DAIICT – End-Semester Examination – Autumn 2021
IT627 – Cloud Computing

PART I
MM: 20 Time: 60 mins

Instructions:
1. There are 5 questions in Part I
2. Each question carries 4 marks
3. Give suitable examples to substantiate your answer wherever possible.

Q1. What is a poison message in context of service bus queue and how does it handle such message and enables developers to diagnose the message at a later point in time?
A.
Poison message -> A situation where a message that arrives in queue and the handler code crashes for some reason while trying to process. Because the handler code does not complete its execution due to the crash mid-way, the message is not removed from the queue. Azure again invokes the same handler code because it has to invoke handler code when a message is present in the queue. But handler code would crash again because there is no change in the message – the content of message that caused the crash is not changed, neither has the code of handler – hence the crash will happen again. This results in repeated crashes and whole system gets choked because the offending message never gets removed from the queue. Such message is called a 'poison' message – it acts as poison pill for whole system.
Azure bus service allows handling of poison message using – Retry logic & Dead-letter queue. Basically, after a given number of retries to handle the massage, Azure service bus moves the message to another queue called Dead-letter queue. This way the main queue does not get choked for long and the system works fine. At the same time, the offending message is available in dead-letter queue (another message queue) for developers to take a look and diagnose the issue.

Q2. What are the key identifying characteristics of a Microservices architecture?
A.
1. Each Microservice has its own database.
2. No Microservice queries any other microservice's database directly.
3. Microservices communicate with each other using standard mechanisms like REST API, message queues etc
4. Microservices are hidden away by services like API gateways. API gateways expose consistent and uniform API to the users/clients and forward the request to relevant Microservice API internally. This way details of Microservices deployed by a system are not exposed to outside world ensuring encapsulation and reducing the threat surface area.

Q3. How does Cosmos DB enable horizontal partitioning and global distribution?
A.
All the data within an Azure Cosmos DB container (e.g. collection, table, graph etc.) is horizontally partitioned and transparently managed by resource partitions. A *resource partition* is a consistent and highly available container of data partitioned by a customer specified *partition-key*
it provides a single system image for a set of resources it manages and is a fundamental unit of scalability and distribution
Customers can elastically scale throughput of a container by programmatically provisioning throughput at a second or minute granularity on an Azure Cosmos DB container.

Give example – eg. Airport names as shown in lecture slides

Q4. In Microservices based architecture, one of the challenges is how to perform queries about information that is to be retrieved from multiple Microservices running underneath the application. How would you solve this problem while adhering to basic tenets of the architecture?

A.

In a large system, there could be many microservices. As per the principles of this architecture, each microservice must have its own database and no microservice could query database of any other microservice. To operate within these constrains, the microservices communicate with each other through standard protocols like HTTP (REST API), message queues etc. This could become a potential performance issue if the business scenarios make microservices architecture very chatty.

To handle this situation, CQRS (Command Query Responsibility Segregation) mechanism is used. In CQRS, the requests coming in to the system are segregated (based on their URL or REST request type) into write (Commands) or read-only (Queries). Based on the Queries part – which are read-only requests to potentially many microservices – an analysis is done of the data objects. The data objects that are frequently queried are put into separate read-only tables. This is call Materialized View Pattern – ie the view (ie the data seen by user) is "materialized" or put into separate read-only tables. These tables are accessible directly by either the api gateway or the first microservice in the chain. This reduces requirement of having to query multiple microservices to build response data objects. The updates to the MVP tables are done by worker processors at some suitable frequencies automatically using scheduler services like Azure functions or AWS Lambdas.

Expect an example of e-commerce system with microservices like Catalog, Orders, Users etc.

Q5. Describe any two of the following three consistency models in Cosmos DB:

A.

Bounded Staleness:

- Bounded staleness consistency guarantees that the reads may lag behind writes by at most *K* versions or prefixes of an item or *t* time-interval.
- Therefore, when choosing bounded staleness, the "staleness" can be configured in two ways: number of versions *K* of the item by which the reads lag behind the writes, and the time interval *t*
- Bounded staleness offers total global order except within the "staleness window." The monotonic read guarantees exist within a region both inside and outside the "staleness window."
- Bounded staleness provides a stronger consistency guarantee than session, consistent-prefix, or eventual consistency. For globally distributed applications, use bounded staleness for scenarios where you would like to have strong consistency but also want 99.99% availability and low latency.
- Azure Cosmos DB accounts that are configured with bounded staleness consistency can associate any number of Azure regions with their Azure Cosmos DB account.
- The cost of a read operation (in terms of RUs consumed) with bounded staleness is higher than session and eventual consistency, but the same as strong consistency.

Session:

- Unlike the global consistency models offered by strong and bounded staleness consistency levels, session consistency is scoped to a client session.
- Session consistency is ideal for all scenarios where a device or user session is involved since it guarantees monotonic reads, monotonic writes, and read your own writes (RYW) guarantees.

- Session consistency provides predictable consistency for a session, and maximum read throughput while offering the lowest latency writes and reads.
- Azure Cosmos DB accounts that are configured with session consistency can associate any number of Azure regions with their Azure Cosmos DB account.
- The cost of a read operation (in terms of RUs consumed) with session consistency level is less than strong and bounded staleness, but more than eventual consistency.

Consistent Prefix

- Consistent prefix guarantees that in absence of any further writes, the replicas within the group eventually converge.
- Consistent prefix guarantees that reads never see out of order writes. If writes were performed in the order A, B, C, then a client sees either A, A,B, or A,B,C, but never out of order like A,C or B,A,C.
- Azure Cosmos DB accounts that are configured with consistent prefix consistency can associate any number of Azure regions with their Azure Cosmos DB account.

Part II

MM: 20 Time: 60 mins

Instructions:

1. There are 3 questions in Part II
2. Each questions marks are mentioned against it
3. Focus on describing key aspects of the cloud service you are using to devise solutions to the problems asked in these questions

Q6 [Marks 5]

An application developed by company deals which high volume of message based communication (like WhatsApp). The backend of this app has been implemented using Azure Service Bus Queue/Topics. The version 1.0 of the application was shipped a year ago by the company. In version 2.0 the company has made dramatic changes to both the User Interface and the back-end system. Because of this, the risk to success of the version 2.0 and stability of the back-end changes is high. Hence company has decided that it will roll out the app gradually to the users. At the back-end, both version 1.0 and 2.0 components will run simultaneously as part of single deployed solution.
Design a solution to this problem using Azure Topics.

A.

Azure Topics is a message queue service that provides filtering and subscription features. The handler functions that process the messages are attached to the queue through a subscription. Each subscription has filter attached to it. When a message arrives from the receiver side, the topic will apply filter from each subscription and the message is passed to those subscriptions whose filter condition accept the message.

Let's assume that version 1.0 of the product has same format of message that arrives in Azure topic and all messages are handled by a handler function (say, h1) with default filter on its subscription. In this setup all messages are passed to the handler function because default filter in a subscription allows all messages through it.

For version 2.0, let's assume that there is separate handler (say, h2). Now we've two handlers – h1 for version 1.0 messages and h2 for version 2.0 messages. We can do following modifications to the topic to enable it to handle both version of messages:

1. The message of version 2.0 must have a property whose value is unique to it. Let's assume the property name is 'version' and its value is set to 2.0 for all v2.0 messages.
2. Attach handler h2 to the topic through a subscription.
3. Delete the default filter of h1 as well as h2 subscription.
4. Add condition on filter of h1 subscription that queries that there is no "version" property – assuming that version 1.0 messages do not have any "version" property. Alternatively, h1 subscription can use any other property of message whose value is unique to v1.0 messages.
5. Add condition on filter of h2 subscription to allow messages if "version" property's value is 2.0.

Now, the version 2.0 application can be deployed in selected region(s) and the messages will be received by the modified setup. Both version 1.0 application (that is already deployed) and 2.0 deployments in selected region(s) send message to same back-end. Because of the filtering and subscription features of topic, it is now able to handle both kind of messages.

Q7 [Marks 5]

A company has training period for employees. It has around 500 training modules that an employee needs to cover over the period of 6 months. A module could be a video, document, e-book or other form or content. The requirement is to track the status of modules covered by employee over the training period. How will you design a data base solution to for this?

A.

1. The content is stored in Azure blob storage. Each content has thus a unique URL that can be used to fetch / download the content. This URL is used to refer to the content in other tables of the database solution.

2. The NoSQL service of Azure – CosmosDB can be used to store information of employees. We can use Azure Tables API of cosmos DB for this.

Following are the tables and their partition-key and row-key details with purpose of the table.

a) Table name: Employee
   Partition-key: Region (could be single region if company is small)
   Row-key: Employee ID – the employee ID is unique, at least within the region
   Value: All information about employee

b) Table name: Training Modules
Partition-key: Category/Class (of training module)
Row-key: Module ID / Name (must be unique within the category/class)
Value: All information about the training module, including the source URL.

c) Table name: Training Status
Partition-key: Region + Employee ID. The key is combination of partition and row keys of Employee table
Row-key: Category/class + Module ID. The key is combination of partition and row keys of Training Modules table.
Value: Contains object that has details of module that an employee is going to complete. It would've status property among others.

The given design allows quick query of the type where one needs to know what training modules an employee is going through and what their status are.

Example scenario: Let's say, an employee named Ramesh Jain joins the organization. The system will update the Employee table with one partition key (say, apac – ie asia pacific region) and an ID (say, 45756 – the ID of employee). The employee object will have all information of Ramesh Jain, like his date of joining, designation, communication address etc. Thus, following entry is made to Employee table:
Partition key: apac
Row key: 45756
Value: <employee object>

Training Modules table is already populated with the records. Example of such records could be:

Partition key	Row key	Module
HR	Benefits	URL: <url>, other details
HR	Reviews	URL: <url>, other details
Compliance	HIPPA	URL: <url>, other details

Then when employee registers for a training module, the Training Status table is updated with inserting an entry of partition key and row key prepared as described above. In our example, if Ramesh registers for Compliance module after completing HR-Benefits module, the Training Status table would have following entries for Ramesh:

Partition key	Row key	Status
apac-45756	HR-Benefits	Status : completed, <other details>
apac-45756	Compliance-HIPPA	Status: In progress, <other details>

Q8 [Marks 10]
Design a (combined or single) Microservices architecture based solution to the following scenarios in an ecommerce application.

1. Product viewing: Users or guests can view the products using various categories listed in the UI. (Assume that Search capability is not available yet and user will navigate through the categories to see the products). The item details shows among other things, the price of the item which may not be latest because latest price will be fetched during ordering.
2. Ordering: The user can click on an item to order it. The latest price of item which is now under ordering scenario is fetched and shown to user to take users' consent to proceed with ordering. For our example, we'll ignore the payment part. The ordering service takes from user the address to ship including the PIN.

3. Shipping: The shipping service is informed by ordering service about an order. The shipping service gets the shipping address and users the first digit of PIN for deciding the warehouse to intimate about shipping an order. The warehouses are located in different regions of the country. Upon receiving the order, a warehouse ships it.

Your architecture should include the following aspects:

- The internal detail about various Microservices should not be exposed to the client side
- Each bounded context should map to a Microservice
- Mention salient features of internal design of individual Microservices as well

A.

Based on the problem description above, it is clear that following bounded contexts emerge:

1. Catalog bounded context – this will represent feature of the system related to catalog of items. Basically, information about the items that user is interested to see including details of item, pictures of it, price, warranty, customer reviews etc.
2. Orders bounded context – this bounded context covers the workflow related to ordering of item post 'checkout' action from customer. It involves fetching the most updated price of item, stock available, shipping address, payment and such.
3. Shipping bounded context – this covers shipping of the item to customer which involves sending the order to relevant / closest warehouse, handling the logistics and tracking the shipping of order.

The bounded contexts can then be mapped to individual microservice. Additionally, Inventory microservice can also be created to be responsible for inventory information of items in various warehouses and their prices. This following Microservices are identified:

1. Catalog
2. Orders
3. Inventory
4. Shipping

Here, we want to see if we should apply Materialized View Pattern (MVP). The objective is to identify if there is a microservice that could become a bottleneck of performance and degrade user experience. Among all of the above, Catalog microservice could be potential performance bottleneck, because it needs to gather data about item (like, price from inventory microservice) and needs to show to user the information as fast as possible. Considering this, we decide that MVP will be implemented for Catalog microservice. That is, an MVP table will be created and kept upto date frequently with inventory information. Catalog microservice will have direct access to this table. A back-end scheduled job will update this table reading information about items from other microservices eg. from Inventory microservice. This way, Catalog microservice can respond to the requests much quicker because it doesn't need to query any other microservice.

Shipping microservice can employ Azure topics based solution which can filter the orders based on PIN in the address to send the order to right warehouse.

Using API gateway, we can avoid exposing above mentioned microservices to the client code. The API gateway can expose a uniform and singleton API to client and internally can call API of individual microservice to serve the requests.