cs4471A/cs9549A – Software Architectures Class Project

Introduction

In this project, groups will create a "microservice-based" system to operate in a Cloud environment, to provide certain services to potential clients. The cloud infrastructure of choice is IBM Cloud. If your group would like to use another Cloud platform instead of IBM Cloud, then you can do so as your own risk and cost. In this case, please make sure that you document your choice in the first progress report.

The central architectural pattern of interest here is Microservices.

Further reference:

https://microservices.io/

++

Design and Implementation requirements

The microservice architecture has at least three components, as described below:

1. **Service Registry/Discovery:** An application component that manages service registrations, maintains an inventory of available services, and processes search queries from service consumers (users). It returns relevant information such as the name, description, and URL of the requested services.
2. **Service Consumer:** A front-end application that enables users to discover, from the Service Registry, the services they need. The application should include a GUI that features, for example, a search box and a list of clickable search results. When clicked, these results should redirect the user to the corresponding service page.
3. **Services:** A set of loosely coupled and independently deployable microservices that communicate with each other via HTTP, AMQP, or gRPC protocols.

Additional Components: Besides the three core components described above, your system can have numerous other components (e.g., UI, DB, Reporting, etc.) depending on the specific requirements of your project.

Key Requirements:

1. Services should be dynamically registerable and removable. "Dynamically" means that services can be added or removed during normal system operation, without the need for downtime or maintenance. For instance, shutting down one service should not impact the others, and the service should be automatically removed from the registry.

2. The system should be deployed on a cloud platform (IBM Cloud by default) using any kind of cloud infrastructure, such as virtual machines (VMs), containers (e.g., Docker), or serverless computing (e.g., IBM Code Engine), to ensure scalability and availability in a cloud-native environment.

Reliability/Availability Requirements:

The Service Registry serves as the focal point for service registration and query processing. Consequently, it poses a potential single point of failure that could compromise the system's reliability (and hence availability). It's thus important to consider scenarios involving downtime or system failures of the Service Registry and to implement appropriate solutions to mitigate these problems

For this class project, your system is encouraged to have capability to address at least one of the following questions (or other possible questions addressing Service Registry's *critical* quality attributes):

1. Is there an automated mechanism that, upon failure of Service Registry, redirects requests to a backup Service Registry?
2. What are the redundancy and recovery plans to minimize the loss of registered service data?
3. How will queued or in-process requests be managed during system failures?

++

There are two options for this project:

(i) *Option A is the default option*: it involves stock data of many companies over a period of a year. The data is available from: Alpha Vantage. The group can creatively design and implement services surrounding this dataset.

(ii) *Option B is where you need to define your own project*: For example, there are customer reviews of various products; thus, these reviews can become your dataset upon which you can create services for clients. Likewise for weather data or other. Discuss with the Prof. for feedback. Choose this option at your own risk!

Starter information for Option A:

An example type of service offered to individuals could be: given date ranges and company names, for example, the service can provide their stock performance over the date-range. Another example could be: given investment sector weights (e.g., forestry x%; hitech: y%; utilities: z%, etc.), the service could create one or more funds respecting the given weights but with composition of different companies along with the fund's historical performance data.

These are two, relatively straightforward, example scenarios. The group can create other such services.

++

Learning Outcomes

- Experience with requirements, architecture design, implementation, and testing of a microservice-based system.
- Familiarity with Cloud infrastructure (IBM Cloud in Option A or other in Option B).
- Groupwork.
- Project planning, execution, management, documentation, delivery, and presentation.
- Pertinent project-data gathering and analysis.

++

Development infrastructure, libraries, and programming languages

- For option A, an account is required for freely using IBM Cloud – details provided separately.
- The project requires the use of GitHub. One designated member needs to ensure that s/he hosts a private repository for the rest of the group. Group members are encouraged to work under their own branch of the repository.
- Each group is required to use intra-group communications and collaboration tools (e.g., "Slack", https://slack.com/).
- There are no constraints on the development frameworks or programming languages that can be used. Use of Spring (Java), Express (Node.js), Flask (python) or similar are encouraged, as appropriate, for creating microservices on the cloud.
- There are no constraints on hosting infrastructures (e.g., virtual machines, serverless) and support infrastructures (e.g., databases), third-party tools, or off-the-shelf (or open source) software that can be used in the project. Third-party software should be publicly available and must be referenced in the documentation.
- Use of Open-source tools and systems is encouraged.

Group member roles in the project

The project involves many different types of tasks (e.g., coding, designing, planning, testing, inspecting, documenting, and others.) and thus there are different roles (corresponding to such tasks) that group members need to assume at different times in the project.

Note that in the project a group member can play more than one role, concurrently or otherwise. Likewise, more than one group member can play the same role (e.g., multiple requirements analysts, collaborative architecting, parallel developers and testers, etc.). The distribution of roles will need to be adapted to the group size.

Here, we list *example* roles and associated tasks for consideration in a group. The group can decide on the roles to be played and by whom. The project, however, encourages diversity in roles and different role-playing by any given member of the group.

- ***Project manager***: project planning, task allocation (with consensus), task tracking, progress monitoring, alerting, information sharing in the group, etc.
- ***Liaison person***: interacting with the instructor, TA, etc., and the rest of the group.
- ***Quality Controller***: ensuring that requirements are met (e.g., testing someone else's code, etc.).
- ***Quality Assurance***: planning for quality control (e.g., developing test cases for Quality Controller, planning what components will be tested when and by whom, etc.).
- ***Builder***: integrating software components as per the build plans.
- ***Designer***: designing (part of) the system.
- ***Programmer***: develop code for given component specification and do unit testing.
- ***Requirements engineer***: eliciting, analysing and prioritizing requirements.
- ***GUI designer/developer***: creating screen layouts, graphical user interface design, etc.
- ***Design and code inspector***: design and code reviewer.
- ***Project documenter***: documenting software artefacts (plans, designs, requirements, etc.)
- ***Subject matter expert*** (person with knowledge and experience on specific items in the project): dig up tutorials, guidelines, literature on the subject matter and share this knowledge (though demos, examples, presentation-talks, etc.) with other members in the group. Examples of such items include:
    - Cloud technologies
    - Microservices
    - Video and demo preparation etc.
    - Specific programming and design languages: Java, Python, C++, Node.js, PHP, UML, etc.
    - Design patterns
    - GUI
    - GitHub
    - Design and Code inspections.
    - Etc.

+++

Project Binder

The project is to be documented where the files should be readable by the following applications:

- Windows applications: MS WORD, Powerpoint, and Excel.
- Arcobat Reader (PDF files)

Those artefacts that are not readable by the above listed applications will not be graded and, consequently, will receive a mark of zero for that portion.

In the project binder, there should be:

- A title page with course name, project title, project group identification and project member names.
- Table of Contents (TOC) page
- Any separate files (e.g., Excel files) should be clearly identified in the TOC.
    - Multiple files in a folder should be ordered (e.g., by prefixing the filename with a number) so that it is easy to find the key files in some logical order.
    - Haphazard file ordering will be penalised as deemed by the instructing team.

The **project folder** should be named: **Group** <number>_Project Documentation

The project folder should be zipped.

Submission to be made to OWL, instructions to follow later.

++

Deliverables

There are **two progress reports and a final report (with demo)** to deliver, as described below.

Each group needs to create a Project Binder containing the following sections.

Progress Report 1 (PR1):

- **Services offered**:
    - List of services to be offered by the system.
        - Each service description is of the format:
            - ID: "S"<num>
            - Service title (brief but informative)
            - Service description: what and why.
        - Populated in a separate tab in a spreadsheet.
- **Service requirements**:
    - Each service has a list of functional (FR) and quality (QR) requirements. Functional requirements indicate WHAT the service will accomplish – action-oriented (e.g., given some numbers, the service returns a sorted list). QR indicate the QUALITY of the service accomplished (e.g., Gold subscribers will receive higher priority responses than the Bronze subscribers).
        - Sometimes, a requirement may describe both WHAT and QUALITY aspects; it is best to split them up into their basic form (as sub-requirements).
    - The format of logging service requirements is as follows:
        - "S" <num>: <service title> (see above for service title)
            - "FR" <num> or "QR" <num> <Requirement description>.

Subrequirements have decimal points, e.g.: FR (or QR) 1.1, 1.2, 1.3, etc.

- **State "Why" (reason for this requirement) if not obvious**.
- Typically each service will have several FRs and QRs.

- Populated in a separate tab in a spreadsheet.
- *Separately, identify known Architecturally Significant Requirements (ASRs) – see ch. 16 from the classtext. Describe "why" these are ASRs; in other words, in which way do you think they do/should/will influence the system architecture?*

➢ **Project PLAN** (chart and spreadsheet):

- ***Chart***: Iterations planned with dates (high-level schedule).
    - Use of Gantt chart or similar techniques.
        - Which services are being planned for which iteration?
- ***Spreadsheet***:
    - Service id and title
    - List of tasks **to be done** to implement the service
    - **Show tasks status**: e.g., Not Started, In progress, Completed
    - Agent (group member) assigned the task

➢ **[OPTIONAL] Preliminary sketch of the system architecture**. For details on architecting, please see Progress Report 2.

➢ **Cloud platform (IBM Cloud or other)** -- Group to describe:

- The extent to which they are familiar with this platform
- How the project system will be built on top of the platform.

➢ **Microservice Architecture**-- Group to describe:

- The extent to which the Group is familiar with microservices.

- **Documentation and Submission:**
    - **PR1 will be documented as powerpoint slides, spreadsheets, text documents.**
    - **PR1 will be presented to the instructing team (and class as scheduled).**
    - **PR1 will be delivered to OWL (deadline provided separately)**

++

Progress Report 2 (PR2):

Artefacts in PR2 are cumulative in that updates to PR1 artefacts are overviewed (as appropriate) and, in addition, new artefacts not previously presented are presented too. There is a preliminary demo at this stage.

Updates (if any) to PR1:

- Services.
- Services requirements, e.g., changes to ASRs.
- Project PLAN.

Documenting Software Architectures:

(Please note: the references to the Bass/Clements/Kazman (BCK) book below are to Edition 3, which is easily obtainable.)

- **Please read Ch. 18** – Documenting Software Architectures, pp. 327-360 from the book: "Software Architecture in Practice", 3rd Edition, by Len Bass, Paul Clements, and Rick Kazman, 2013.

- Create a *Documentation Package* (see Section 18.6) for certain views:

  (i) **Structural views:** At least one Structural view of each type: (i) **Module view** and (ii) **Component-and-Connector view**. Optionally, (iii) **Allocation view** if you have reached the development stage where you have allocated software units to elements of the environment.

  (ii) **Quality Views:** Three different types of Quality views (see pg. 340-341):
  - Quality views are orthogonal to structural views. They depict software units that cut across different structural views. Examples of quality views include: a *security* view, *communications* view, *exception* or *error-handling*, *reliability*, *performance*, and others. Quality views depict the *behaviour* of a (part of the) system.
  - Please read Section 18.7 that describes how to document behaviour (e.g., using *sequence charts* – Figure 18.5, and *state models* – Figure 18.6, but other behaviour model types are also described, e.g., Use cases, communication diagram, activity diagrams, etc.). Such behavioural models are useful for documenting "*Quality views*". [Note, such behavioural models have emerged from the design of programs but are also useful at the architectural level where the systems are generally much larger and much more complex than programs.]

  NOTE: The structural views and Quality views are *not* simply diagrams on some lose-cut pages! Each view diagram must be documented in its TEMPLATE (see Figure 18.3 and the supporting text in five Sections – see pp. 345-347).

  (iii) **Combined Views:** Some views may be better shown in a "combined form" with other views, e.g., the "decomposition view" (hierarchical breakdown of a system) together with information on the agents (e.g., names and Depts.) that have been assigned to implement and test them. So, please read up Section 18.5 on how to combine views. Look for opportunities in your system to show a combined view and provide some supporting description of the combination.

(iv) **System Overview:** Please also include an abstract (or high-level) architecture planned or implemented. Show clearly the three key elements (described above).

- **See Figure 4.2** in the book: "DevOps – A Software Architect's Perspective", by Bass, Weber and Zhu, Addison Wesley, 2015. (I recommend reading Chapter 4 from this book to get a fuller context). [Note: the university library had this book and so you might like to borrow it for a short time so that others can also borrow it.]

In PR2, w.r.t architectural documentation, I expect to see solid examples of the above types of models documented at this stage. In the Final Report, I expect to see full architectural documentation.

- **Documentation and Submission:**
  - **PR2 will be documented as powerpoint slides, spreadsheets, text documents.**
  - **PR2 will be presented to the instructing team (and class as scheduled).**
  - **PR2 will be delivered to OWL (deadline provided separately)**

++

Final Report, System demo, and System artefacts:

- **Final Report**:
  - Building up on Progress Reports 1 and 2, your group will document the final state of the system. Please package your final report as you would expect from any system documentation, example:
    - **Format**: pt. size 12, single spaced, page numbers, etc.
    - **Title page** (project title, authorship, class details, instructor, Dept. and university)
    - **Table of contents**
    - **Body sections**, e.g.:
      - Introduction,
      - Requirements
        - fully described requirements.
        - Architecturally Significant Requirements (ASRs) clearly identified together with their justification
      - Architecture and Design
        - Context model
        - Overall system architecture with pub/sub design

        - Structural views, quality views, component interfaces for critical components, etc.

        [**Important:** diagrams without supporting explanations are not acceptable and will be downgraded heavily]

    - Implementation details
        - What tools and technologies were used to implement the system?
    - Testing (Examples only)
        - Different types of tests carried out (e.g., unit, black-box, subsystem, regression, etc.),
        - Key test cases and test results
    - Operational Evidence
        - Using screen dumps as operational evidence show step-by-step user interaction with the system and the output that is generated by the system
            - The screen dumps must be annotated to explain the interaction and outputs.
    - Appendix
    - References

- **System demo and supporting presentation**: please read up on good practices for doing a system demo together with supporting slides.
    - **Format of slides**: min. font size of 20; page X of Y; etc.
    - **Title page** (project title, authorship, class details, instructor, Dept. and university)
    - **Demo breakdown by project members:** a table showing who is going to demo what. It should be equitable.
    - **Timing specs** will be informed later**.**
    - **Demos will be scheduled separately.**
- **System artefacts:**
    - Submit the code of the system, executable, and supporting information on the required hardware, and instructions on how to run the system.

++

- **Submission:**
    - Please package separately the following items: (a) Final Report, (b) Presentation slides, and (c) System artefacts, and please submit to OWL.
    - Deadline provided separately

++

Project Assessment:
The overall percentage points gained will be weighted according to the project's weight in the course (please see the course description):
1) **Progress Report 1:** 10 points.
2) **Progress Report 2:** 20 points.
3) **Final Report and demo:** 70 points.

++

Group Work, Responsibility, and Peer Reviews

This project is group-work. Individuals in the group collaboratively create different parts of the system into an integrated whole. All the members of a group are thus expected to contribute earnestly according to the plan.

For any reason, if one or more members do not contribute as expected, the rest of the group is required to fill the shortfall.

Peer review may be conducted by the instructor if in his interpretation:

(a) there is significant complain from the rest of the group (not only one person);
(b) the complain is made to the instructor in writing – by the rest of the group -- at the time of the occurrence of the negative situation (any complain relating to matters over a week old at that time will not be accepted; the idea is to nip issues in the bud and not let it fester and explode);
(c) after preliminary inquiry the instructor is convinced that the situation warrants a peer review.
The instructor's decision is final and may not be appealed.
The project mark given to the individuals in the group would then be: the base project mark adjusted by *peer-review* feedback received from the group.

The peer review penalty is as follows -- % grade of the overall project mark reduced:
Minor infraction: 10%
Significant infraction but not considered major: 30%
Major infraction: 60%

NOTE:
(1) All interpretations will be made by the instructor and may not be appealed.
(2) A second time infraction by the same person will automatically lead to "major infraction" if previous case was assessed at "minor or significant" levels. If the previous case was assessed as "major infraction" then this will automatically result in a zero mark for the final project for the person concerned.
(3) The best policy is to collaborate with the group for everyone's success. Remember the old adage: *united we stand, divided we fall*.

++

END.