | postgres
(11 rows)
```

The Cloud User table is indexed by id, which can be referenced as cloud_user_id by other tables. The cloud user's relationship to the nova database is that he/she can be logged in at one of the instances in order to interface with the app. The Cloud user can also be an Openstack user. The app protects the account using an encrypted password (bcrypt one-way encryption). The encrypted password is stored in the, and is not recoverable. An admin Cloud user has access to other user accounts, and can also view Nova data such as key-pairs, security rules and view the Openstack users.

```
class CloudUser < App
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

  attr_accessible :email, :password, :password_confirmation, :remember_me,
  :username, :sign_in_count, :current_sign_in, :last_sign_in,
  :current_sign_in_ip, :last_sign_in_ip, :admin, :created_at, :updated_at

  has_many :virtual_interface_cloud_user_associations
  has_many_elsewhere :virtual_interfaces, :through => :virtual_interface_cloud_user_association
  has_many_elsewhere :fixed_ips, :through => :virtual_interfaces
  has_many_elsewhere :floating_ips, :through => :fixed_ips
  has_many :user_cloud_user_associations

end
```

The Virtual Interface Cloud User Association table and User Cloud User Association provide the links for the cloud user to the Nova database.

```
class VirtualInterfaceCloudUserAssociation < App
  attr_accessible :cloud_username, :created_at, :deleted_at, :shared_ip,
  :updated_at, :virtual_interface_id

  alias_attribute  :current_sign_in_ip, :shared_ip
  alias_attribute :address, :shared_ip

  belongs_to :cloud_user
  belongs_to :virtual_interface

end
```

If a cloud user is modified, then a named table is created for them in order to track their admin status, who changed it, when it changed and the login ip of the modifier. This relational table is dissassociated from the app.

1.2 Nova Database (mysql)

```
+-------------------------------------+
| Tables_in_nova                      |
+-------------------------------------+
| agent_builds                        |
| aggregate_hosts                     |
| aggregate_metadata                  |
| aggregates                          |
| auth_tokens                         |
| block_device_mapping                |
| bw_usage_cache                      |
| cells                               |
| certificates                        |
| compute_nodes                       |
| console_pools                       |
| consoles                            |
| dns_domains                         |
| fixed_ips                           |
| floating_ips                        |
| instance_actions                    |
| instance_faults                     |
| instance_info_caches                |
| instance_metadata                   |
| instance_type_extra_specs           |
| instance_types                      |
| instances                           |
| iscsi_targets                       |
| key_pairs                           |
| migrate_version                     |
| migrations                          |
| networks                            |
| projects                            |
| provider_fw_rules                   |
| quotas                              |
| s3_images                           |
| schema_migrations                   |
| security_group_instance_association |
| security_group_rules                |
| security_groups                     |
| services                            |
| sm_backend_config                   |
| sm_flavors                          |
| sm_volume                           |
| snapshots                           |
| user_project_association            |
| user_project_role_association       |
```

```
| user_role_association               |
| users                               |
| virtual_interfaces                  |
| virtual_storage_arrays              |
| volume_metadata                     |
| volume_type_extra_specs             |
| volume_types                        |
| volumes                             |
+-------------------------------------+
50 rows in set (0.00 sec)

mysql>
```

The Instances table is the largest relation in the Nova database. It requires several relational tables in order to connect it to the Openstack environment. Instances are connected to the network with a fixed IP. Connectivity to this instance is managed through a security group, therefore the relation between the instance and its fixed IP is not direct.
The instance connects to the web through a virtual interface, which allows outside connections through a floating IP. The Virtual Interface table associated with the instance and provides a floating IP from a pool, which is associated with a single fixed IP.
Similarly to these relationships, the Openstack User interfaces with the network through similary associations storing information on roles that a user may have on projects and their instances. A user may be, for example, a sysadmin on a project, with the ability to create new instances, and a netadmin on another, with the ability to ssh remotely into the instances.
To save paper, we will only show models that have been modified enough to be considered unique among the others.

1.3 Glance Database (mysql)

```
+------------------+
| Tables_in_glance |
+------------------+
| image_members    |
| image_properties |
| images           |
| migrate_version  |
+------------------+
4 rows in set (0.00 sec)
```

Glance provides data for the instances volume and type information to determine the nature of the instance (Ubuntu Server, etc..)

2 Keys

All records have primary key 'id'. If another primary key is to be used, say in a query, then it is overloaded in the definition of the model. However, in this case it is not needed as tables that require additional primary keys use a 'unique key' constraint, in this case a uuid.
The ERD has left out many of the relations in order to get a reasonable picture. Foreign keys in the database are handled by appending 'id' to a record if it is to be a foreign key. All the foreign keys are indexed on their id in order to provide a hash for search. Each index hash is stored alongside the table in the database. In the case that 'id' is appended to a record, the app will automatically look for its relational table (what 'id' is appended to). In the case that two tables do not have the same name, the foreign key is aliased, as we can see in the Virtual Interface Cloud User Association definition.
For example, in the Cloud database foreign keys are cloud_user_id and virtual_interface_cloud_user_id.
A printout of the schemas and keys are included at the end of the report.

2.1 Queries

The app uses various methods to query the databases.

1. Instance Status and History In each page, the database is queried and dynamically loads the data to a view. Only active and running instances, fixed and floating ips are shown.
   The app provides history of instances that are no longer running, and tracks changes to the Cloud users admin status.
   A method of creating views is to create static scopes in the model definition, which can then be used in queries. This query is managed easily using scopes defined in the model.

```
@instances = Instance.instances_active
```

```
Instance Load (4.8ms)  SELECT `nova`.`instances`.*
  FROM `nova`.`instances` WHERE (vm_state ='active' OR vm_state =
      'building' AND deleted=0)
```

   Another way to query is to pass the sql query directly to the connection. We use this to find all instances that were created within a certain time period.

```
query ="SELECT display_name, users.name, instances.created_at
    FROM instances LEFT JOIN users ON instances.user_id = users.id
     WHERE instances.created_at > (NOW() - INTERVAL 1 DAY);"
```

```
@hash = Nova.connection.select_all(query)
```

2. User Status and History (Queries 1, 2 and 3 on Cloud User) On accessing the app, the database is queried on each page to make sure the user is logged in, on restricted pages the query will also check if the user is an admin. The implicit query on the cloud user redirects the app to the main dashboard.

```
CloudUser Load (0.4ms)  SELECT "cloud_users".* FROM "cloud_users"
    WHERE "cloud_users"."id" = 2 LIMIT 1
```

Selecting a distinct cloud user account view passes the primary key as a parameter.

```
@cloud_user = CloudUser.find(params[:id])
```

```
CloudUser Load (0.3ms)  SELECT "cloud_users".* FROM "cloud_users"
    WHERE "cloud_users"."id" = $1 LIMIT 1  [["id", "2"]]
```

When querying a table that may not exist, more information is needed from the connection.

```
username = @cloud_user.username
if App.connection.table_exists? username
    @hash = App.connection.select_all("SELECT * FROM \"#{
        username}\" WHERE updated_at  = (SELECT max(updated_at)
        FROM \"#{username}\" );")
else
    redirect_to :back, :notice => "No History for this User."
end
```

```
(0.4ms)  SELECT * FROM "Jordan" WHERE updated_at = (SELECT max(
    updated_at)
FROM "Jordan" );
```

An interesting query is to see if a cloud user is signed into the app on one of the hosts that is hosting the app. This requires an association of two different databases on two different hosts. When associating tables from the two databases on two different hosts, more assocations were needed in the query, the queries are linked together in the associations controller to provide the view. A new reference table is created for this purpose.

```
floating_shared = FloatingIp.shared
CloudUser.all.each do |cshared|
  @shared = VirtualInterfaceCloudUserAssociation.new
  FloatingIp.shared.each do |fshared|
      if cshared.current_sign_in_ip == fshared.address
          shared_interface_id = FixedIp.fixed_active.select('
              virtual_interface_id')
```

```
            .where('id=?',fshared.fixed_ip_id)
            @shared.update_attributes(:cloud_username => cshared.
                username,
            :shared_ip => cshared.current_sign_in_ip, :
                virtual_interface_id =>
            shared_interface_id)
        end
    end
    @shared.save
end
```

```
  FloatingIp Load (0.8ms)  SELECT address, fixed_ip_id FROM `nova
      `.`floating_ips` WHERE (`nova`.`floating_ips`.`fixed_ip_id`
      IS NOT NULL)
(0.3ms)  BEGIN
SQL (0.9ms)  INSERT INTO "
    virtual_interface_cloud_user_associations"
("cloud_username", "created_at", "deleted_at", "shared_ip", "
    updated_at", "virtual_interface_id") VALUES ($1, $2, $3, $4, $5
    , $6) RETURNING "id" [["cloud_username", nil], ["created_at",
    Thu, 16 May 2013 13:02:29 UTC +00:00], ["deleted_at", nil], ["
    shared_ip", nil], ["updated_at", Thu, 16 May 2013 13:02:29 UTC
    +00:00], ["virtual_interface_id", nil]]
(29.1ms)  COMMIT
  FloatingIp Load (1.0ms)  SELECT address, fixed_ip_id FROM `nova
      `.`floating_ips` WHERE (`nova`.`floating_ips`.`fixed_ip_id`
      IS NOT NULL)
(0.3ms)  BEGIN
SQL (0.7ms)  INSERT INTO "
    virtual_interface_cloud_user_associations" ("cloud_username", "
    created_at", "deleted_at", "shared_ip", "updated_at", "
    virtual_interface_id") VALUES ($1, $2, $3, $4, $5, $6)
    RETURNING "id" [["cloud_username", nil], ["created_at", Thu, 16
     May 2013 13:02:29 UTC +00:00], ["deleted_at", nil], ["
    shared_ip", nil], ["updated_at", Thu, 16 May 2013 13:02:29 UTC
    +00:00], ["virtual_interface_id", nil]]
(15.9ms)  COMMIT
```

3. Fixed and Floating IP Status (Queries 4, 5 and 6 on Ip Status) Finding the status of the IP pools, we again used various query methods. To find unassigned Ips, it was easier to use scoping.

```
@floating_ips = FloatingIp.floating_active
```

```
FloatingIp Load (1.5ms)  SELECT `nova`.`floating_ips`.* FROM `
    nova`.`floating_ips` WHERE (`nova`.`floating_ips`.`
    fixed_ip_id` IS NOT NULL)
```

```
@fixed_ips = FixedIp.fixed_active
```

```
FixedIp Load (1.5ms)  SELECT `nova`.`fixed_ips`.* FROM `nova`.`
    fixed_ips` WHERE (allocated=1)
```

To find more specific values, sql queries were used.

```
@hash = Nova.connection.select_all(query)
```

```
(0.7ms)  SELECT address, projects.name FROM floating_ips LEFT
    JOIN projects ON floating_ips.project_id = projects.id WHERE
    auto_assigned = 1;
```

```
@hash = Nova.connection.select_all(query)
```

```
(0.8ms)  SELECT address, instances.hostname FROM fixed_ips LEFT
    JOIN instances ON fixed_ips.instance_id = instances.id WHERE
    instances.deleted_at > (NOW() - INTERVAL 1 DAY);
```

2.2 Updates

The cloud user is allowed to change their username, email and password. Changing any of these values requires password authentication. Since the password is stored encrypted, a forgotten password can be changed using authentication via email.

```
self.cloud_user = cloud_user_class.reset_password_by_token(
    cloud_user_params)
cloud_user.unlock_access! if unlockable?(cloud_user)
sign_in(cloud_user_name, cloud_user)
```

```
Processing by Devise::RegistrationsController#update as HTML
  Parameters: {"utf8"=>"    ", "authenticity_token"=>"1
      pGEt8ogzojB0F6L9620wmH7HrLNXVxZXQLWbMCl0wg=", "cloud_user"=>{"
      email"=>"admin@mail.com", "username"=>"Admin", "password"=>"[
      FILTERED]", "password_confirmation"=>"[FILTERED]", "
      current_password"=>"[FILTERED]"}, "commit"=>"Update"}
  CloudUser Load (0.4ms)  SELECT "cloud_users".* FROM "cloud_users"
     WHERE "cloud_users"."id" = 2 LIMIT 1
  CloudUser Load (0.4ms)  SELECT "cloud_users".* FROM "cloud_users"
     WHERE "cloud_users"."id" = $1 LIMIT 1  [["id", 2]]
   (0.3ms)  BEGIN
   (0.6ms)  UPDATE "cloud_users" SET "encrypted_password" = '
      $2a$10$yctJoP30d0lgsRhR.K/5.ebqFVCZm2atExYnjn/9BCtMBIQOTj1zq', "
      updated_at" = '2013-05-16 13:35:27.262321' WHERE "cloud_users"."
      id" = 2
```

```
(29.3ms)  COMMIT
```

Changing the username is similar, we do not need to unlock the resource, since the method is authenticated by the password.

```
cloud_user.update_with_password(account_update_params)
sign_in cloud_user_id, cloud_user, :bypass => true
```

```
Processing by Devise::RegistrationsController#update as HTML
  Parameters: {"utf8"=>"✓", "authenticity_token"=>"1
    pGEt8ogzojB0F6L9620wmH7HrLNXVxZXQLWbMCl0wg=", "cloud_user"=>{"
    email"=>"admin@mail.com", "username"=>"Admin", "password"=>"[
    FILTERED]", "password_confirmation"=>"[FILTERED]", "
    current_password"=>"[FILTERED]"}, "commit"=>"Update"}
  CloudUser Load (0.6ms)  SELECT "cloud_users".* FROM "cloud_users"
    WHERE "cloud_users"."id" = 2 LIMIT 1
  CloudUser Load (0.4ms)  SELECT "cloud_users".* FROM "cloud_users"
    WHERE "cloud_users"."id" = $1 LIMIT 1  [["id", 2]]
 (0.3ms)  BEGIN
 CloudUser Exists (0.5ms)  SELECT 1 AS one FROM "cloud_users" WHERE ("
    cloud_users"."email" = 'admin@mail.com' AND "cloud_users"."id" !=
    2) LIMIT 1
 (0.4ms)  UPDATE "cloud_users" SET "email" = 'admin@mail.com', "
    updated_at" = '2013-05-16 13:31:25.407878' WHERE "cloud_users"."id"
     = 2
 (16.8ms)  COMMIT
```

An admin cloud user is also allowed to change the admin status of any other cloud user.

In order to assist with data tracking, two new tables were dynamically created in two different styles. The first is a reference table that is created when a cloud user's admin status is updated. The table is given the username.

```
@cloud_user = CloudUser.find(params[:id])
  username = @cloud_user.username
  query = "INSERT INTO \"#{username}\" (updated_at, status, admin,
      admin_ip) VALUES (CURRENT_TIMESTAMP, True, '#{
      current_cloud_user.username}', '#{current_cloud_user.
      current_sign_in_ip}');"
if App.connection.table_exists? username
  App.connection.execute(query)
else
  App.connection.create_table(username) do |t|
    t.column :updated_at, :datetime
    t.column :status, :boolean
    t.column :admin, :string
    t.column :admin_ip, :string
  end
  App.connection.execute(query)
```

```
end
```

The second table is used to populate data in a model view in order to see which cloud users are logged in on the Openstack network. The table is dropped each time a user tries to access the view, and rebuilt, ensuring that the data is always current.

```
if App.connection.table_exists? '
    virtual_interface_cloud_user_associations'
  App.connection.execute('DROP TABLE
      virtual_interface_cloud_user_associations')
end
App.connection.create_table('
    virtual_interface_cloud_user_associations') do |t|
    t.column :created_at, :datetime
    t.column :deleted_at, :datetime
    t.column :updated_at, :datetime
    t.column :shared_ip, :string
    t.column :cloud_username, :string
    t.column :virtual_interface_id, :integer
end
floating_shared = FloatingIp.shared
CloudUser.all.each do |cshared|
  @shared = VirtualInterfaceCloudUserAssociation.new
  FloatingIp.shared.each do |fshared|
    if cshared.current_sign_in_ip == fshared.address
      shared_interface_id = FixedIp.fixed_active.select('
          virtual_interface_id').where('id=?',fshared.fixed_ip_id)
      @shared.update_attributes(:cloud_username => cshared.username,
           :shared_ip => cshared.current_sign_in_ip, :
          virtual_interface_id => shared_interface_id)
    end
  end
  @shared.save
```

3 Future

We considered implementing a query that would compare the user and cloud user, and set up the framework to do so, however, it is currently only possible to compare these two records by username. In order to know if a cloud user is also an openstack admin, we would need for them to authenticate themselves using the secret key that Openstack provides for them. Then their identity could be confirmed using this key and their username. Currently, the secret key for each openstack user is viewable, which would need to change and other considerations for protecting the openstack secret key should be considered.