Academic Year:

Assessment Introduction:

Course: various	**Module Code: CO3404** **Module Title: Distributed Systems**
Title of the Brief: Microservices	**Type of assessment**: Implementation

This assessment is worth 50% of the overall module mark

This Assessment Pack consists of a detailed assignment brief, hand-in instructions and a video illustrating a first-class submission of the application. Lab sessions will be used to answer questions and provide direction. If you need additional support, please make a note of the services detailed in this document.

How, when, and where to submit:

Submission is via Blackboard. Details of what to submit are covered in this document.

Submission should be before 13:00 hours on Friday, March 22nd after which point, submissions will be automatically recorded as late.

Feedback should be provided by Friday, April 19th.

You should aim to submit your assessment in advance of the deadline.

Note: If you have any valid mitigating circumstances that mean you cannot meet an assessment submission deadline and you wish to request an extension, you will need to apply online, via MyUCLan with your evidence **prior to the deadline**. Further information on Mitigating Circumstances via this link.

We wish you all success in completing your assessment. Read this guidance carefully, and any questions, please discuss with your Module Leader.

Learning outcome from the module specification covered by this assignment, in full or in part:

1.	Critically evaluate patterns, technologies, and frameworks
2.	Compare potential technologies for the development of a distributed enterprise system
3.	Implement a distributed application using appropriate technology and frameworks

Additional Support available:

All links are available through the online Student Hub

1. Academic support for this assessment will be provided by contacting *Tony Nicol*

2. Our **Library resources** link can be found in the library area of the Student Hub or via your subject librarian at SubjectLibrarians@uclan.ac.uk.

3. Support with your academic skills development (academic writing, critical thinking and referencing) is available through **WISER** on the Study Skills section of the Student Hub.

4. For help with Turnitin, see Blackboard and Turnitin Support on the Student Hub

5. If you have a disability, specific learning difficulty, long-term health or mental health condition, and not yet advised us, or would like to review your support, **Inclusive Support** can assist with reasonable adjustments and support. To find out more, you can visit the Inclusive Support page of the Student Hub.

6. For mental health and wellbeing support, please complete our online referral form, or email wellbeing@uclan.ac.uk. You can also call 01772 893020, attend a drop-in, or visit our UCLan **Wellbeing Service** Student Hub pages for more information.

7. For any other support query, please contact **Student Support** via studentsupport@uclan.ac.uk.

8. For consideration of Academic Integrity, please refer to detailed guidelines in our policy document . All assessed work should be genuinely your own work, and all resources fully cited.

A Microservice-based distributed systems architecture

Overview

This assignment will assess your ability to implement, test and deploy a distributed application as a service, across a distributed system of microservices. You will gain experience of developing and deploying a solution using a modern software architecture pattern, utilising a modern hosting approach and tools used in industry, particularly in large organisations.

The complexity of the microservices is intentionally basic as this is a systems' module so is focussed on service independence, reliability, scalability and security.

Independence will be achieved by implementing a microservice architecture pattern.

Reliability will be achieved by loose coupling of the microservices and the use of appropriate data storage and recovery patterns including database, message broker persistence and data volumes.

Scalability will be achieved by deploying each microservice into containers hosted on its own virtual machine to enable scaling using, for example, scale-sets and load balancers, Kubernetes or other scaling technologies (not required for this assignment).

Security and scalability will be achieved by using an API gateway with appropriate authentication, secure web communication and decoupling of the services from the internet.

Four microservices will be accessed using the single IP address of the API gateway and communicate with each other using message brokers for asynchronous communication and direct http for synchronous messaging within a secure private network.

NOTE: develop and test using local host before cloud deployment, so you do not exhaust your Azure credit. Stop or delete virtual machines when not in use. It is your responsibility to monitor your cloud usage which is something a developer needs to be aware of in industry.

High level requirements

You will develop, test, deploy and document a Jokes Service. Users can request a random joke and submit a new joke. A moderator can moderate new jokes and an analyst can analyse patterns of usage. The full service will consist of four microservices using the architecture illustrated on page 11. However, you can choose how much of the architecture you implement based on the grade you are aiming to achieve.

Your microservices should be loosely coupled, independent and resilient to failure of any of the other microservices. The high-level requirements for each of the four microservices are outlined below. Detailed requirements are provided in the next section.

Joke microservice. Deliver a random joke on request by a user from a web page. The user can select the type of joke from an up-to-date list of available joke types which are stored in a mysql database.

Submit microservice. A user can submit new jokes. They will submit their jokes using a web page and select from an up-to-date list of joke types to categorise their submission. The joke will be written into the database, or to the moderator depending on which architecture you choose.

Moderate microservice. A moderator can check the suitability of a submitted joke. They can edit the joke, override the joke type the submitter has chosen if necessary or add a new type based on the joke content. They can either submit the new joke to the joke microservice, or delete it if, for example, it isn't considered funny or is inappropriate.

Analytics microservice. This service should receive logs from the moderator microservice and store the data in a MongoDb database. The microservice should be accessible from an analytical application such as excel, power BI or a tool of your choice. The analysis is not important, the ability to connect to the data and ingest into an analytical application is the goal; a simple graph or chart of say, number of jokes rejected vs number of jokes submitted would be adequate - you choose the application, what you display and how.

Detailed requirements

Read with reference to the architectural diagram on page 11.
Some of the components and microservices are optional so carefully check the assessment criteria on page 10.
Any communication between components within the microservice should use a Docker network and Docker DNS service names. Communication between microservices deployed on Azure virtual machines (VM) should use VM private IP addresses.
Each microservice should be resilient to any or all the other microservices failing; this should be tested and demonstrated.

Note: if you feel that any of the requirements are ambiguous or you don't fully understand them, do not guess. As a software developer, you would go back to the business analyst, or product owner for clarification as implementing incorrectly is expensive to the business. Also, if it's not a requirement then don't implement it as over-engineering a solution is also a drain on company resources and extends delivery times; make your code flexible and easily maintainable so future enhancements are relatively simple to add.

Detailed requirements

Joke microservice

The joke microservice consists of three components: an application (joke), a message consumer (etl) and a database (mysql). Each will run in its own Docker container.

1. Create a basic web user interface to request a single joke of a specific type selected from a list of available types
2. The "types" list should be refreshed from the database on a page reload or when the dropdown is clicked or interacted with
3. The joke component should run on a nodejs express server
4. The Web page, css and javascript client files should be served up as static content from the node server
5. The joke component should have two endpoints:
    a. **/type** returns the current list of types in the database to refresh the "types" list in the UI
    b. **/joke** returns one or more jokes of a specific type, selected from the database and returned in a random order
    This endpoint should take between zero and two query values:
    If none are provided, the API returns one joke of type 'any'
    If 'type' is provided, it should provide a single joke of that type
    If 'type' and 'count' are provided, then 'count' jokes of type 'type' should be returned
    Although the UI only requires one joke, your endpoint should be capable of returning more as others wishing to use this API should not be restricted to a single joke. i.e. the API should be flexible. Demonstrate using Thunder client
6. The node server and application should be deployed into a Docker container
7. The database should be mysql server and deployed into a Docker container. This is the master data store for the whole jokes service. Other microservices will work on the eventual consistency principle with regard to the jokes and their types
8. If a joke type read from the queue is not present in the database, it should be added
9. A persistent volume should be created to ensure the mysql database data survives a container restart or re-creation
10. An Extract Transform and Load (ETL) application should be implemented to extract jokes from a message queue, transform the data as required, then load it into the database. This application should be implemented as a RabbitMQ queue consumer which detects available jokes, reads them from the queue and writes them into the database independently of the joke application
11. Note: etl consumes from either the moderate microservice or the submit microservice depending on how much of the architecture you are implementing
12. The etl application should run on its own nodejs server deployed into a Docker container

Submit microservice

The Submit microservice consists of two components: an application (submit), and a RabbitMQ message broker configured as a basic queue (SUBMITTED_JOKES). Each will run in its own Docker container. The rmq-admin queue administration application is already included in the RabbitMQ Docker image so is deployed within the same container

1. Create a basic web user interface to submit a new joke of a specific type selected from an up-to-date list of available types
2. The "types" list should be refreshed on a page reload, dropdown click or other interaction, to reflect any new types that may have been added to the joke microservice database
3. The submit component should run on a nodejs express server
4. The Web page, css and javascript client files should be served up as static content from the submit component's node server
5. The submit component should have three endpoints:
    a. **/types** returns the current list of types to refresh the "types" list in the UI
    b. **/sub** receives the new joke posted from the UI. The new joke is written to the message queue
    c. **/docs** returns OpenAPI (Swagger) compliant documentation for the endpoints of this microservice. This should be demonstrated
6. If the joke service is operational, the types are provided by an http request to the joke /types endpoint from the submit component
7. If the jokes service is down, submit should have a backup copy of the types to ensure the microservice is resilient to joke microservice failure. This should be implemented using a file stored on a Docker volume

Moderate microservice

The moderate microservice consists of two components: an application (moderate), and a RabbitMQ message broker configured as a basic queue (MODERATED_JOKES). Each will run in its own Docker container. The rmq-admin queue administration application is already included in the RabbitMQ Docker image so is deployed within the same container. The moderate component reads a message from the SUBMITTED_JOKES queue if one is available and displays it in the UI. If there are none available, an appropriate message should be displayed and the UI should poll for a new joke. When a joke arrives, the moderator can simply submit it, edit it, change its type, or provide a new type then submit it. Alternatively, the joke can be disposed of and the next one, if available, should be loaded into the UI and reflect the correct type in the "type" dropdown box

1. Create a basic web user interface to display a submitted joke read from the SUBMITTED_JOKES queue
    a. Display the setup and punchline in editable web UI elements
    b. The joke type read from the queue should be used to display its type in the types list as the selected type by default
    c. The moderator can choose a different type from the dropdown if they wish
    d. The moderator should be able to add a new type which overrides the submitted type
    e. The moderator has the option to submit the joke or delete it
    f. The moderator UI should display a new joke on page load if one is available. Note: don't worry about losing jokes by further refreshing the page as that's just further front-end implementation
    g. The moderator should be able to delete a joke - basically don't write the current joke to the queue and request a new one
    h. If no joke is available, start a timer to poll the /mod endpoint
2. The "types" list should be refreshed on a page reload or on interacting with the dropdown list to reflect any new types that may have been added to the joke microservice database
3. The moderate component should run on a nodejs express server
4. The Web page, css and javascript client files should be served up as static content from the submit component's node server
5. The moderate component should have two endpoints:
    a. **/types** returns the current list of types to refresh the types list in the UI
    b. **/mod** reads the SUBMITTED_JOKES queue and returns a joke to the UI or displays a response indicating there are no jokes available
6. If the joke service is operational, the types are provided by an http request to the joke /types endpoint from the submit component
7. If the joke service is down, the moderator component should have a backup copy of the types to ensure the microservice is resilient to joke microservice failure. This should be implemented using a file on a Docker volume
8. The moderator is required to authenticate to gain access to the moderator microservice

9. Note: *The following is only required if the analytics microservice is implemented*
   The moderate component should write log messages to the LOGGED_JOKES_QUEUE
   Each message should contain:
   a. The submitted joke
   b. The moderated joke (even if it's not been changed)
   c. A changed field (true or false)
   d. The moderator's username, API key or other identifying feature you choose if one is available
   e. The date and time the new joke was read
   f. The date and time the moderated joke was submitted
   g. Anything else you feel is useful

API Gateway

This should be implemented using Kong.

1. All microservices are available from a single IP address at port 80 or 443
2. The gateway has three endpoints:
   a. **/joke/** or **/joke/index.html** to retrieve the UI and access the joke API
   b. **/submit/** or **/submit/index.html** to retrieve the UI and access the submit API
   c. **/mod/** or **/mod/index.html** to retrieve the UI and access the moderator API
3. You can assume Kong is resilient and will not fail as it is resilient by design
4. You are only required to run Kong without a database but you can use a database if you have a requirement to do so
5. Kong should be run in a Docker container

Analytics microservice

This microservice is used to collect and store log data for analytics. There is no UI for this microservice, a third-party application should be used to consume from this service. You have much more flexibility with this microservice to develop it as you wish within the constraints below. i.e. you can choose your implementation framework and language, endpoints etc. You only have two fixed requirements:

1. The database should be mongodb either running in a container or as DBaaS using Mongo Atlas. Your choice - but the data should be persistent
2. The third-party application of your choice should consume the data via the Kong gateway and display some form of simple analytical output

Video presentation

A 10 minute video presentation (+/- 1 minute). This should cover the operation of your solution, key tests, and a discussion of the implementation.
For the lowest level grade, the report and video should focus on detail of the implementation and testing at the code level. i.e., the video narration should include a line-by-line discussion of how the code works, demonstrate the error checking, show the operation of the application, and clearly illustrate the tests and responses. If you find your video is less than nine minutes then you are unlikely to be explaining it in enough detail or not covering all the points listed which will affect your grade.

For the highest grade, there will be more to discuss so focus on the operation, key tests of resilience, any novel or complex code with particular focus on any optional additions.

For grades between the extremes, the discussion moves from boilerplate detailed code discussion towards key operational functionality, resilience and complexity.

The video should be submitted as a **single** standard .mp4 format - i.e., **not** as part of a .zip file. Your video should be playable in Windows so test it before submission.

Report

The report, no more than 2000 words, will partially assess learning outcomes 1 and 2; they will be more comprehensively assessed in the exam.
The key components of the report are:

1. Self-assessment / critical evaluation of how well you have satisfied the requirements and what areas could be improved. Using the assessment criteria, **propose an overall grade justifying the mark you are awarding yourself**. This will help you to carefully review the requirements and critically assess your own work. Your mark will not count but will be used as a measure of your ability to critically self-assess. i.e., it should be close to my awarded mark as I will be following the same specification.
2. A critique of the design patterns used in this design and a potential alternative or improved approach with justification supporting your proposal.
3. A table of tests carried out and results. These should be discussed and demonstrated in the video.

The report should be submitted as a **single** Microsoft .docx format file, **not** as part of a .zip file or any other file format.

Source Code
Source code should be submitted as a **single .zip** file. The file must be decompressable in Windows. Test it before you submit it.

Do **NOT** include node_modules

I should be able to easily recreate your application - I'll add the node modules where required using your package .json files and I'll create containers using your .yaml files

Assessment criteria

The video and report are required for all grades including a bare pass and their quality should be considered in your self-assessment as it will be in mine.
You should complete the requirements of the lower grades before attempting the higher grades.

40% - Bare pass
Fully implement and test all requirements of the Joke microservice on a local machine
You will need to provide jokes either from a test producer (you can use a modification of one of my demos), or enter the messages into the queue manually through the message broker administration console.

41% - 45%
Deploy the joke microservice to the cloud.

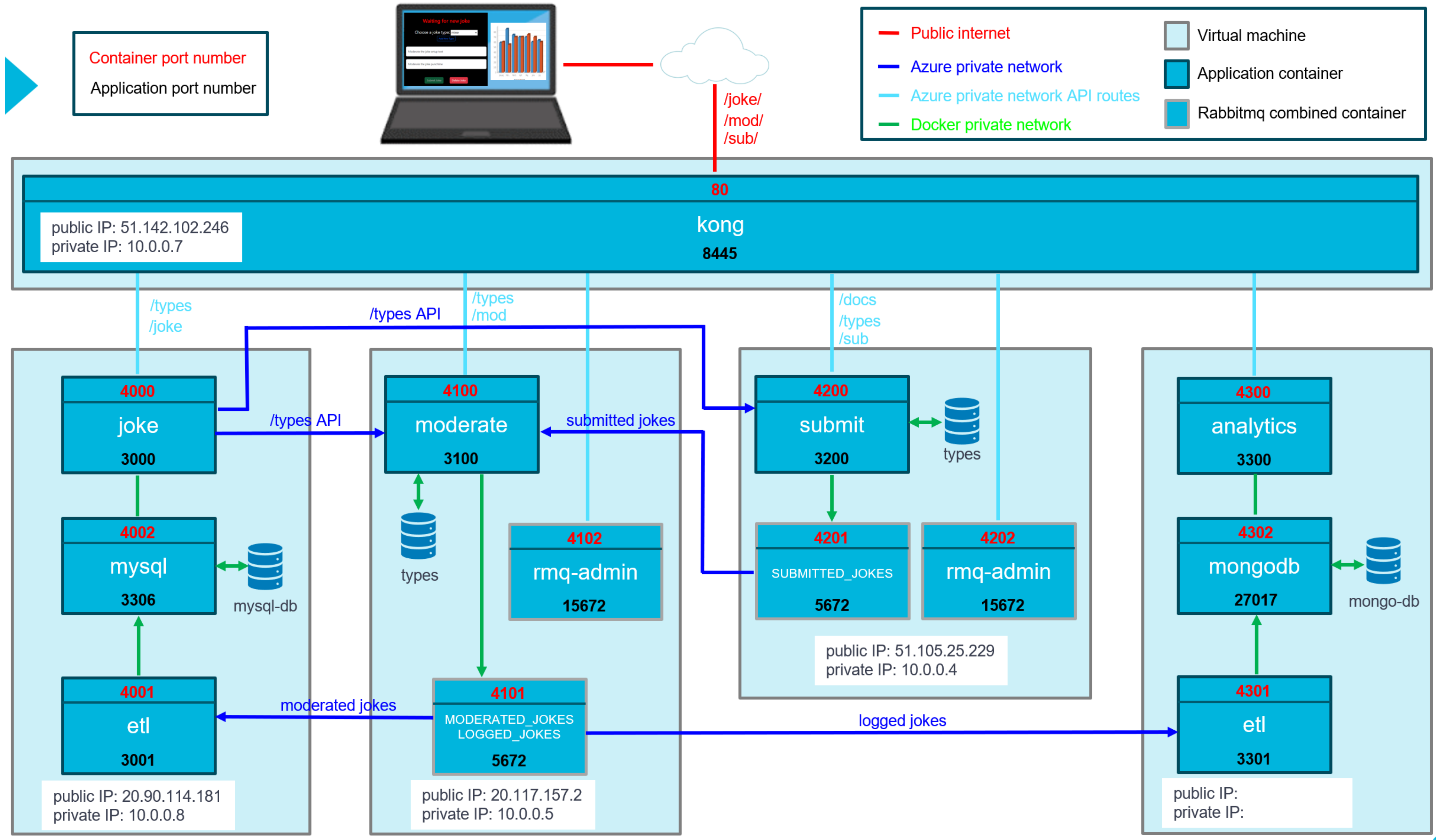
46% - 65%
Fully implement the Submission microservice other than requirement 7.
Connect the submission microservice to the joke microservice such that submitted jokes are added to the database via the message queue SUBMITTED_JOKES.
Deploy the Kong API gateway to access both services.
Deploy and test in Azure.

66% - 75%
Implement the moderate microservice other than requirement 8: *authentication*.
Implement requirement 7 of your submission microservice: *types backup*.
You do not need to implement requirement 9 as this is for a higher mark.
Add another route to the API gateway.
Deploy to Azure.

76% - 100%
For this band you will need to implement one or more additional components based on personal research. Focus on the **implementation** and test of these in the video and report.
The mark will be limited to 100%.
Select from the following:

10% Implement the analytics microservice and add it to the gateway.
10% Implement moderator authentication in the Kong gateway.
10% Implement https at the Kong gateway entry point.
10% Document the Moderator microservice but deploy from Kong rather than mod server.

System Architecture



Feedback Guidance:

Reflecting on Feedback: how to improve.

From the feedback you receive, you should understand:

- The grade you achieved.
- The best features of your work.
- Areas you may not have fully understood.
- Areas you are doing well but could develop your understanding.
- What you can do to improve in the future - feedforward.

Use the WISER: Academic Skills Development service. WISER can review feedback and help you understand your feedback. You can also use the WISER Feedback Glossary

Next Steps:

- List the steps have you taken to respond to previous feedback.
- Summarise your achievements
- Evaluate where you need to improve here (keep handy for future work):