Design on Notification Center

Requirement:

Design notification center : Send notification to user based on their Support Group and Product access. We will send the notifications when:-

1. the Question will have it's status changed to "PENDING" or a new Question will be inserted with "Pending" status.
*2. the score is changed.

*For score, we will consider environment score, as the change can be associated with subsequent changes to pillar or overall scores. This will avoid duplication notifications. We have determined that environment score is the most consistent low level score and hence it makes sense to send notifications based on that parameter.

Resources: There will be 3 lambda's and 1 new DynamoDB table. There will be two new endpoints created on API gateway's (Both BFF and Backend).

*Additional lambda scope is to insert/ update notification-map in notification center table. Notification map is mapping of users to support group for a product. **Remark**- Discussions Pending.

Lambda:

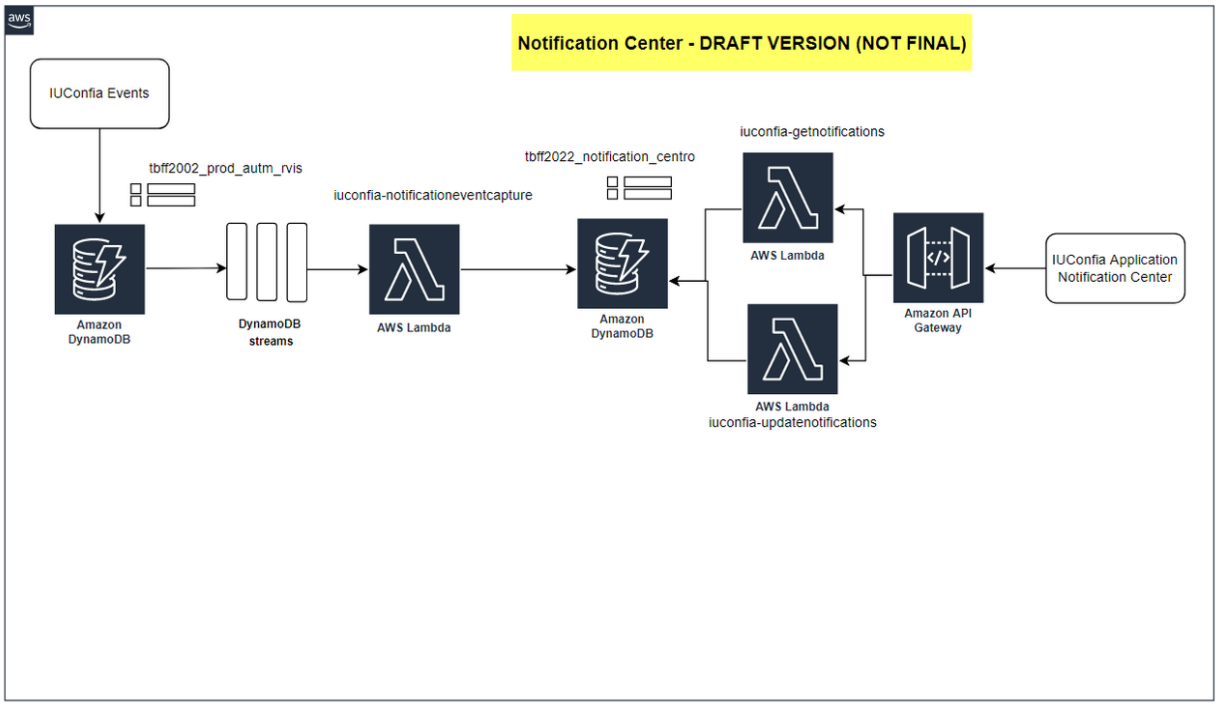
- iuconfia_productscorestreamconsumer
- iuconfia_getnotifications
- iuconfia_updatenotificationstatus

Dynamodb: **tbff2022_notification_centro** : schema defined below in here

API gateway :

1. Get notification (Get) /notifications
2. Post Update Notification /notifications (post)
3. Amazon API gateway integration with lambda to get notifications and update notifications

Arch Diagram:

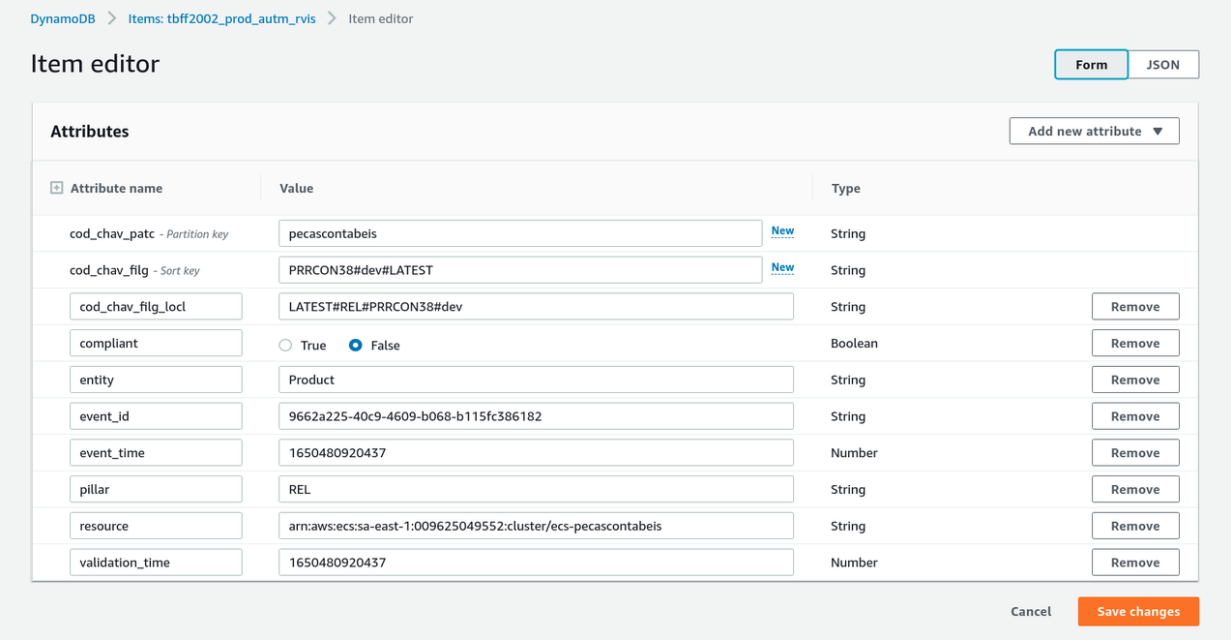


Work Item Description:

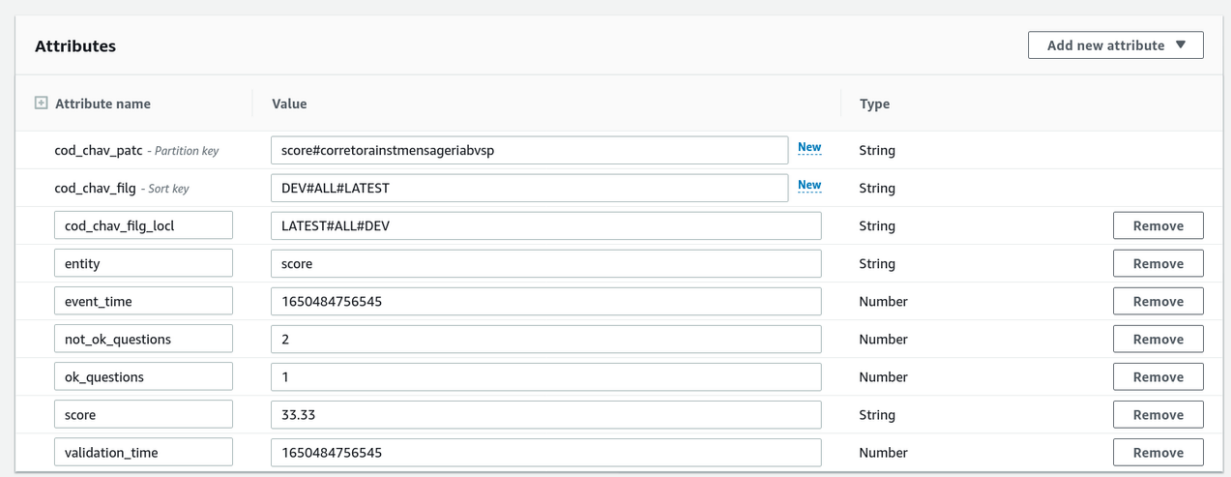
1. Creation of DynamoDB Stream to capture events - This is the functionality to capture DynamoDB item changes using streams for following two cases:

   a. When the item is updated/inserted for entity = “product”. This is to handle any questions that are having compliance = false and are seen as “pending” on UI screen.
   b. When the item for score#product is updated for cod_chav_filg = “<ENV>#ALL#LATEST. This to capture score changes on a product per environment.

Sample product#questionid Data from DynamoDB



Sample Score Data from DynamoDB



Filter for stream:

```
FilterCriteria:
        Filters:
          - Pattern: '{ "eventName": ["INSERT"], "dynamodb": { "NewImage": { "entity":
          - Pattern: '{ "eventName": ["MODIFY"], "dynamodb": { "NewImage": { "entity":
          - Pattern: '{ "eventName": ["MODIFY"], "dynamodb": { "NewImage": { "entity":
```

Lambda receives event for above event mapping filter pattern

1. Lambda: iuconfia_notificationeventcapture

a. capture stream events

Stream events captured for

1. Product compliance status pending:
    a. Lambda receives event for entity product, all DynamoDB streams will send old and new image of item.
    b. Sample item is shown in following table
    c. Parse item (received as new image). Attributes cod_chav_patc is product name(productid), parse questionid from attribute cod_chav_filg, compliance value is always false for new item (i.e. insert or update).
    d. For insert event there is no old image hence no old compliance value. For update old compliance value = True and new compliance value is False (as we capture transition from compliance to pending i.e. True to False)
    e. We get productid = productname, questionid= questionid, old_compliance_value and new_compliance_value after this step.

Sample item for product entity

cod_chav_patc	cod_chav_filg	cod_chav_filg_locl	compliant	entity	event_id	event_time	pillar	resource	validation_time
sandboxsre	PRRCON38#dev#LATEST	LATEST#REL#PRRCON38#dev	FALSE	Prodcut	some_has_value	123232321	REL	arn of resource	1323415623

2. Score update event:
    a. For score event we capture score update for each environment (no ALE acronym for aggregated score of all environment) and all pillar type (pillars refer to AWS well architected pillars).
    b. Each event will give old image and new image of score update.
    c. Sample item for score entity is shown in following table.
    d. Parse new item image attributes as below
        i. cod_chav_patc, gives product id, *product_name = item[cod_chav_patc].split("#")[1]*
        ii. environment, attribute cod_chav_filg gives environment value, *env = itme[cod_chav_filg].split("#")[0]*
        iii. pillar, attribute cod_chav_filg gives pillar information, *pillar= item[cod_chav_filg].split("#")[1]*
        iv. new_score, attribute score
    e. Parse old image attribute to get old score.
    f. We get, product_name, pillar, env as environment, new_score, old_score.

Sample item for entity=score

cod_chav_patc	cod_chav_filg	cod_chav_filg_locl	entity	not_ok_questions	event_time	ok_questions	score	validation_time
score#product_name	DEV#ALL#LATEST	LATEST#ALL#DEV	score	9	1.23E+16	3	24	
score#product_name	HOM#ALL#LATEST	LATEST#ALL#HOM	score	9	1.23E+16	3	24	

score#product_name	PROD#ALL#LATEST	LATEST#ALL#PROD	score	9	1.23E+16	3	24	
score#product_name	TOO#ALL#LATEST	LATEST#ALL#TOO	score	9	1.23E+16	3	24	

Once values are retrieved from the event, next step is to insert relevant values to notification center table.

b. Inserting user information to notification center table : (8 hours)

The first step will be to find which users notifications belong to. This is done by querying the notification map partition in notification center table.

Notification map partition in notification center table (**tbff2022_notification_centro)** stores user id list for product. There is unique entry for combination of productid#supportgroup in notificationmap partition.

The notification map partition is as follows

Sample item for notificationmap partition in notification center table

code_chav_patc	cod_ordernar_chav	status_ordernar	tipo_ordernar	userlist	
notificationmap	productid1#supportgroup1	supportgroup#productid	time#supportgroupid#productid	[u1,u2,u3]	
notiifcationmap	productid2#sg2	supportgroup#productid	time#supportgroupid#productid	[u1,u5]	

Steps:

1. Query table for partition notificationmap.
2. We will get product id from the notification events.. Product name or id is used to query primary index of notification center table on hash key cod_chav_patc = notificationmap, sort key cod_ordernar_key = begins_with(productid)
3. For each user, insert the record for that notification type in next step.

Important consideration:

Notificationmap is stored in Notification Center table (**tbff2022_notification_centro).**
This table has two additional local secondary index {attributes: status_ordernar , tipo_ordernar} these attributes cannot be empty.
Hence we need to add unique value for these attributes we will add
{status_ordernar: LATEST#ProductID#Group, tipo_ordenar: LATEST#Group#prodcutid}.

c. Logic for score and product event notification:

In this step we use data obtained from previous steps i.e. a and b to prepare data and insert into notification center table.

1. Loops and Iterate through list of users obtained in previous step.
2. For each user, we can get two types of events based on entity i.e. score or question.
3. In each loop, i.e. for each user we prepare JSON to insert into notification center table.
4. For Score entity type, JSON would have following values:
    a. cod_chav_patc : u1 , this is user id data
    b. cod_ordenar_chav: all#time (time = epoc time) : this is for all_events_time
    c. status_ordenar: unread#time (time used in cod_ordernar_chav): this is for read or unread status with time
    d. tipo_ordernar: score#unread#time (same time as used in attribute cod_ordernar_chav)
    e. env: environment
    f. pillar: pillar_information
    g. old_score: old_score
    h. new_score: new_score
    i. event_type: score
5. For Question entity type, JSON would have following values:
    j. cod_chav_patc : u1 , this is user id data
    k. cod_ordenar_chav: all#time (time = epoc time) : this is for all_events_time
    l. status_ordenar: unread#time (time used in cod_ordernar_chav): this is for read or unread status with time
    m. tipo_ordernar: question#unread#time (same time as used in attribute cod_ordernar_chav)
    n. env: environment
    o. compliance: false
    p. event_type: question
    q. stream_event: insert or update (this might not be required)
    r. question: question_id
6. Insert the record into notification center table (**tbff2022_notification_centro)**

Schema for notification centro table (tbff2022_notification_centro)

Primary index in notification center table:

Partition Key: cod_chav_patc : will store userid e.g. uid123

Sort Key: cod_ordenar_chav: will be used to store event time and used in query for getting all notifications e.g. all#123456789456

Local secondary index 1: will be used to store event and time and used in query for getting unread notifications e.g. unread#123456789456

Index Name: (lsi1: xxnc_2022) :
Attribute Name: status_ordenar

Local Secondary index 2: will be used store event, status and time. This will be used in querying unread or read notifications of certain type (question or score). This is not currently required but can be used in future. e.g. questiontype#unread#time

Index Name: (lis2: xxnc_2022_v)

Attribute Name: tipo_ordenar
type_read_unread_time : read#time#score or unread#time#score or read/unread#time#question

Other attributes:
entity: score or question
question_id : PRRCON38 (only product entity event)
product_id: product_name (only score event)
score_old: 45 (only score event)
score_new: 23 (only score event)
env: environment information (Both)

cod_chav_patc	cod_ordenar_chav	status_ordernar	tipo_ordernar	entity	env
u1	all#time	unread#time	questiontype#unread#time	questionupdate	dev
u2	all#time	unread#time	score#unread#time	scoreupdate	hom
u3	all#time	unread#time	questiontype#unread#time	question	dev
after update opertaions(probably UI will return notifications viewed whcih will delete from the above partitions and insert new record)					
u1	all#timeofupdate	read#time	questiontype#read#time	questionupdate	dev

CloudFormation Template for DynamoDB

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  dynamodbtable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: "cod_chav_patc"
          AttributeType: "S"
        - AttributeName: "cod_ordenar_chav"
          AttributeType: "S"
        - AttributeName: "status_ordenar"
          AttributeType: "S"
        - AttributeName: "tipo_ordenar"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "cod_chav_patc"
          KeyType: "HASH"
        - AttributeName: "cod_ordenar_chav"
          KeyType: "RANGE"
      BillingMode: "PAY_PER_REQUEST"
      TableName: "tbff2022_notification_centro"
      SSEEnabled: True
      LocalSecondaryIndexes:
        - IndexName: "xxnc_2022"
          KeySchema:
            - AttributeName: "cod_chav_patc"
```

```
              KeyType: "HASH"
            - AttributeName: "status_ordenar"
              KeyType: "RANGE"
          Projection:
            ProjectionType: "ALL"
        - IndexName: "xxnc_2022_v1"
          KeySchema:
            - AttributeName: "cod_chav_patc"
              KeyType: "HASH"
            - AttributeName: "tipo_ordenar"
              KeyType: "RANGE"
          Projection:
            ProjectionType: "ALL"
      Tags:
        - Key: "app"
          Value: "notificationcenter"
```

2. Lambda: iuconfia_getnotification:

This lambda retrieves notifications for user. There will be two types of notification retrieved. (All and Unread). The notifications have a paginated index that will be used to send a paginated list of notifications. The notifications will be sorted on time descending.

This lambda is integrated with API Gateway

1. path: /notifications, method type Get,
2. parameters to be passed
    a. userid=uid,
    b. lastitemretrieved= {json of item retrieved can be null for first request} ex :
        i. `"LastEvaluatedKey":{"HashKeyElement":{"AttributeValue3":"S"},`
        `"RangeKeyElement":{"AttributeValue4":"N"}`
    c. status = unread or read

Improvement, to give control of pagination size we can let user pass limit or size in request parameter, this value will be passed as Limit parameter in query api of DynamoDB. See below code example.

Retrieve paginated list of notifications.

For request parameter status= unread:

1. Query notification center table for first secondary index (xxnc_2022), begins with unread# sort in descending order with pagination enabled. (In code snippet listed below Limit decides pagination size)
2. Retrieve first set of paginated data using DynamoDB API call. Query on secondary index begins with unread for user id u1.
3. For next pagination data, (next request to the API) API will include LastItemRetrieved(map to LastEvaluatedKey) JSON value in request parameter. Logic in the code will check for LastEvaluatedKey value , if this is not null than

proceed query for next page pass LastEvaluatedKey as ExclusiveStartKey in query API of dynamodb. We will be using the Pagination capability of DynamoDB APIs

4. If LastEvaluatedKey is null, it will be termed as first request to notification API.

For request parameter status = read:

1. Similarly same logic follows for status=read, only change would be query on secondary index (xxnc_2022), begins with read# sort in descending order.

For request parameter status = all (logic is similar to case status=read or unread but query is done on primary index)

1. Query notification center table for primary index, begins with all# range key in descending order with pagination enabled. (In code snippet listed below Limit decides pagination size)
2. Retrieve first set of paginated data using DynamoDB API call. Query on secondary index begins with unread for user id u1.
3. For next pagination data, (next request to the API) API will include LastItemRetrieved(map to LastEvaluatedKey) JSON value in request parameter. Logic in the code will check for LastEvaluatedKey value , if this is not null than proceed query for next page pass LastEvaluatedKey as ExclusiveStartKey in query API of dynamodb. We will be using the Pagination capability of DynamoDB APIs
4. If LastEvaluatedKey is null, it will be termed as first request to notification API.

Only For developer:

a.

```
#sample python code shows how we can use pagiantion
#Key is existing class
#Sample input to query
#This will be added in common code in Dynamodb module.

query(
TableName= "notification_centro",
IndexName ="local_secondary_index",
Limit=20,
ScanIndexForward=False,
KeyConditionsExpression=Key(partition_key).eq(partition_key_value)&Key(range_
)

#Response will give us lastkeyreturned this can be used for pagination for ne


#when we make second request , UI need to send us last key access we need to
#we will pass ExclusiveStartKey


#Since we are using limit to get paginated data here ExclusiveStartKey values
```

Explanation of code:

query secondary index of notification center begins with unread to retrieve all unread notification for user id.
In the query pass `ScanIndexForward: false,` this will sort the result in descending order

ExclusiveStartKey is used to get next page results

Limit is passed to set the page size

Pay load for apigateway

```
Request:
/notifications
Method: Get
Request Parameters:
user = uid
status = unread or read or all
lastiemtretreieved= {Json of last item retrieved or null}

Response:
response would be same either if lastevaluatedkey null or not null, this will be for p

[{
"user":"1213",
"productid": "productname",
"question": "if question type return question id, null or empty for score",
"type": "compliance or score",
"old_score": "old score", # emtpy for question type
"new_score": "new score", # empty for question type
"compliance": "pending", # empty for score type
"env": "environment", # environment type dev, hom, prod, too.
"notification_identifier": "all#eventtime", # this is needed for UI to return during u
"LastEvaluatedKey" : {range_key: "uuid", sort_key: "lastitem_returned ie unread#time_t
}, .... ]
```

Note: (FOR UI)

notification_identifier <all#time>, this is very important JSON property.

This will uniquely identify notification in notification center table.
UI should store this value temporarily, return as request body with userid to update status of notification that has been read by user.

3. Lambda: iuconfia_updatenotifications:

This lambda will update notifications and will be Integrated with API Gateway

API Gateway
url: /notifications
method: post
Request Body: [{user: uuid, notification_identifier: all#time}, ... {}]

Here all#timeinnanonsecond is used to uniquely identify notification

```
Request:
url: /notifications
method: post
Body:
[{
"user": “uid”,
"notification_identifier": “all#time”
},
......
.....
]

Response:
{status: “Update completed successfully”}
```

UI will send list of notifications already viewed as JSON list of user and notification identifier.

Lambda will transform the request to match attribute of DynamoDB item:

Transformation:
user → cod_chav_patc
notification_identifier → cod_ordenar_chav

After transformation lambda will update notification center table

During update notifications it will involve two steps we will use transactional feature of DynamoDB to perform update in notification center table.

1. First delete the item from the DynamoDB using uid and all#time
2. Insert new item to the DynamoDB using same uid and all#time and read#time , here we don't change the time we will preserve same time for all#time while for read#time we can update the time.
3. Use TransactItemWrite for above two operations *

Questions (Discussed during design):

Documentation for other consideration during design

- Do we need score for ALE or each environment ?
  - Not ALE. We just need for each environment (after talking with Gustavo from Itau)
- Do we need score for each pillars or only ALL ?
  - Only ALL. We just need for Overall (after talking with Gustavo from Itau)

```
Filters changes "cod_chav_filg":{"S": [ { "anything-but": { "prefix": "ALE#" } } ]},"c
```

- If we put notifications_map in notifications centro table what would be 2nd LSI? -Pankaj Pattewar,

  notifications-map will be in notification center table

Tasks:

Task 1: lambda **iuconfia_notificationeventcapture (80 hours)**
- a. Write code for DynamoDB stream with appropriate filters. details see section a of lambda
- b. get user information by querying notification map
- c. create notification in notification center see detail section c of lambda

Task 2: lambda iuconfia_getnotifications (40 hours)
- a. implement logic for getting notifications for the user provided in request parameter
- b. integrate with api gateway notifications get method

Task 3: lambda iuconfia_updatenotifications (40 hours)
- a. implement logic to update notifications, update unread records and add read records using the hash and key range provided in request
- b. integrate with API gateway notifications post method

Task 4: Create DynamoDB schema and repo (16 hours)
- Description: Create cloudformation for DynamoDB to hold notification center details. Schema is as provided to Mauro.

Task 5: Create API gateway endpoints and integration (32 hours)
- Description: Create Get endpoint to retrieve all notifications for user. Parameterize the endpoint to return unread when asked by user.

Task 6: Integration Testing - (24 hours)
- Description: Do end to end testing by sending score and question updates.

Task 7: Getting notification Map - User-SupportGroup Info (Mauro)

Discussion Items:

*Enhancement to Current IUConfia functionality

1. Deep Link to retrieve question details for multiple questions on a product

*Notification Map design:
/ TO DO

FUTURE SCOPE: