IT491 – Cloud Computing **Date** – April 15, 2023

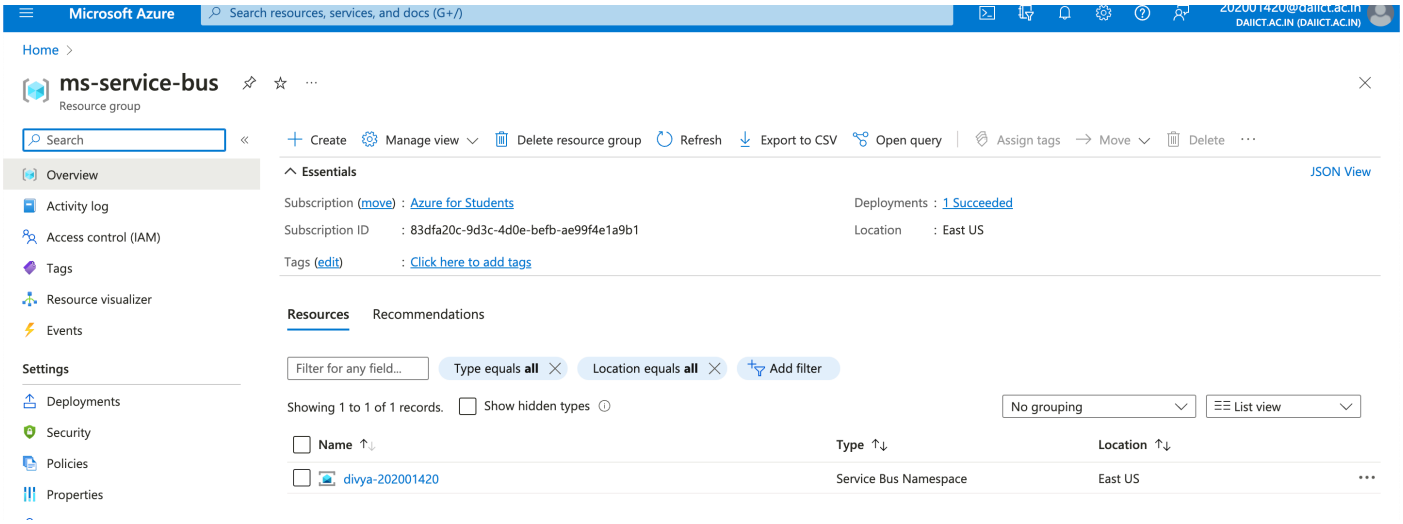
Lab Report - 6

Name – Divya Kirtikumar Patel

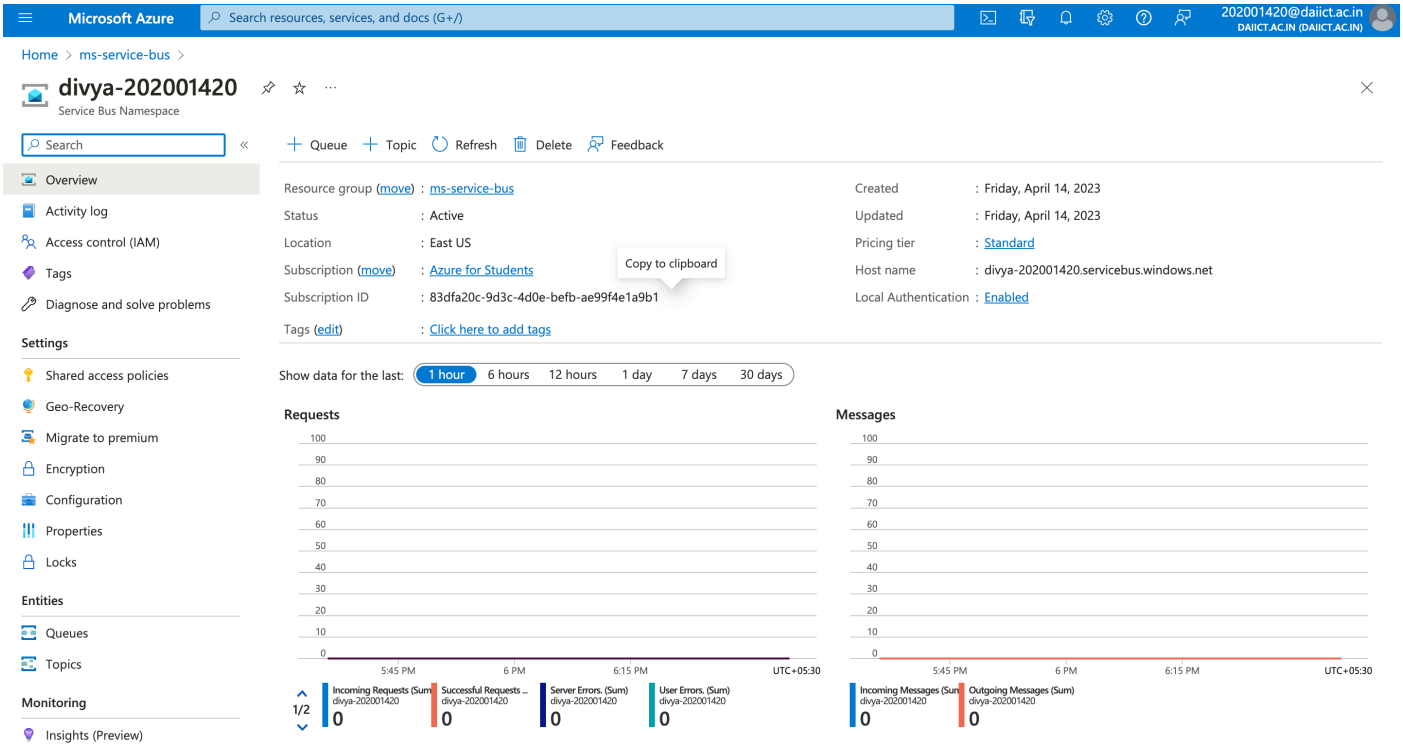
Student ID - **202001420**

1. Send messages to an Azure Service Bus topic and receive messages from subscriptions to the topic (Python)

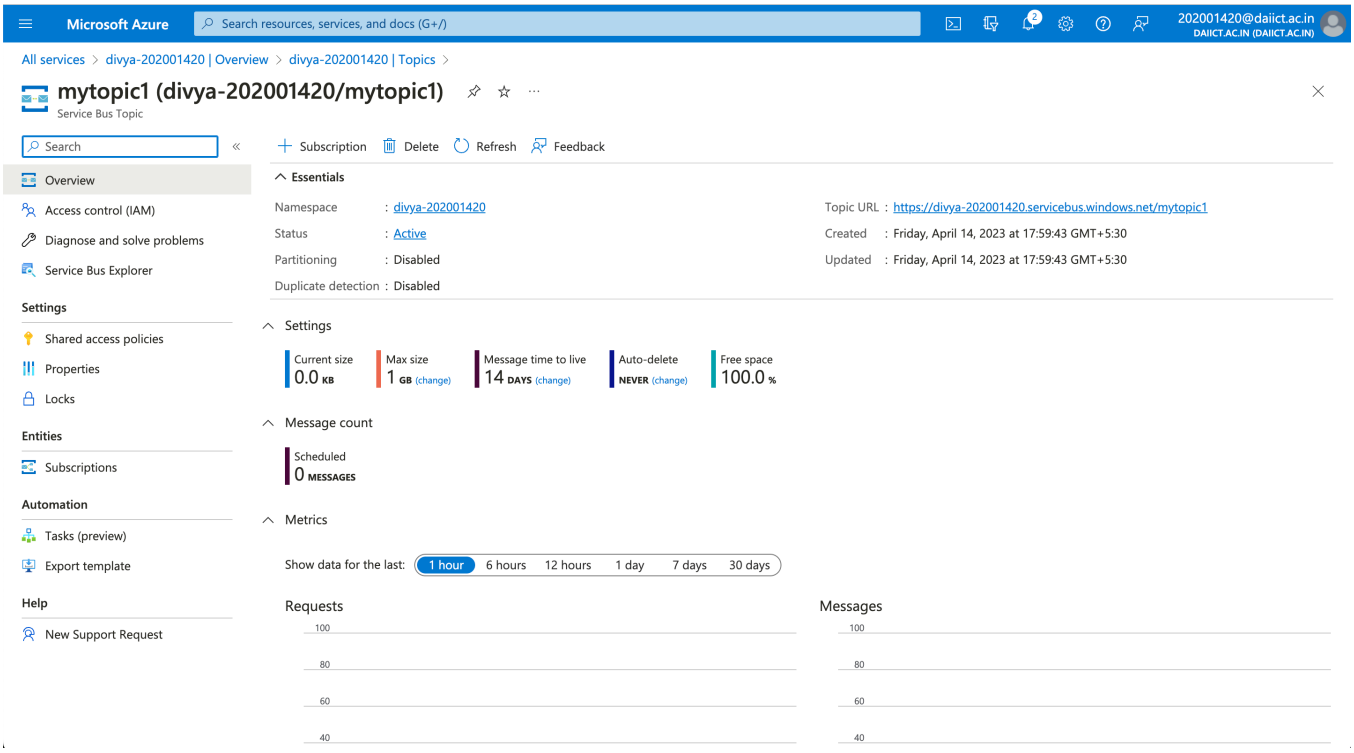
i. Create service Bus



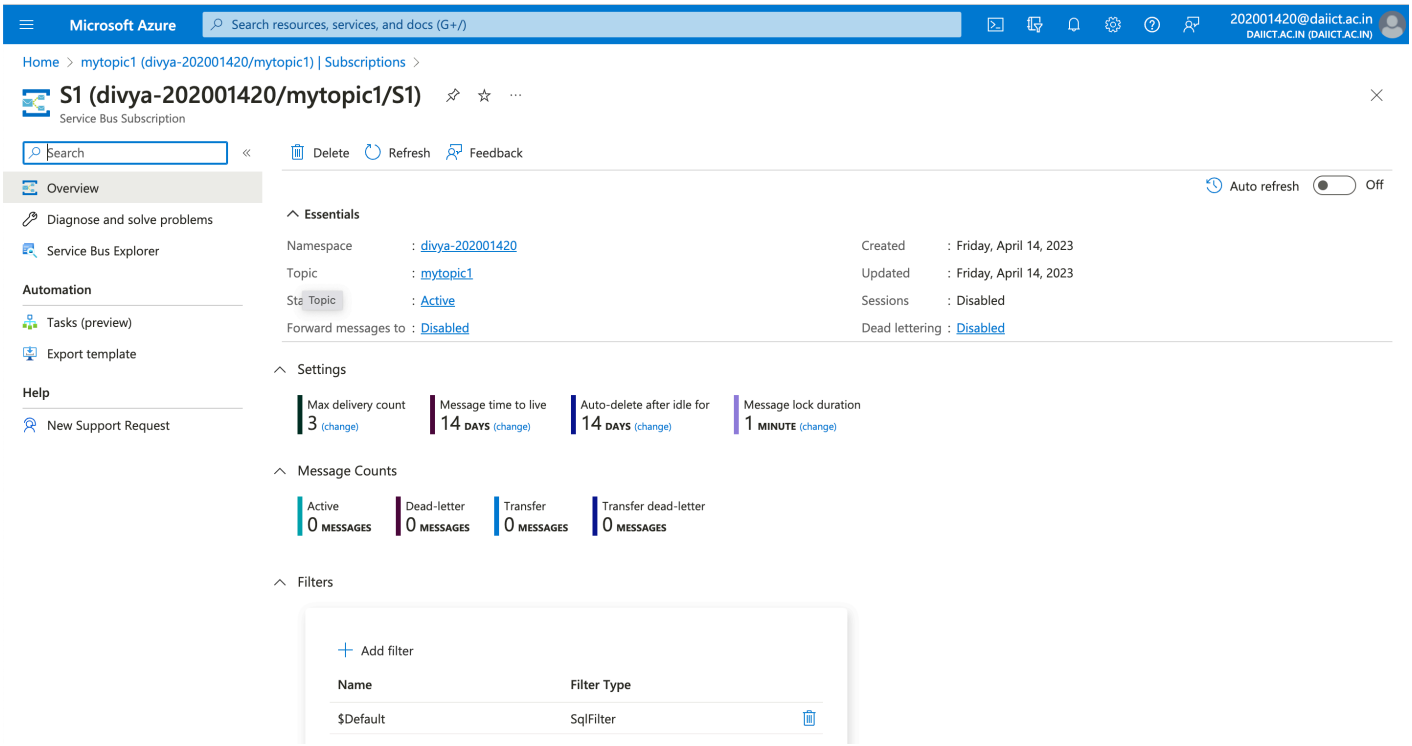
ii. Service Bus Dashboard



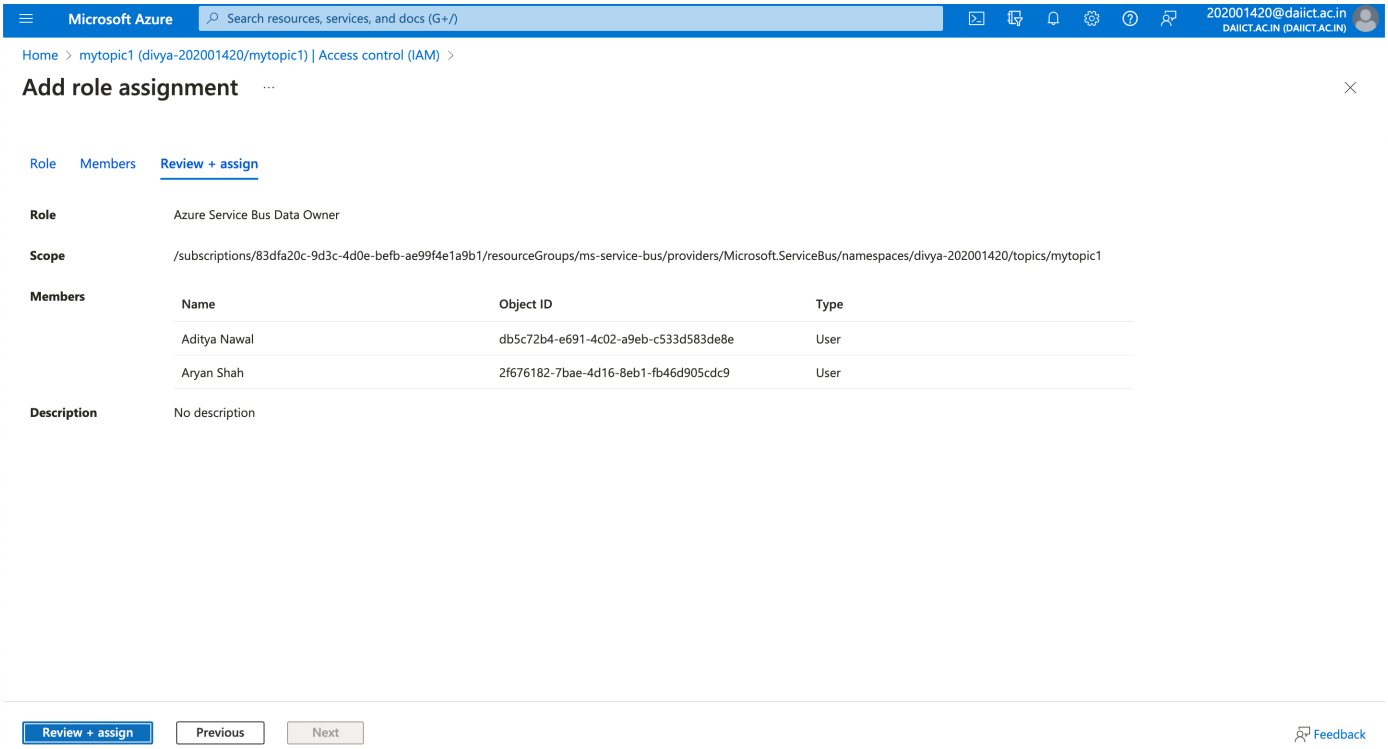
iii. Create New Topic in Service Bus



iv. Create new subscription in topic



v. Add Role Assignment



Send.py

```
import asyncio
from azure.servicebus.aio import ServiceBusClient
from azure.servicebus import ServiceBusMessage

NAMESPACE_CONNECTION_STR = "Endpoint=sb://divya-
202001420.servicebus.windows.net/;SharedAccessKeyName=newpolicy;SharedAcces
sKey=6Q1uwO/JyLevdQ7vF2z7jBYk14c0EspLy+ASbF9dlSQ=;EntityPath=mytopic1"
TOPIC_NAME = "mytopic1"


async def send_single_message(sender):
    # Create a Service Bus message
    message = ServiceBusMessage("Single Message")
    # send the message to the topic
    await sender.send_messages(message)
    print("Sent a single message")


async def send_a_list_of_messages(sender):
    # Create a list of messages
    messages = [ServiceBusMessage("Message in list") for _ in range(5)]
    # send the list of messages to the topic
    await sender.send_messages(messages)
```

```
    print("Sent a list of 5 messages")


async def send_batch_message(sender):
    # Create a batch of messages
    async with sender:
        batch_message = await sender.create_message_batch()
        for _ in range(10):
            try:
                # Add a message to the batch
                batch_message.add_message(ServiceBusMessage(
                    "Message inside a ServiceBusMessageBatch"))
            except ValueError:
                # ServiceBusMessageBatch object reaches max_size.
                # New ServiceBusMessageBatch object can be created here to
send more data.
                break
        # Send the batch of messages to the topic
        await sender.send_messages(batch_message)
    print("Sent a batch of 10 messages")


async def run():
    # create a Service Bus client using the connection string
    async with ServiceBusClient.from_connection_string(
            conn_str=NAMESPACE_CONNECTION_STR,
            logging_enable=True) as servicebus_client:
        # Get a Topic Sender object to send messages to the topic
        sender = servicebus_client.get_topic_sender(topic_name=TOPIC_NAME)
        async with sender:
            # Send one message
            await send_single_message(sender)
            # Send a list of messages
            await send_a_list_of_messages(sender)
            # Send a batch of messages
            await send_batch_message(sender)

asyncio.run(run())
print("Done sending messages")
print("-----------------------")
```

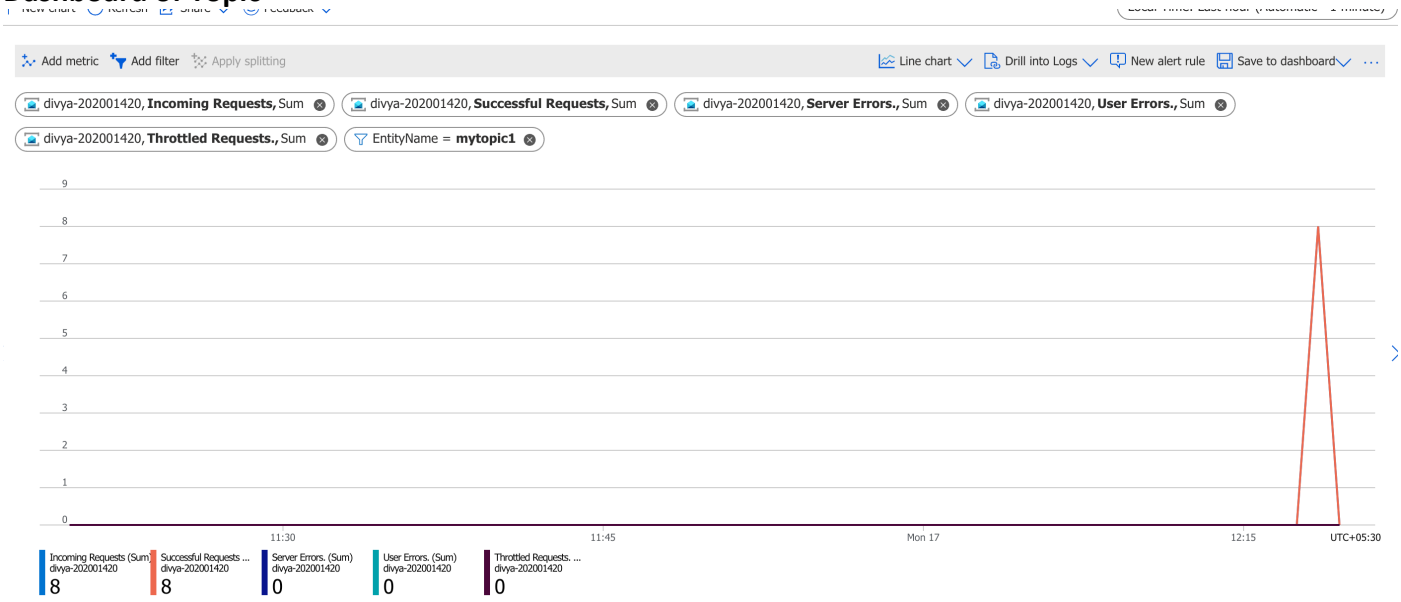
recv.py

```
import asyncio
from azure.servicebus.aio import ServiceBusClient

NAMESPACE_CONNECTION_STR = "Endpoint=sb://divya-
202001420.servicebus.windows.net/;SharedAccessKeyName=newpolicy;SharedAcces
sKey=6Q1uwO/JyLevdQ7vF2z7jBYk14c0EspLy+ASbF9dlSQ=;EntityPath=mytopic1"
SUBSCRIPTION_NAME = "S1"
TOPIC_NAME = "mytopic1"


async def run():
    # create a Service Bus client using the connection string
    async with ServiceBusClient.from_connection_string(
            conn_str=NAMESPACE_CONNECTION_STR,
            logging_enable=True) as servicebus_client:

        async with servicebus_client:
            # get the Subscription Receiver object for the subscription
            receiver = servicebus_client.get_subscription_receiver(
topic_name=TOPIC_NAME,
subscription_name= SUBSCRIPTION_NAME, max_wait_time=5)
            async with receiver:
                received_msgs = await
receiver.receive_messages(max_wait_time=5, max_message_count=20)
                for msg in received_msgs:
                    print("Received: " + str(msg))
                    # complete the message so that the message is removed
from the subscription
                    receiver.complete_message(msg)
asyncio.run(run())
```

Running Scripts

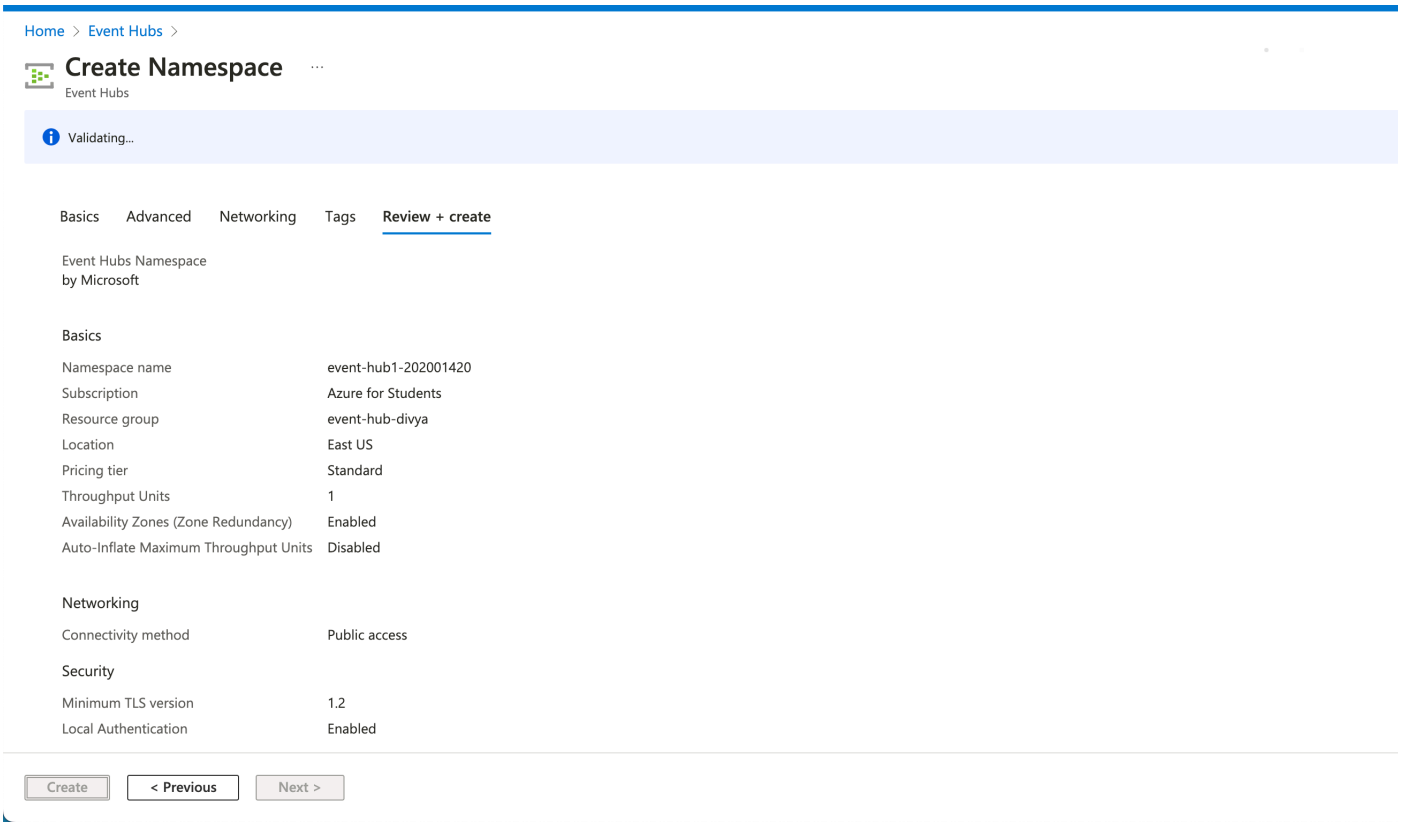
```
divya@Divyas-MacBook-Air ~ % python send.py & python recv.py

Sent a single message
Sent a list of 5 messages
Sent a batch of 10 messages
Done sending messages
----------------------
Received: Single Message
Received: Message in list
Received: Message in list
Received: Message in list
Received: Message in list
Received: Message in list
Received: Message inside a ServiceBusMessageBatch
Received: Message inside a ServiceBusMessageBatch
Received: Message inside a ServiceBusMessageBatch
Received: Message inside a ServiceBusMessageBatch
Received: Message inside a ServiceBusMessageBatch
Received: Message inside a ServiceBusMessageBatch
Received: Message inside a ServiceBusMessageBatch
Received: Message inside a ServiceBusMessageBatch
Received: Message inside a ServiceBusMessageBatch
```

Dashboard of Topic

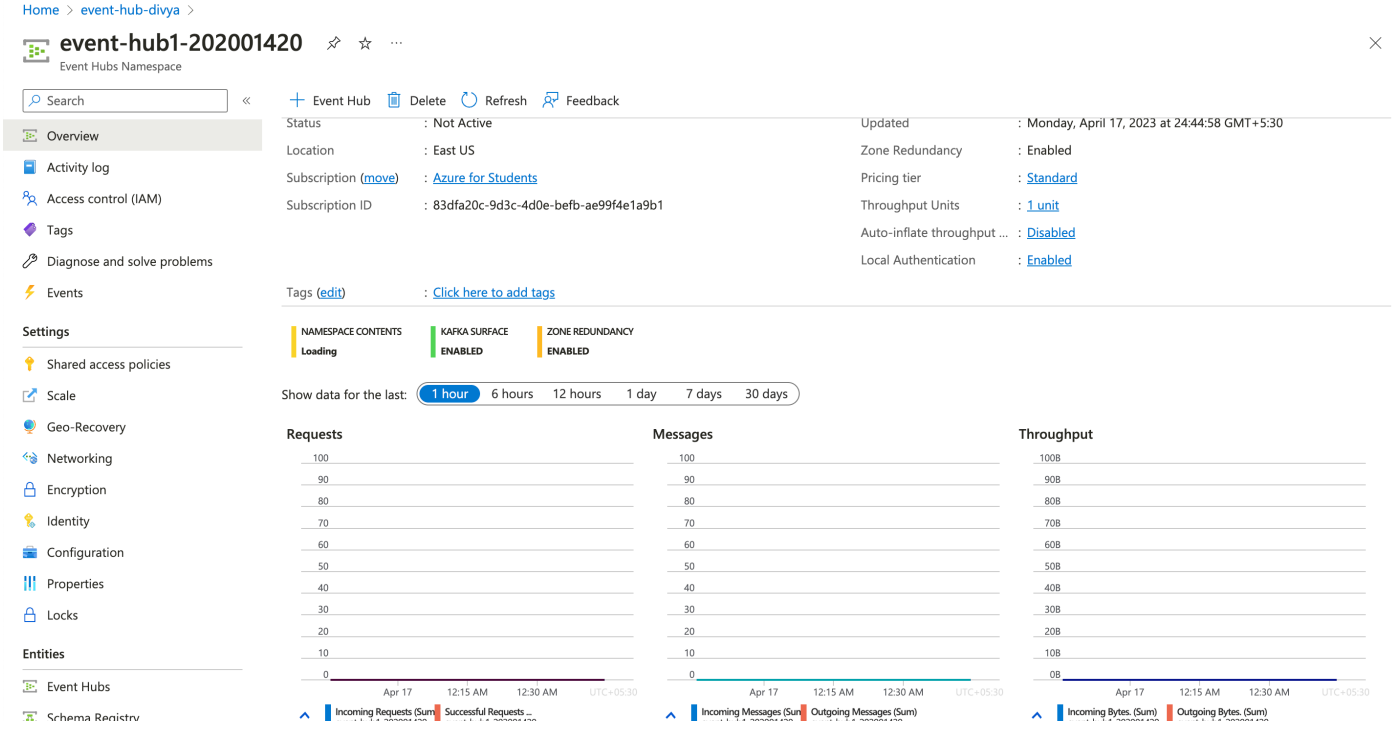


2. Send events to or receive events from event hubs by using Python

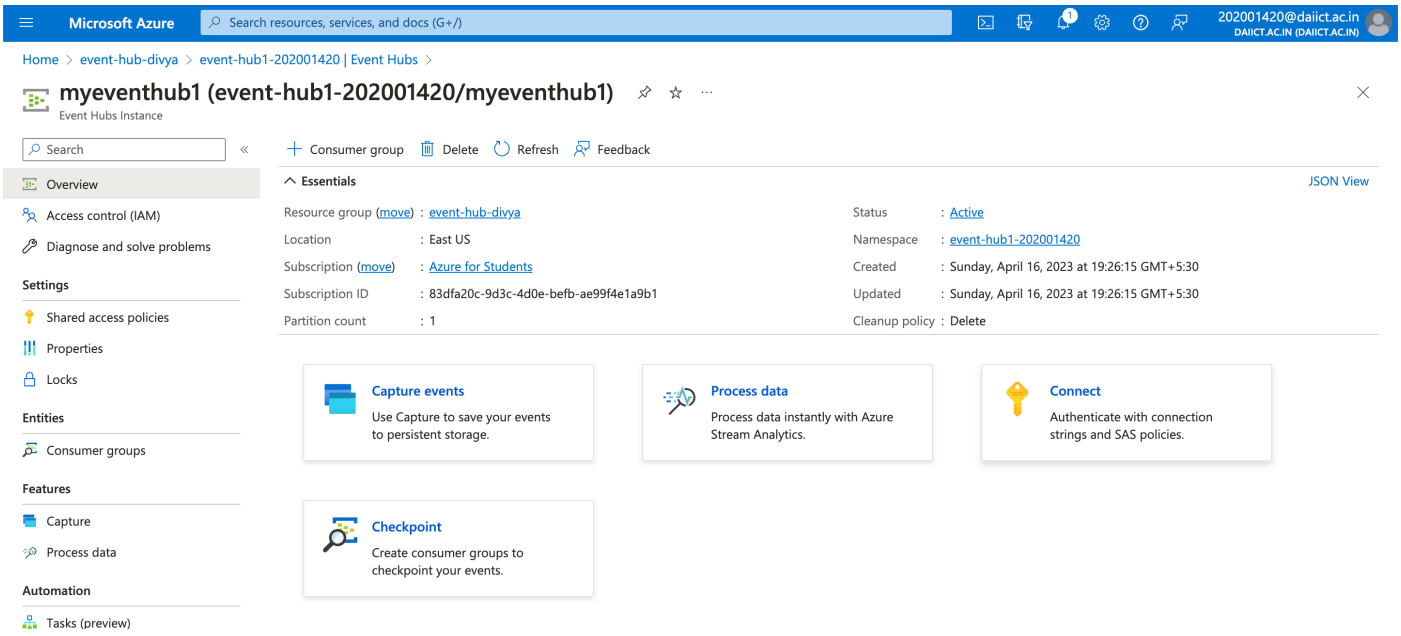
i. Create event-hub namespace



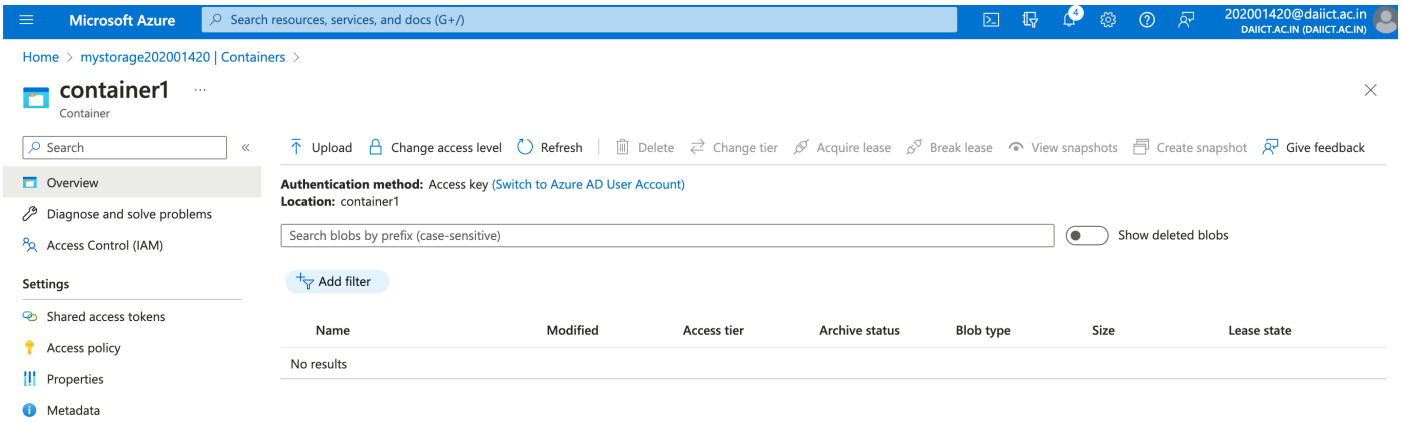
ii. Create Event-hub



iii. Event-hub details



iv. Container Details



Run.py

```
import asyncio

from azure.eventhub import EventData
from azure.eventhub.aio import EventHubProducerClient
from azure.identity import DefaultAzureCredential

EVENT_HUB_FULLY_QUALIFIED_NAMESPACE = "event-hub1-202001420"
EVENT_HUB_NAME = "myeventhub1"

credential = DefaultAzureCredential()
async def run():
    # Create a producer client to send messages to the event hub.
    # Specify a credential that has correct role assigned to access
    # event hubs namespace and the event hub name.
    producer = EventHubProducerClient(
        fully_qualified_namespace=EVENT_HUB_FULLY_QUALIFIED_NAMESPACE,
        eventhub_name=EVENT_HUB_NAME,
        credential=credential,
    )
    async with producer:
        # Create a batch.
        event_data_batch = await producer.create_batch()

        # Add events to the batch.
        event_data_batch.add(EventData("First event "))
        event_data_batch.add(EventData("Second event"))
        event_data_batch.add(EventData("Third event"))

        # Send the batch of events to the event hub.
        await producer.send_batch(event_data_batch)

        # Close credential when no longer needed.
        await credential.close()

asyncio.run(run())
```

Recv.py

```
from azure.identity.aio import DefaultAzureCredential
from azure.eventhub.extensions.checkpointstoreblobaio import (
    BlobCheckpointStore,
)
from azure.eventhub.aio import EventHubConsumerClient
import asyncio

from azure.eventhub import EventData
from azure.eventhub.aio import EventHubProducerClient
from azure.identity import DefaultAzureCredential

EVENT_HUB_FULLY_QUALIFIED_NAMESPACE = "event-hub1-202001420"
EVENT_HUB_NAME = "myeventhub1"


BLOB_STORAGE_ACCOUNT_URL =
"DefaultEndpointsProtocol=https;AccountName=mystorage202001420;AccountKey=S
Su5MB7UY1C7fClikzk4eT4zoGnuIqrSmMNRL2simnrOp6UdfWZl9bZm3EcZJJqh8VebJnYuFZ6G
+AStD912Ug==;EndpointSuffix=core.windows.net"
BLOB_CONTAINER_NAME = "container1"

credential = DefaultAzureCredential()


async def on_event(partition_context, event):
    # Print the event data.
    print(
        'Received the event: "{}" from the partition with ID: "{}"'.format(
            event.body_as_str(encoding="UTF-8"),
partition_context.partition_id
        )
    )

    # Update the checkpoint so that the program doesn't read the events
    # that it has already read when you run it next time.
    await partition_context.update_checkpoint(event)


async def main():
    # Create an Azure blob checkpoint store to store the checkpoints.
    checkpoint_store = BlobCheckpointStore(
        blob_account_url=BLOB_STORAGE_ACCOUNT_URL,
        container_name=BLOB_CONTAINER_NAME,
```

```
        credential=credential,
    )

    # Create a consumer client for the event hub.
    client = EventHubConsumerClient(
        fully_qualified_namespace=EVENT_HUB_FULLY_QUALIFIED_NAMESPACE,
        eventhub_name=EVENT_HUB_NAME,
        consumer_group="$Default",
        checkpoint_store=checkpoint_store,
        credential=credential,
    )
    async with client:
        # Call the receive method. Read from the beginning of the partition
        # (starting_position: "-1")
        await client.receive(on_event=on_event, starting_position="-1")

    # Close credential when no longer needed.
    await credential.close()

if __name__ == "__main__":
    # Run the main method.
    asyncio.run(main())
```

Running send.py

Running recv.py