Azure Bus Service

pub/sub POC

Antariksh Kalita

2023

Table of Contents

[Introduction 2](#_Toc142948438)

[Overview 2](#_Toc142948439)

[Azure Service Bus 2](#_Toc142948440)

[Key Concepts 2](#_Toc142948441)

[Topics and Subscriptions 2](#_Toc142948442)

[Queues 2](#_Toc142948443)

[Message Brokering 2](#_Toc142948444)

[In this Project we are using Topics and Subscription, lets dive into its details 3](#_Toc142948445)

[What are Service Bus topics and subscriptions? 3](#_Toc142948446)

[Event Publishing System: Overview 4](#_Toc142948447)

[TOPICS (for demonstration) 4](#_Toc142948448)

[Project High Level Design: 4](#_Toc142948449)

[Getting Started 5](#_Toc142948450)

[Cloud resource: 5](#_Toc142948451)

[Azure Service Bus: Resource Creation 5](#_Toc142948452)

[Running the Applications locally: 5](#_Toc142948453)

[Dependency 5](#_Toc142948454)

[Prerequisites: 5](#_Toc142948455)

[The publisher application: 5](#_Toc142948456)

[The Subscriber Application: 5](#_Toc142948457)

[Project Structure 6](#_Toc142948458)

[Publisher 8](#_Toc142948459)

[References: 9](#_Toc142948460)

# Introduction

Welcome to the User Manual for the Event Publishing System using Azure Service bus.

This comprehensive guide is designed to help you understand and utilize the functionalities of our innovative system for efficiently publishing and subscribing of the topic.

# Overview

## Azure Service Bus

Azure Service Bus is a cloud-based messaging service that provides reliable communication between applications and services. This document provides an overview of Azure Service Bus and how it can enhance your application's messaging capabilities.

# Key Concepts

## Topics and Subscriptions

Topics and subscriptions are fundamental components of Azure Service Bus. Understand how they work and how they enable effective communication between publishers and subscribers.

## Queues

Queues offer point-to-point communication and help decouple sender and receiver applications.

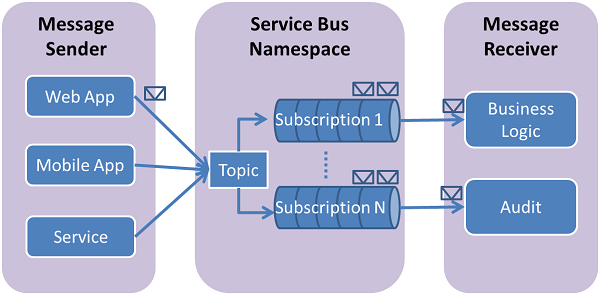
## Message Brokering

Learn about message brokers and how they facilitate asynchronous communication.

## In this Project we are using Topics and Subscription, lets dive into its details

## What are Service Bus topics and subscriptions?

Service Bus topics and subscriptions support a *publish/subscribe* messaging communication model. When using topics and subscriptions, components of a distributed application do not communicate directly with each other; instead they exchange messages via a topic, which acts as an intermediary.



In contrast with Service Bus queues, in which each message is processed by a single consumer, topics and subscriptions provide a one-to-many form of communication, using a publish/subscribe pattern. It is possible to register multiple subscriptions to a topic. When a message is sent to a topic, it is then made available to each subscription to handle/process independently. A subscription to a topic resembles a virtual queue that receives copies of the messages that were sent to the topic. You can optionally register filter rules for a topic on a per-subscription basis, which allows you to filter or restrict which messages to a topic are received by which topic subscriptions.

Service Bus topics and subscriptions enable you to scale to process a large number of messages across a large number of users and applications.

# Event Publishing System: Overview

Welcome to the documentation for the Flask application, a platform that allows users to subscribe to topics, publish articles. This document provides an overview of the application's functionality and guides you through the setup and usage.

This project contains two services independently interacting with Azure Service Bus.

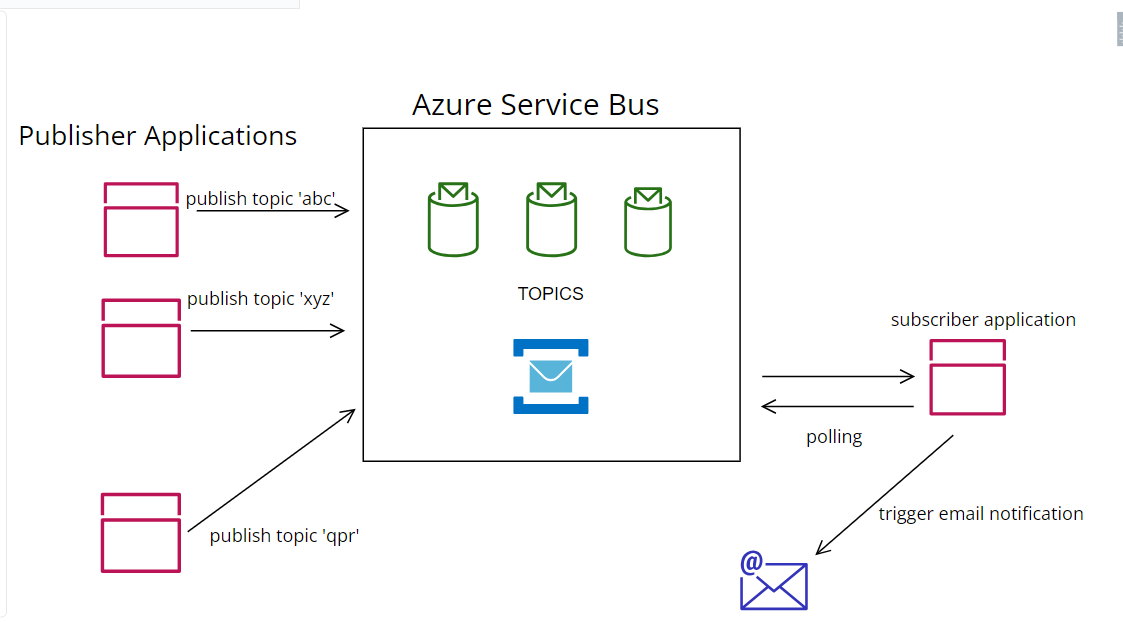
In this project Azure Service Bus’s Topic’s publisher and subscriber feature is being used.

1. Publisher Application: A flask application that allows users to:
   1. User registration and login
   2. Subscription to multiple topics
   3. Publishing articles to topics
2. Subscriber Application: a python script that triggers email notification whenever an event is published not a topic.

# TOPICS (for demonstration)

* Esports
* Event
* Marketing

# Project High Level Design:



# Getting Started

Guide users through the steps to set up and run the Flask application locally:

## Cloud resource:

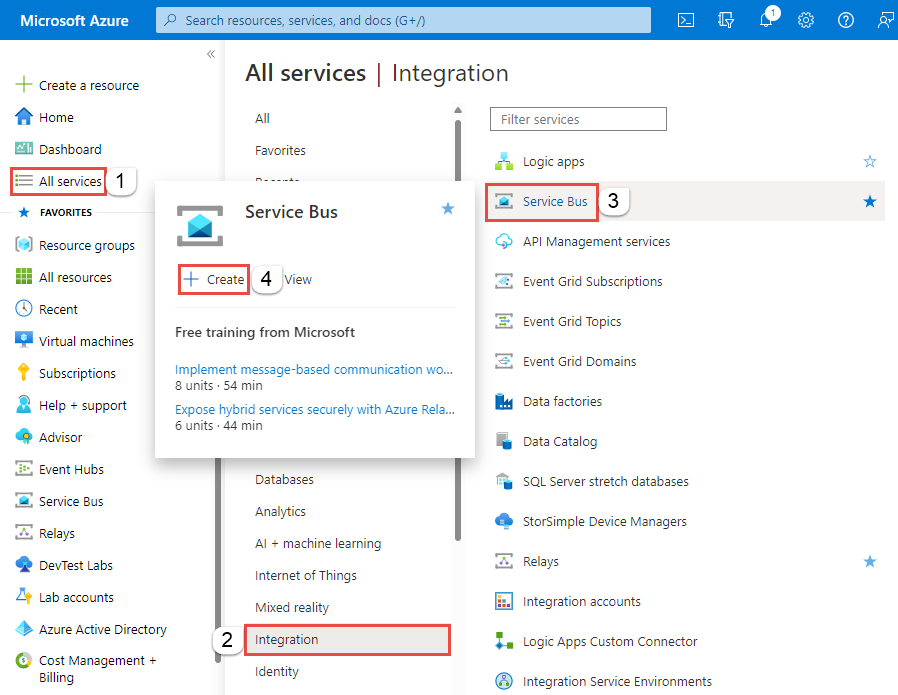
* Azure Service Bus (standard tier and above)

### Azure Service Bus: Resource Creation

#### Create a namespace in the Azure portal

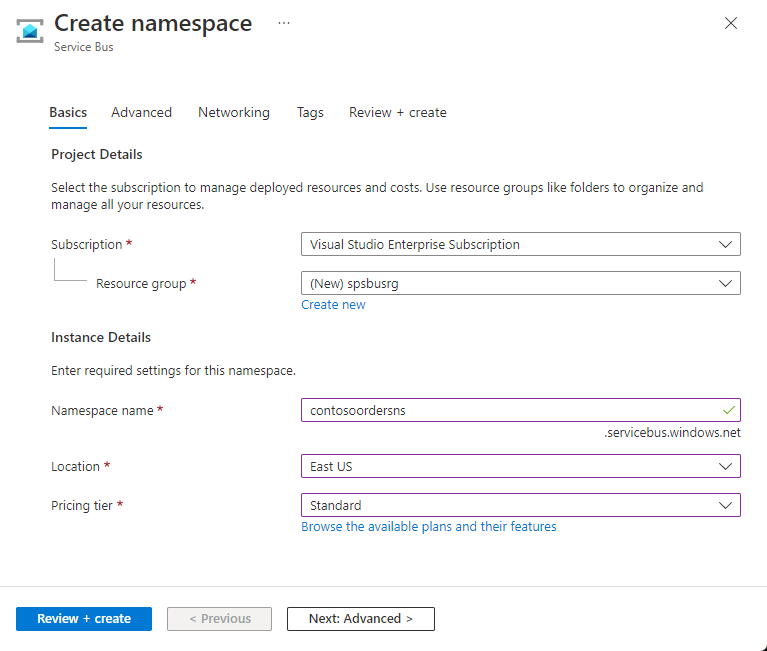
To begin using Service Bus messaging entities in Azure, you must first create a namespace with a name that is unique across Azure. A namespace provides a scoping container for Service Bus resources within your application.

To create a namespace:

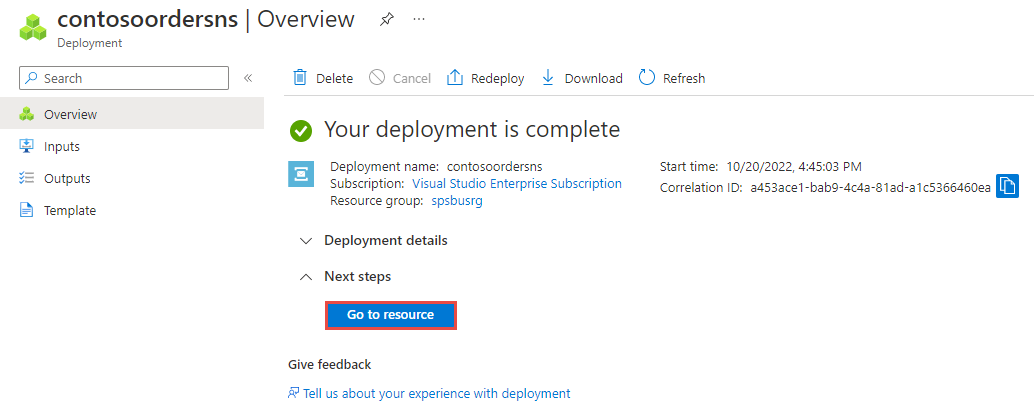
1. Sign in to the [Azure portal](https://portal.azure.com/).
2. In the left navigation pane of the portal, select **All services**, select **Integration** from the list of categories, hover the mouse over **Service Bus**, and then select **Create** on the Service Bus tile.
3. In the **Basics** tag of the **Create namespace** page, follow these steps:
   1. For **Subscription**, choose an Azure subscription in which to create the namespace.
   2. For **Resource group**, choose an existing resource group in which the namespace will live, or create a new one.
   3. Enter a **name for the namespace**. The namespace name should adhere to the following naming conventions:
      1. The name must be unique across Azure. The system immediately checks to see if the name is available.
      2. The name length is at least 6 and at most 50 characters.
      3. The name can contain only letters, numbers, hyphens “-“.
      4. The name must start with a letter and end with a letter or number.
      5. The name doesn't end with “-sb“ or “-mgmt“.
   4. For **Location**, choose the region in which your namespace should be hosted.
   5. For **Pricing tier**, select the pricing tier (Basic, Standard, or Premium) for the namespace. For this, select **Standard**.

**Important**

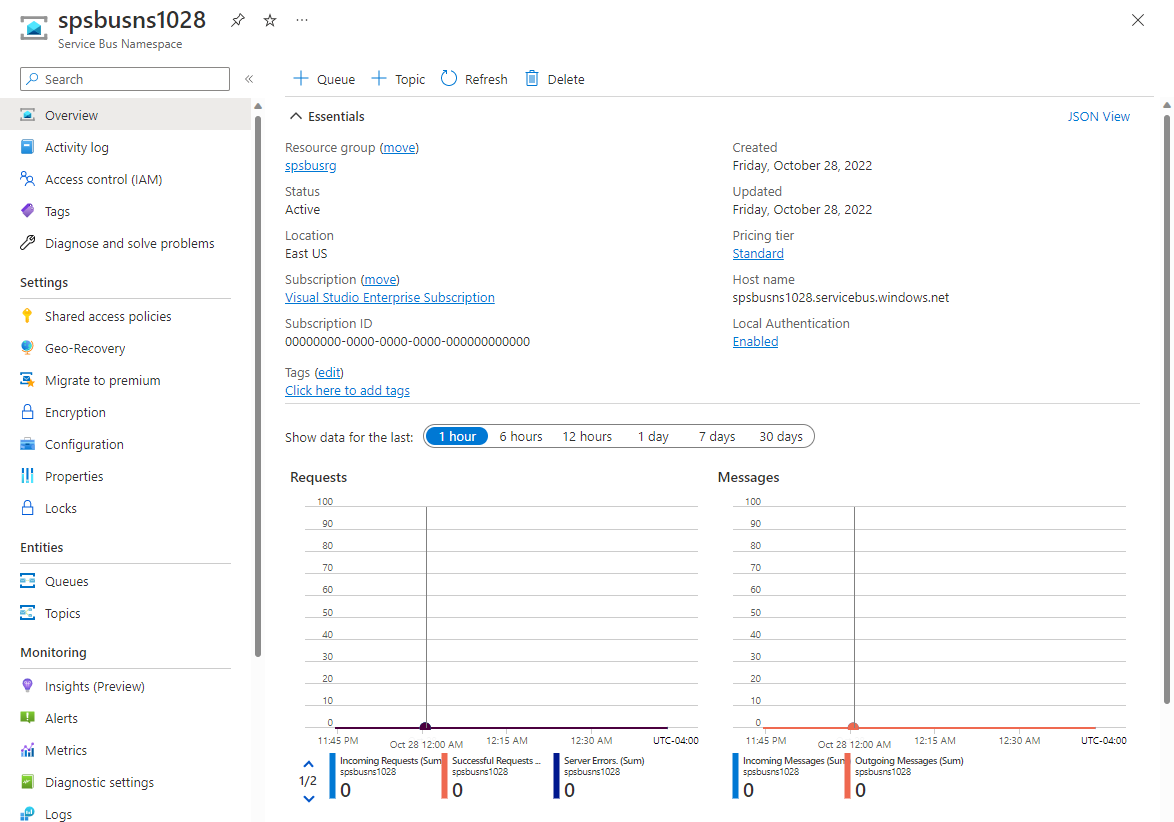
**If you want to use**[**topics and subscriptions**](https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions#topics-and-subscriptions)**, choose either Standard or Premium. Topics/subscriptions aren't supported in the Basic pricing tier.**

* 1. Select **Review + create** at the bottom of the page.
  2. On the **Review + create** page, review settings, and select **Create**.

1. Once the deployment of the resource is successful, select **Go to resource** on the deployment page.

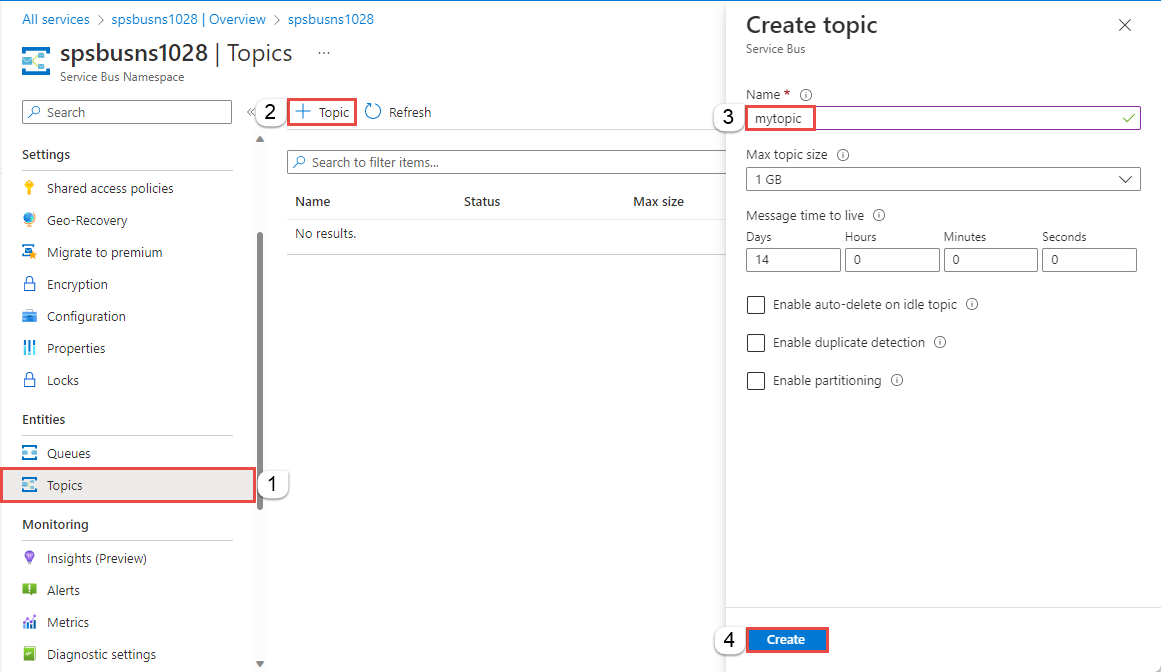


1. You see the home page for your service bus namespace.

[](https://learn.microsoft.com/en-us/azure/service-bus-messaging/includes/media/service-bus-create-namespace-portal/service-bus-namespace-home-page.png#lightbox)

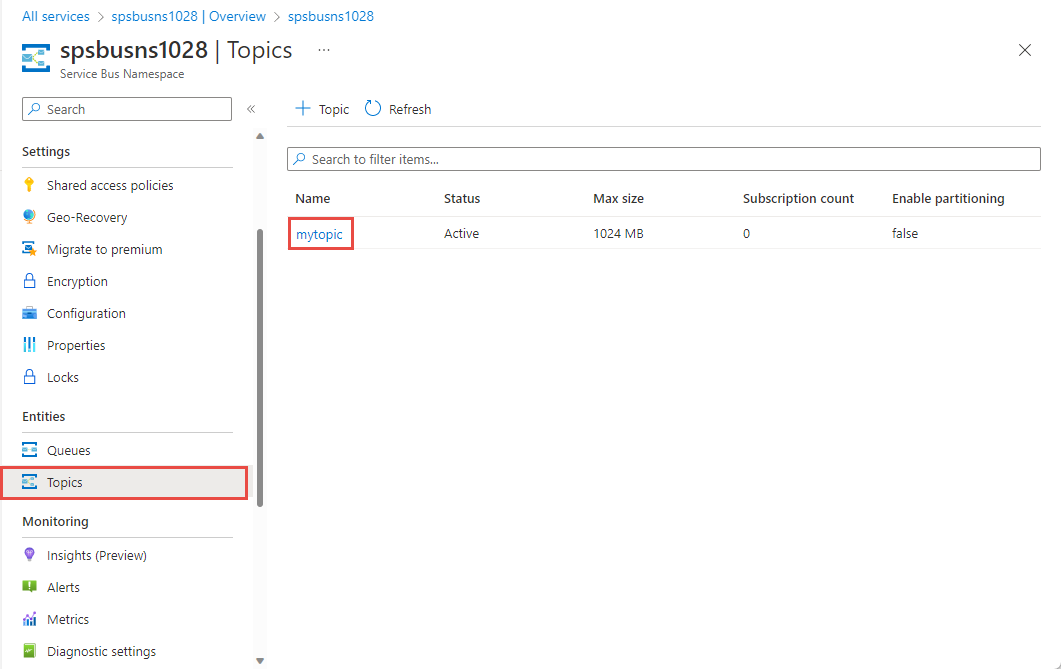
#### Create a topic using the Azure portal

1. On the **Service Bus Namespace** page, select **Topics** on the left menu.
2. Select **+ Topic** on the toolbar.
3. Enter a **name** for the topic. Leave the other options with their default values.
4. Select **Create**.
5. Create the required number of topics as per the use case.

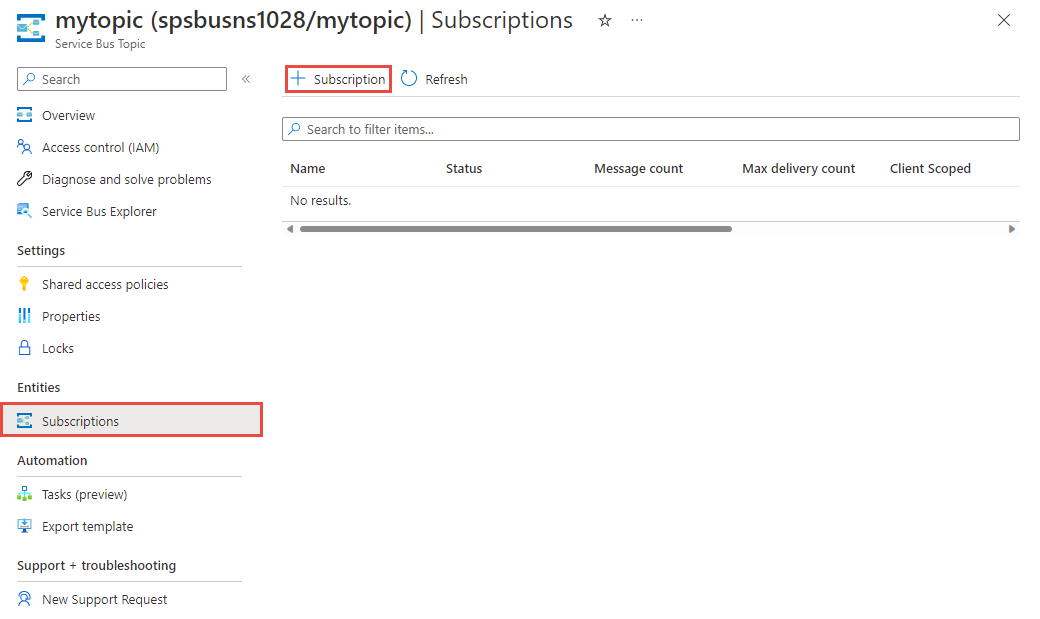
[](https://learn.microsoft.com/en-us/azure/service-bus-messaging/includes/media/service-bus-create-topics-subscriptions-portal/create-topic.png#lightbox)

#### Create subscriptions to the topic

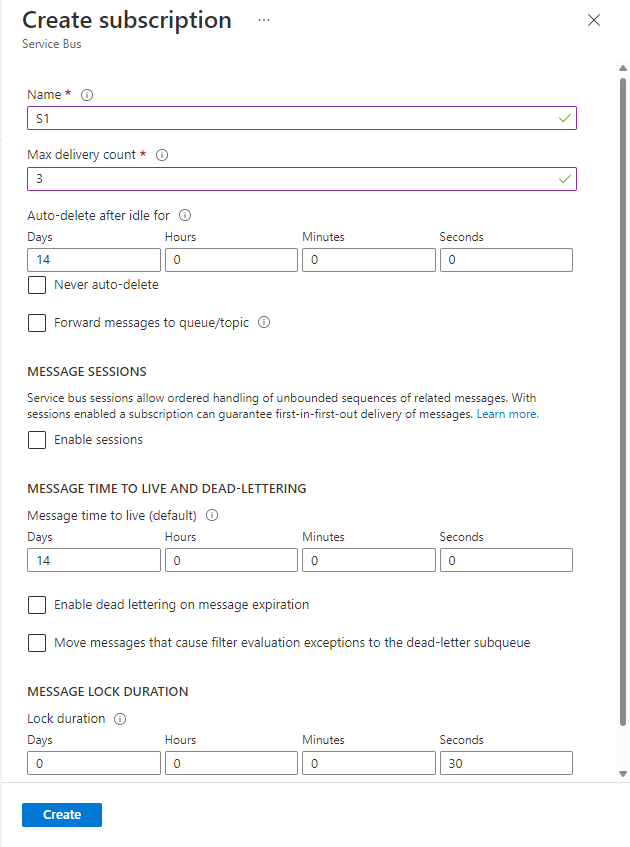
1. Select the **topic** that you created in the previous section.

[](https://learn.microsoft.com/en-us/azure/service-bus-messaging/includes/media/service-bus-create-topics-subscriptions-portal/select-topic.png#lightbox)

1. On the **Service Bus Topic** page, select **Subscriptions** from the left menu, and then select **+ Subscription** on the toolbar.



1. On the **Create subscription** page, follow these steps:
   1. Enter **event** for **name** of the subscription.
   2. Enter **1** for **Max delivery count**.
   3. Then, select **Create** to create the subscription.



1. Repeat the previous step twice to create subscriptions named **Marketing** and **Esports**.

**Connect the resource using the shared access credentials available on the left panel.**

## Running the Applications locally:

### Dependency

* Create the Azure Service Bus resource.

### Prerequisites:

Python 3.8 and above.

#### Publisher Application:

* + alembic==1.11.2
  + azure-core==1.29.1
  + azure-servicebus==7.11.1
  + blinker==1.6.2
  + certifi==2023.7.22
  + charset-normalizer==3.2.0
  + click==8.1.6
  + colorama==0.4.6
  + Flask==2.3.2
  + Flask-Login==0.6.2
  + Flask-Migrate==4.0.4
  + Flask-SQLAlchemy==3.0.5
  + Flask-WTF==1.1.1
  + greenlet==2.0.2
  + idna==3.4
  + isodate==0.6.1
  + itsdangerous==2.1.2
  + Jinja2==3.1.2
  + Mako==1.2.4
  + MarkupSafe==2.1.3
  + requests==2.31.0
  + six==1.16.0
  + SQLAlchemy==2.0.19
  + typing\_extensions==4.7.1
  + urllib3==2.0.4
  + Werkzeug==2.3.6
  + WTForms==3.0.1
  + gunicorn==20.1.0

#### Subscriber application:

* azure-core==1.29.1
* azure-servicebus==7.11.1
* certifi==2023.7.22

### The publisher application:

* Navigate to the publisher application source code directory and open command prompt.
* Install the dependencies by running the command ‘pip install -r requirements.txt” in the command prompt.
* Start the application by executing command “python app.py” in the command prompt.

### The Subscriber Application:

* Navigate to the publisher application source code directory and open command prompt.
* Install the dependencies by running the command ‘pip install -r requirements.txt” in the command prompt.
* Start the application by executing command “python subscriber.py” in the command prompt.

# Project Structure

+---publisher

| | app.py

| | Dockerfile

| | extention.py

| | requirements.txt

| | settings.py

| | topics.json

| |

| +---forms

| | | login\_form.py

| | | publish\_form.py

| | | registration\_form.py

| | | \_\_init\_\_.py

| |

| +---instance

| | db.sqlite3

| |

| +---log

| | | log.py

| | | \_\_init\_\_.py

| +---models

| | | TopicSubscriptionModel.py

| | | UserModel.py

| | | \_\_init\_\_.py

| |

| +---routes

| | | dashboard\_route.py

| | | login\_route.py

| | | publisher\_route.py

| | | registration\_route.py

| | | root.py

| | | \_\_init\_\_.py

| |

| +---services

| | | publish\_utils.py

| | | utils.py

| | | \_\_init\_\_.py

| | |

| +---static

| | \---styles

| | main.css

| |

| +---templates

| | base.html

| | dashboard.html

| | login.html

| | navbar.html

| | publish.html

| | register\_user.html

|

+---subscriber\_stand\_alone

| | Dockerfile

| | log.py

| | requirements.txt

| | send\_email.py

| | setting.py

| | subscriber.py

| | \_\_init\_\_.py

## Publisher

This directory contains files related to the main Flask application that handles user registration, topic subscription, article publishing, and more.

* **app.py**: The main Flask application file.
* **Dockerfile**: A Docker configuration file for containerization.
* **extention.py**: Extension configuration file.
* **requirements.txt**: List of Python dependencies for the application.
* **settings.py**: Application configuration settings.
* **topics.json**: JSON file containing topic data.

**forms**

This subdirectory contains form classes for user registration, login, and publishing articles.

* **login\_form.py**: Form class for user login.
* **publish\_form.py**: Form class for publishing articles.
* **registration\_form.py**: Form class for user registration.
* **\_\_init\_\_.py**: An empty file to mark the directory as a Python package.

**instance**

This directory could potentially contain instance-specific configuration files, including databases.

* **db.sqlite3**: SQLite database file.

**log**

This subdirectory contains files related to logging.

* **log.py**: Logging configuration file.
* **\_\_init\_\_.py**: An empty file to mark the directory as a Python package.

**models**

This subdirectory contains model classes that define the database schema.

* **TopicSubscriptionModel.py**: Model class for topic subscription data.
* **UserModel.py**: Model class for user data.
* **\_\_init\_\_.py**: An empty file to mark the directory as a Python package.

**Note**: **UserModel and TopicSubscriptionModel have 1 to many relationships**

**routes**

This subdirectory contains route files for different application functionalities.

* **dashboard\_route.py**: Route handling dashboard functionality.
* **login\_route.py**: Route handling user login.
* **publisher\_route.py**: Route handling article publishing.
* **registration\_route.py**: Route handling user registration.
* **root.py**: Root route handling.
* **\_\_init\_\_.py**: An empty file to mark the directory as a Python package.

**services**

This subdirectory contains utility functions and services.

* **publish\_utils.py**: Utility functions for publishing articles.
* **utils.py**: General utility functions.
* **\_\_init\_\_.py**: An empty file to mark the directory as a Python package.

**static**

This directory contains static assets like CSS, JavaScript, and images.

* styles: Subdirectory for CSS styles.
* main.css: Main CSS style file.

**templates**

This directory contains Jinja2 templates for HTML pages.

* base.html: Base template that other templates extend.
* dashboard.html: Template for the dashboard page.
* login.html: Template for the login page.
* navbar.html: Template for the navigation bar.
* publish.html: Template for the article publishing page.
* register\_user.html: Template for the user registration page.

# Deployment:

## Azure App Services:

* <https://learn.microsoft.com/en-us/azure/app-service/quickstart-python?tabs=flask%2Cwindows%2Cazure-cli%2Clocal-git-deploy%2Cdeploy-instructions-azportal%2Cterminal-bash%2Cdeploy-instructions-zip-azcli>

EC2 Docker container:

* Clone the code from git repository or pull the container image from the any registry.

## Step 1: Set Up AWS EC2 Instance:

* Launch an EC2 instance with the desired AMI (e.g., Amazon Linux).
* Configure security groups to allow incoming traffic on the necessary ports (e.g., 80 for HTTP).

## Step 2: Connect to Your EC2 Instance:

* Use SSH to connect to your EC2 instance from your local machine: ssh -i path/to/your/key.pem ec2-user@your-instance-ip.

## Step 3: Install Docker:

* Update package index: sudo yum update -y.
* Install Docker: sudo yum install -y docker.

## Step 4: Configure Docker to Start on Boot:

* Start Docker service: sudo service docker start.
* Enter sudo chmod 666 /var/run/docker.sock
* Enable Docker to start on boot: sudo chkconfig docker on.

## Step 5: Build Docker Image:

Build the Docker image: docker build -t your-app-name .

## Step 6: Run Docker Container:

Run the Docker container: docker run -d -p 80:80 your-app-name

## Step 7: Access Your App:

Access your Flask app via your EC2 instance's public IP: http://your-instance-ip

Optional Steps:

**Domain Configuration**: If you have a custom domain, configure it to point to your EC2 instance's IP address using Route 53.

**HTTPS**: For HTTPS support, consider setting up an SSL certificate using services like AWS Certificate Manager or Let's Encrypt.

**Logging and Monitoring**: Set up monitoring and logging solutions like CloudWatch to monitor your EC2 instance and Docker container's health.

**Scaling**: If your app experiences increased traffic, you might consider using Auto Scaling to manage multiple EC2 instances.

# References:

* <https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview>
* <https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-quickstart-topics-subscriptions-portal>
* <https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-quickstart-topics-subscriptions-portal>