CS 6343: CLOUD COMPUTING
Mid Term Project Milestones

Midterm requirements

- Study the PaaS platforms
  - Discuss how they are different in important features
  - Discuss similar functionalities in different PaaSs and how their APIs are different
- RoboCode should have basic functionalities, fully running
  - Allow user to read, create, edit, save, compile, and play the robot programs
  - Improve the GUI and coding of the current web-based RoboCode program
- Install and manage Cloud Foundry (CF)
  - All documents for CF: https://docs.cloudfoundry.org/
- Develop and deploy the basic RoboCode on Cloud Foundry (CF)
  - Run the CF on a cluster of servers
    - OpenStack deployment
      - https://docs.openstack.org/newton/install/ (OpenStack Newton installation abstract)
      - https://docs.openstack.org/newton/install-guide-ubuntu/ (OpenStack Newton installation on Ubuntu)
      - https://docs.cloudfoundry.org/deploying/openstack/ (CF deployment on OpenStack)
      - https://bosh.io/docs/init-openstack.html (CF Bosh installation on OpenStack)
      - https://github.com/cloudfoundry/bosh (Cloud Foundry Bosh download from GitHub)
    - Simple CF deployment on each machine
      - Need a router in the final project (can be an existing router or implement your own)
    - Simple CF deployment on one machine
      - If all else fails
  - Deploy services on CF instances
    - Pick a few Java programs you have written (can be one duplicated into multiple instances)
    - Better to have some Java programs that do perform some significant computation
    - Deploy each on the CF cluster
  - Build RoboCode as a few microservices and deploy them on each CF
  - Include one authentication mechanism that CF supports
  - Include one shared DB for all CF instances (distributed DB preferred, one that CF supports)
  - Include a router package to route the http requests securely to the RoboCode instance in each CF
- Multiple users and security
  - Show that you can create users on CF and assign different privileges to them
    - Some users may not be able to access some services
      - If you only deploy your RoboCode as one service, then you can deploy other services to show your service level access control
  - Store the data at the local database or in a distributed DB that CF supports
    - Each user has a list of robots
    - Each robot has its source code and its executable (will need to add other properties later, such as playing results, ranking, score, etc.)
    - Each user should be able to access her/his own data (robots, etc.) and other users should not be allowed to access them
  - User management in cloud foundry
    - CF has developed its own "user account and authentication (UAA) service
      - Instructions about UAA deployment: https://docs.cloudfoundry.org/concepts/architecture/uaa.html

- Use UAA to create an admin account; from the admin account, use CF CLI to create users; then use CF CLI to assign roles to users
  - https://docs.cloudfoundry.org/concepts/architecture/uaa.html
  - https://docs.cloudfoundry.org/uaa/uaa-user-management.html
  - https://docs.cloudfoundry.org/adminguide/cli-user-management.html
  - Check https://docs.cloudfoundry.org/cf-cli/cf-help.html for specific role assignment commands
  - You can use UAAC to create user accounts, but cannot associate CF roles to them, because UAA does not know the predefined roles in CF and its access control is at a lower level; Thus, configuring user access rights via UAAC would be more complex
  - If you create users via CF CLI, users are created not only in UAA but also in Cloud Controller databases, CF CC database stores the role assignments
- To make life easier, you can also explore some additional tools on top of CF for GUI based user management
  - https://github.com/MonsantoCo/cf-users
  - https://github.com/cloudfoundry-incubator/admin-ui

- Overall system
  - Your RoboCode on CF should be a complete solution
  - From the user login and management service to RoboCode functionalities to the DB service with some basic security features (as stated above)
- Preliminary design of the overall system you are going to implement as the final project
  - Does not have to be a complete design, but should have been well thought and planned
  - Will be discussed after midterm demo
  - RoboCode
    - Additional functionalities in RoboCode, such as scoring, etc
      - Cloud be a separate gaming microservice that is responsible for general scoring and ranking, etc.
      - Cloud consider other potential functionalities
    - Improved GUI
    - Improved design
  - Design of the multi-tenant access control system
    - The overall admin should be able to create new tenants or delete tenants
    - Each tenant admin should be able to create and delete users
    - The admin of a tenant should be able to
      - Define a role hierarchy and the access rights for each role
      - Assign users in the domain to roles
      - Define access rights
        - For example, the access rights to the robots should include read/update/play
    - Each user may access the services and data according to their rights
      - All the data created by the users in the domain belong to the domain
      - E.g., the robots created by the users in the domain belongs to the domain
    - The system should support cross tenant accesses
      - Each tenant should also have role mapping tables for other tenants
      - Roles from other tenants (not necessarily all tenants) are mapped to local roles for accesses to local data
      - It is possible that a domain has only a single user but multiple roles
    - To make the access right assignment on a large number of resources easier, your system should also support the grouping of resources (data and services, especially data)
      - Better to consider resource hierarchy

        - Access rights can be assigned to resource groups
    - How to integrate with CF
        - It is above CF
        - Consider the APIs needed by the upper level applications (like your RoboCode)
        - Consider the CF capabilities to be used to achieve the goal
        - Consider the features that are not offered in the CF and need your implementation
- Design of the router package
    - It is above CF
    - The routing should be secure
    - You may consider load balancing, but optional
- Experimentation procedure
    - Goal of your experiments (what you want to measure or observe)
    - Additional code needed for the experimentation
    - How to collect the experimental results
- Planned work distribution
    - Very high level, based on components in the system

Final requirements

- Please refer to the original project description and midterm design description
- More information will be given if additional requirements are needed

Midterm submissions

- Please refer to the original project description for information