Final project 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

- Install and manage Cloud Foundry (CF)
  - All documents for CF: [https://docs.cloudfoundry.org/](https://docs.cloudfoundry.org/)
  - Simple CF deployment on one machine
    - CF deployment on a cluster
      - Install independent CS instances to a cluster of machines
      - Use a router to route incoming http requests to the CF instances securely with certain load balancing routing policy
      - You can write your own router or use an existing router
    - An example of an existing router: The reverse proxy NGINX
      - [https://nginx.org/en/docs/](https://nginx.org/en/docs/)
  - You will need a single UAA

- 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

- 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
  - Deploy the basic RoboCode on Cloud Foundry (CF)

- Multi-tenancy access control on CF
  - Current CF provides a primitive multi-tenancy infrastructure (orgs and spaces), UAA authentication and authorization, but does not really provide access control enforcement
  - Need to define your multi-tenancy access control model and its correlation to CF multi-tenancy access control model
    - The access control model within each tenant
    - The access control model for cross tenant sharing
    - The access control model for the administrative services
      - 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
    - Subjects
      - Mapping system subjects to the multi-tenancy access control model you have
    - Objects
      - Services, data, access right assignments
  - Need to provide mechanisms to enforce access control
Beyond what CF is doing
- APIs for applications for access control enforcement
  - Alternative: modify the application to incorporate the access control mechanism for each application
- Database proxy (query modification and access control)
  - Alternative: modify the database queries manually in each application

♦ You need to prepare some demo case to show that your access control policy is being enforced
  - Create several tenants
  - For some tenants, create many users
  - For each user, creates many robots
  - Show that your access control policy is being enforced (properly designed policy)
    - In a single tenant, cross space sharing
    - User concept mapping and cross user sharing
    - Cross tenant accesses
  - Show that you can change your access control policy dynamically
  - Show that your updated access control policy is properly enforced

➢ Simple performance study
  ♦ Original performance study
  ♦ At least study the performance difference due to the use of CF (compared to original RoboCode)

Final submissions
➢ Report and code should be submitted following the original main project document
➢ Final report should also include (in the design section, beside other design issues)
  ♦ Access control model in your system
    - Detailed
  ♦ Access control enforcement mechanisms