Directory structure

cloud-scheduler-gui/

- **db/** : store pragma.sql. It can be removed after import to mysql.
- **scripts/** : store all of python scripts.
    - Python scripts can be divided into 3 groups:
        1. scripts which is requested by frontend.
            - **getDashboard.py**

              input : -

              output : amount = number of sites

              sites = list of all sites with all sites' description
            - **getSiteDescription.py**

              input : site_id = id of the site which you want to request its description

              date_req = date and time to request sites' description (yyyy-mm-dd HH:00:00)

              output : sites = required site's description

              running = number of running virtual cluster
            - **checkConnectionType.py** : for checking the connection types for creating single cluster spanning multiple sites reservation.

              input : connection_type = list of all connection types to be checked separated by | (pipe).

              output : result = True if all sites have at least one same connection type, otherwise False.
            - **checkForReservation.py** : to check in creating a reservation function step 1.

              input : session_id = session id of user

              begin = begin date and time for reservation.

end = end date and time for reservation.

sites_id = list of sites' id which user want to reserve separated by , (comma).

Resources = list of all resources' amount. Each site is separated by | (pipe) and each resource type is separated by , (comma). For example, "12,16|4,6" -> there are 2 sites. First site is required 12 of first resource type and 16 of second resource type. Second site is required 4 of first resource type and 6 of second resource type. (now first resource type is CPU and second is memory)

img_type = image type user chose.

output : result = True if reservation can be created, otherwise False.

isResourceError = True if the reservation cannot be created because some resource is not available, False if it occurs from another reason.

site_error = list of all sites (site_id) and index of resources (resource_index) which are not available.

- **createReservation.py**

input : session_id = session id of user

begin = begin date and time for reservation.

end = end date and time for reservation.

sites_id = list of sites' id which user want to reserve separated by , (comma).

Resources = list of all resources' amount. Each site is separated by | (pipe) and each resource type is separated by , (comma). For example, "12,16|4,6" -> there are 2 sites. First site is required 12 of first resource type and 16 of second resource type. Second site is required 4 of first resource type and 6 of second resource type. (now first resource type is CPU and second is memory)

img_type = image type user chose.

title = title of a reservation

description = description of a reservation

type = reservation type : single cluster on single site or single cluster spanning multiple sites.

output : result = True if reservation can be created, otherwise False.

isResourceError = If the result is False, it returns True if the reservation cannot be created because some resource is not available and False if it occurs from another reason.

site_error = list of all sites (site_id) and index of resources (resource_index) which are not available. (if any)

reserve_id = id of this reservation if it can be created.

- **Search.py**

input : resources = list of all resources' amount separated by , (comma).

connection_type = additional network connection type.

image_type = image type.

begin = begin date and time for reservation.

end = end date and time for reservation.

all_period = True if user select to reserve 'from begin to end', otherwise False.

days = if user don't select to reserve 'from begin to end', it requires the number of days.

hours = number of hours.

output : result_type = result if there are some sites matched all criteria, suggest if there are some sites have enough resources but they are not available in that time criteria, none if there is no site match.

amount = number of site results.

sites = list of all site results with each site description.

time : begin and end available date time for reservation.

- **getMyReservations.py**

  input : session_id = session id of user

  output : result = list of this specific user's existing reservations with their description. (reservation_id, title, description, begin, end, owner, image_tpe, type, sites, admin_description)

- **getAllReservations.py**

  input : session_id = session id of user (admin only)

  output : result = list of all users' existing reservations with their descriptions. (reservation_id, title, description, begin, end, owner, image_tpe, type, sites, admin_description)

- **getMyEndedReservations.py**

  input : session_id = session id of user

  output : result = list of all users' past reservations with their descriptions. (reservation_id, title, description, begin, end, owner, image_tpe, type, sites, admin_description)

- **getAllEndedReservations.py**

  input : session_id = session id of user (admin only)

  output : result = list of all users' past reservations with their descriptions. (reservation_id, title, description, begin, end, owner, image_tpe, type, sites, admin_description)

- **extendReservation.py**

  input : session_id = session id of user

  end = new end date and time of reservation

  reservation_id = id of the reservation which is requested to extend.

output : result = True if the extension is complete, otherwise False.

- **cancelReservation.py**

  input : session_id = session id of user

  reservation_id = id of the reservation which is requested to cancel.

  reason = reason of this cancelation

  output : result = True if the cancelation is complete, otherwise False.

- **signIn.py**

  input : username = username of user

  password = password of user

  output : result = True if login successfully, otherwise False.

  session_id = session id of user

  firstname = user's first name

  lastname = user's last name

  email_address = user's email address

  phone_number = user's phone number

  status = role of user (admin/user)

  organition = user's organization

  position = user's position

  language = user's language

  timezone = user's timezone

  public key = user's public key

- **setTimezone.py**

  input : session_id = session id of user

  timezone = new timezone

  output : result = True if change user's time zone successfully, otherwise False.

- **getAllImages.py**

  input : -

  output : image_type = list of images with its description including 'name' and 'description'

- **forgetPassword.py (incomplete)**

  input : username = username of user

  password = new user's password

  output : result = True if prepare to reset password and send email to user successfully, otherwise False.

2. class

- **AuthenticationManager.py** : To login and check session expiration and after login successfully it creates an instance (object) of class User.
- **User.py** : To store, set and get user data such as username, password, session id etc.
- **ReservationManager.py** : To check for reservation, create a reservation, extend reservation, cancel reservation and get user's reservation.
- **Reservation.py** : To store, set and get data about reservation such as title, description, sites etc.
- **SiteManager.py** : To get all or specific sites' description and search sites by criteria.

- **Site.py** : store, set and get data about site such as name, resources, image type, additional network connection type etc.
- **Resource.py** : CPU and Memory are subclasses of Resource. They are used for storing, setting and getting data about the resources in any site such as total amount of resource, available amount of resource etc.
- **Database.py** : For basic operation to access the database such as connect, execute, commit etc. Hostname, username, password and database name are specified in this file.
- **JSONFormatter.py** : To convert data to be JSON format before return to frontend.

3. scripts which is requested by backend (pcc).
   - **getUserData.py**

     input : session_id = session id of admin only
     username = username of user that you want to request a description

     output : result = True if there is this user in the system, otherwise False.
     session_id = session id of user
     firstname = user's first name
     lastname = user's last name
     email_address = user's email address
     phone_number = user's phone number
     status = role of user (admin/user)
     organition = user's organization
     position = user's position
     language = user's language
     timezone = user's timezone

public key = user's public key

- **updateReservationStatus.py**

  input : session_id = session id of admin only

  reservation_id = id of the reservation you want to update status

  site_id = id of site you want to update status in this reservation

  status = status of this virtual cluster

  description = some description of this site's status specified by admin

  output : result = True if the reservation's status and admin's description can be updated (the description cannot be as same as the description of previous version), otherwise False.

  reservation = all information of this updated reservation including reservation_id, title, description, begin, end, owner, image_type, type, sites with their name, their reserved CPU in this reservation, their reserved memory in this reservation, their status, and their description from admin.

How to add more resource type

1. **Table 'site' , 'site_reserved' , 'schedule'** : should add a new column to keep amount of new resources AFTER 'memory' column.
2. **Resource.py** : should create a new resource's subclass extending from the 'Resource' class
   - now there are only 'CPU' and 'Memory' subclass

```
class CPU(Resource,object):

    def __init__(self,siteId=None,total=None,availAmount=None):
        super(CPU,self).setType("CPU")
        super(CPU,self).setTotal(total)
        super(CPU,self).setSiteId(siteId)


    def setAvailableAmount(self,db=None,begin=datetime.now().strftime("%Y-%m-%d %H:00:00"),end=datetime.now().strftime("%Y-%m-%d %H:00:00")):
        self.__availableAmount = super(CPU,self).setAvailableAmount(db=db,begin=begin,end=end)



class Memory(Resource,object):

    def __init__(self,siteId=None,total=None,availAmount=None):
        super(Memory,self).setType("memory")
        super(Memory,self).setTotal(total)
        super(Memory,self).setSiteId(siteId)

    def setAvailableAmount(self,db=None,begin=datetime.now().strftime("%Y-%m-%d %H:00:00"),end=datetime.now().strftime("%Y-%m-%d %H:00:00")):
        self.__availableAmount = super(Memory,self).setAvailableAmount(db=db,begin=begin,end=end)
```

   - If you want to add another resource type such as GPU, you should create subclass name 'GPU' and set type for it.

```
class GPU(Resource,object):

    def __init__(self,siteId=None,total=None,availAmount=None):
        super(GPU,self).setType("GPU")
        super(GPU,self).setTotal(total)
        super(GPU,self).setSiteId(siteId)

    def setAvailableAmount(self,db=None,begin=datetime.now().strftime("%Y-%m-%d %H:00:00"),end=datetime.now().strftime("%Y-%m-%d %H:00:00")):
        self.__availableAmount = super(GPU,self).setAvailableAmount(db=db,begin=begin,end=end)
```

3. **Site.py** : in constructor, you should call method addResource() with an instance of new resource type.

```
def __init__(self, site=None, site_id=None):
    self.__resources = []

    if site_id != None:
        self.__siteId = site_id

    if site != None:
        self.__siteId = site[0]
        self.__name = site[1]
        self.__description = site[2]
        self.__contact = site[3]
        self.__location = site[4]
        self.__pragma_boot_path = site[5]
        self.__pragma_boot_version = site[6]
        self.__python_path = site[7]
        self.__temp_dir = site[8]
        self.__username = site[9]
        self.__deployment_type = site[10]
        self.__site_hostname = site[11]
        self.__latitude = site[12]
        self.__longitude = site[13]

        db = Database()
        if db.connect() :
            db.execute("START TRANSACTION;")
            self.addResource(db,CPU(siteId=self.__siteId, total=site[14]))
            self.addResource(db,Memory(siteId=self.__siteId, total=site[15]))
            self.addResource(db,GPU(siteId=self.__siteId, total=site[16]))
        db.close
```

How to add more site description

- Add more columns in ‘site’ table after all of the resource types’ columns.
- Edit the constructor in Site.py to set new attribute. For example, adding ‘continent’ of each site.

```
class Site:
    __resources = []

    def __init__(self, site=None, site_id=None):
        self.__resources = []

        if site_id != None:
            self.__siteId = site_id

        if site != None:
            self.__siteId = site[0]
            self.__name = site[1]
            self.__description = site[2]
            self.__contact = site[3]
            self.__location = site[4]
            self.__pragma_boot_path = site[5]
            self.__pragma_boot_version = site[6]
            self.__python_path = site[7]
            self.__temp_dir = site[8]
            self.__username = site[9]
            self.__deployment_type = site[10]
            self.__site_hostname = site[11]
            self.__latitude = site[12]
            self.__longitude = site[13]

            db = Database()
            if db.connect() :
                db.execute("START TRANSACTION;")
                self.addResource(db,CPU(siteId=self.__siteId, total=site[14]))
                self.addResource(db,Memory(siteId=self.__siteId, total=site[15]))
            db.close

            self.__continent = site[16]
```

- Add method get for each new attribute.

```
def getContinent(self):
    return self.__continent
```

- Edit method formatSite() in JSONFormatter which gets site's data to return to frontend.

```
def formatSite(self,s):
    jsonStr = '{"id" : "'+str(s.getSiteId())+'",'
    jsonStr += '"name" : "'+str(s.getName())+'",'
    jsonStr += '"description" : "'+str(s.getDescription())+'",'
    jsonStr += '"contact" : "'+str(s.getContact())+'",'
    jsonStr += '"location" : "'+str(s.getLocation())+'",'
    jsonStr += '"pragma_boot_path" : "'+str(s.getPragmaBootPath())+'",'
    jsonStr += '"pragma_boot_version" : "'+str(s.getPragmaBootVersion())+'",'
    jsonStr += '"python_path" : "'+str(s.getPythonPath())+'",'
    jsonStr += '"temp_dir" : "'+str(s.getTempDir())+'",'
    jsonStr += '"username" : "'+str(s.getUsername())+'",'
    jsonStr += '"deployment_type" : "'+str(s.getDeploymentType())+'",'
    jsonStr += '"site_hostname" : "'+str(s.getSiteHostname())+'",'
    jsonStr += '"latitude" : "'+str(s.getLatitude())+'",'
    jsonStr += '"longitude" : "'+str(s.getLongitude())+'",'
    jsonStr += '"continent" : "'+str(s.getContinent())+'",'

    #get site's image type
    jsonStr += '"image_type" : ['
    for img in s.getImageType():
```

How to add description for resources

- Create new tables for resources with column ‘name’ and ‘description’.

How to add description to images

- Now I’ve changed table and source code to support images’ description, so you don’t need to change anything.