QUESTION 82
Case Study 7
Policy service
You develop and deploy a stateful ASP.NET Core 2.1 web application named Policy service to an Azure App Service Web App. The application reacts to events from Azure Event Grid and performs policy actions based on those events. The application must include the Event Grid Event ID field in all Application Insights telemetry.
Policy service must use Application Insights to automatically scale with the number of policy actions that it is performing.
Other
Anomaly detection service
You have an anomaly detection service that analyzes log infrmation for anomalies. It is implemented as an Azure Machine Learning model. The model is deployed as a web service.
If an anomaly is detected, an Azure Function that emails administrators is called by using an HTTP WebHook.
Health monitoring
All web applications and services have health monitoring at the /health service endpoint.
Issues
Policy loss
When you deploy Policy service, policies may not be applied if they were in the process of being applied during the deployment.
Performance issue
When under heavy load, the anomaly detection service undergoes slowdowns and rejects connections.
Notification latency
Users report that anomaly detection emails can sometimes arrive several minutes after an anomaly is detected.
Policies
Log policy
All Azure App Service Web Apps must write logs to Azure Blob storage. All log files should be saved to a container named logdrop. Logs must remain in the container for 15 days.
Authentication events
Authentication events are used to monitor users signing in and signing out. All authentication events must be processed by Policy service. Sign outs must be processed as quickly as possible.
PolicyLib
You have a shared library named PolicyLib that contains functionality common to all ASP.NET Core web services and applications. The PolicyLib library must:
‧ Exclude non-user actions from Application Insights telemetry.
‧ Provide methods that allow a web service to scale itself.
‧ Ensure that scaling actions do not disrupt application usage.
Relevant portions of the app files are shown below. Line numbers are included for reference only and include a two-character prefix that denotes the specific file to which they belong.

```
EventGridController.cs
EG01 public class EventGridController : Controller
EG02 {
EG03    public static AsyncLocal<string> EventId = new AsyncLocal<string>();
EG04    public IactionResult Process([FromBody] string eventsJson)
EG05    {
EG06      var events = JArray.Parse(eventsJson);
EG07
EG08      foreach (var @event in events)
EG09      {
EG10        EventId.Value = @event["id"].ToString();
EG11        if (@event["topic"].ToString(). Contains("providers/Microsoft.Storage"))
EG12        {
EG13        SendToAnomalyDetection Service(@event["data"]["url"].ToString());
EG14          }
EG15
EG16        {
EG17        EnsureLogging(@event["subject"].ToString());
EG18        }
EG19      }
EG20      return null;
EG21    }
EG22    private void EnsureLogging(string resource)
EG23    {
EG24      . . .
EG25    }
EG26    private async Task SendToAnomalyDetectionService(string uri)
EG27    {
EG28    var content = GetLogData(uri)
EG29    var scoreRequest = new
EG30    {
EG31      Inputs = new Dictionary<string, List<Dictionary<string, string>>>()
EG32      {
EG33        {
EG34          "input1",
EG35        new List<Dictionary<string, string>>()
EG36        {
EG37          new Dictionary<string, string>()
EG38          {
EG39          {
EG40            "logcontent", content
EG41          }
EG42        }
EG43          }
EG44        },
EG45      },
EG46    GlobalParameters = new Dictionary<string, string>() ( )
EG47    };
EG48    var result = await (new HttpClient()).PostAsJsonAsync(" …", scoreRequest);
EG49    var rawModelResult = await result.Content.ReadAsStringAsync();
EG50    var modelResult = JObject.Parse(rawModelResult);
EG51 if (modelResult ["notify"].HasValues)
EG52    {
EG53    . . .
EG54    }
EG55 }
EG56 private (string name, string resourceGroup) ParseResourceId(string resourceId)
EG57 {
EG58    . . .
EG59 }
EG60 private string GetLogData(string uri)
EG61 {
EG62    . . .
EG63 }
EG64    static string BlobStoreAccountSAS (string containerName)
EG65    {
EG66    . . .
EG67    }
EG68 }
```

Relevant portions of the app files are shown below. Line numbers are included for reference only and include a two-character prefix that denotes the specific file to which they belong.

```
LoginEvent.cs
LE01 public class LoginEvent
LE02 {
LE03
LE04 public string subject { get ; set ; }
LE05 public DateTime eventTime { get ; set ; }
LE06 public Dictionary<string, string> data { get; set; }
LE07 public string Serialize()
LE08 {
LE09     return JsonConvert.SerializeObject(this);
LE10 }
LE11 }
```

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.
After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.
You need to ensure that authentication events are triggered and processed according to the policy.
Solution: Ensure that signout events have a subject prefix. Create an Azure Event Grid subscription that uses the subjectBeginsWith filter.
Does the solution meet the goal?

 A. Yes
 B. No

Answer: A
Explanation:
Scenario: Authentication events are used to monitor users signing in and signing out. All authentication events must be processed by Policy service.
Sign outs must be processed as quickly as possible.
References:
https://docs.microsoft.com/en-us/azure/event-grid/subscription-creation-schema

QUESTION 83
Case Study 7
Policy service
You develop and deploy a stateful ASP.NET Core 2.1 web application named Policy service to an Azure App Service Web App. The application reacts to events from Azure Event Grid and performs policy actions based on those events.
The application must include the Event Grid Event ID field in all Application Insights telemetry.
Policy service must use Application Insights to automatically scale with the number of policy actions that it is performing.
Other
Anomaly detection service
You have an anomaly detection service that analyzes log infrmation for anomalies. It is implemented as an Azure Machine Learning model. The model is deployed as a web service.
If an anomaly is detected, an Azure Function that emails administrators is called by using an HTTP WebHook.
Health monitoring
All web applications and services have health monitoring at the /health service endpoint.
Issues
Policy loss
When you deploy Policy service, policies may not be applied if they were in the process of being applied during the deployment.
Performance issue
When under heavy load, the anomaly detection service undergoes slowdowns and rejects connections.

Notification latency

Users report that anomaly detection emails can sometimes arrive several minutes after an anomaly is detected.

Policies

Log policy

All Azure App Service Web Apps must write logs to Azure Blob storage. All log files should be saved to a container named logdrop. Logs must remain in the container for 15 days.

Authentication events

Authentication events are used to monitor users signing in and signing out. All authentication events must be processed by Policy service. Sign outs must be processed as quickly as possible.

PolicyLib

You have a shared library named PolicyLib that contains functionality common to all ASP.NET Core web services and applications. The PolicyLib library must:

· Exclude non-user actions from Application Insights telemetry.

· Provide methods that allow a web service to scale itself.

· Ensure that scaling actions do not disrupt application usage.

Relevant portions of the app files are shown below. Line numbers are included for reference only and include a two-character prefix that denotes the specific file to which they belong.

```
EventGridController.cs
EG01 public class EventGridController : Controller
EG02 {
EG03    public static AsyncLocal<string> EventId = new AsyncLocal<string>();
EG04    public IactionResult Process([FromBody] string eventsJson)
EG05    {
EG06      var events = JArray.Parse(eventsJson);
EG07
EG08      foreach (var @event in events)
EG09      {
EG10        EventId.Value = @event["id"].ToString();
EG11        if (@event["topic"].ToString(). Contains("providers/Microsoft.Storage"))
EG12        {
EG13      SendToAnomalyDetection Service(@event["data"]["url"].ToString());
EG14          }
EG15
EG16        {
EG17      EnsureLogging(@event["subject"].ToString());
EG18        }
EG19      }
EG20      return null;
EG21    }
EG22    private void EnsureLogging(string resource)
EG23    {
EG24      . . .
EG25    }
EG26    private async Task SendToAnomalyDetectionService(string uri)
EG27    {
EG28    var content = GetLogData(uri)
EG29    var scoreRequest = new
EG30    {
EG31      Inputs = new Dictionary<string, List<Dictionary<string, string>>>()
EG32      {
EG33        {
EG34          "input1",
EG35        new List<Dictionary<string, string>>()
EG36        {
EG37          new Dictionary<string, string>()
EG38          {
EG39            {
EG40              "logcontent", content
EG41            }
EG42          }
EG43          }
EG44        },
EG45      },
EG46    GlobalParameters = new Dictionary<string, string>() { }
EG47    };
EG48    var result = await (new HttpClient()).PostAsJsonAsync(" …", scoreRequest);
EG49    var rawModelResult = await result.Content.ReadAsStringAsync();
EG50    var modelResult = JObject.Parse(rawModelResult);
EG51 if (modelResult ["notify"].HasValues)
EG52    {
EG53      . . .
EG54      }
EG55 }
EG56 private (string name, string resourceGroup) ParseResourceId(string resourceId)
EG57 {
EG58      . . .
EG59 }
EG60 private string GetLogData(string uri)
EG61 {
EG62      . . .
EG63 }
EG64    static string BlobStoreAccountSAS (string containerName)
EG65    {
EG66      . . .
EG67 }
EG68 }
```

Relevant portions of the app files are shown below. Line numbers are included for reference only and include a two-character prefix that denotes the specific file to which they belong.

```
LoginEvent.cs
LE01 public class LoginEvent
LE02 {
LE03
LE04 public string subject { get ; set ; }
LE05 public DateTime eventTime { get ; set ; }
LE06 public Dictionary<string, string> data { get; set; }
LE07 public string Serialize()
LE08 {
LE09     return JsonConvert.SerializeObject(this);
LE10 }
LE11 }
```

Drag and Drop Question
You need to implement telemetry for non-user actions.
How should you complete the Filter class? To answer, drag the appropriate code segments to the correct locations.
Each code segment may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content.
NOTE: Each correct selection is worth one point.

Code segments

```
/health
/status
RequestTelemetry
PageViewTelemetry
ITelemetryProcessor
ITelemetryInitializer
```

Answer Area

```
public class filter :[          ]
{
    private readonly [          ] _next;
    public filter([          ] next)
    {
        _next = next;
    }
    public void Process(ITelemetry item)
    {
        var x = item as [          ];
        if (x?.Url.AbsolutePath == " [          ] ")

        {
            return;
        }
        _next.Process(item);
    }
}
```

Answer:

Code segments

```
/health
/status
RequestTelemetry
PageViewTelemetry
ITelemetryProcessor
ITelemetryInitializer
```

Answer Area

```
public class filter : ITelemetryProcessor
{
    private readonly ITelemetryProcessor  _next;
    public filter( ITelemetryProcessor   next)
    {
        _next = next;
    }
    public void Process(ITelemetry item)
        {
            var x = item as RequestTelemetry  ;
            if (x?.Url.AbsolutePath == " /health          ")

            {
                return;
            }
            _next.Process(item);
        }
    }
```

Explanation:
Scenario: Exclude non-user actions from Application Insights telemetry.
Box 1: ITelemetryProcessor
To create a filter, implement ITelemetryProcessor. This technique gives you more direct control over what is included or excluded from the telemetry stream.
Box 2: ITelemetryProcessor
Box 3: ITelemetryProcessor
Box 4: RequestTelemetry
Box 5: /health
To filter out an item, just terminate the chain.
References:
https://docs.microsoft.com/en-us/azure/azure-monitor/app/api-filtering-sampling

QUESTION 84
Case Study 7
Policy service
You develop and deploy a stateful ASP.NET Core 2.1 web application named Policy service to an Azure App Service Web App. The application reacts to events from Azure Event Grid and performs policy actions based on those events.
The application must include the Event Grid Event ID field in all Application Insights telemetry.
Policy service must use Application Insights to automatically scale with the number of policy actions that it is performing.
Other
Anomaly detection service
You have an anomaly detection service that analyzes log infrmation for anomalies. It is implemented as an Azure Machine Learning model. The model is deployed as a web service.
If an anomaly is detected, an Azure Function that emails administrators is called by using an HTTP WebHook.
Health monitoring
All web applications and services have health monitoring at the /health service endpoint.
Issues
Policy loss
When you deploy Policy service, policies may not be applied if they were in the process of being applied during the deployment.
Performance issue
When under heavy load, the anomaly detection service undergoes slowdowns and rejects connections.
Notification latency
Users report that anomaly detection emails can sometimes arrive several minutes after an anomaly is detected.
Policies
Log policy
All Azure App Service Web Apps must write logs to Azure Blob storage. All log files should be saved to a container named logdrop. Logs must remain in the container for 15 days.

Authentication events
Authentication events are used to monitor users signing in and signing out. All authentication events must be processed by Policy service. Sign outs must be processed as quickly as possible.

PolicyLib
You have a shared library named PolicyLib that contains functionality common to all ASP.NET Core web services and applications. The PolicyLib library must:

▪ Exclude non-user actions from Application Insights telemetry.

▪ Provide methods that allow a web service to scale itself.

▪ Ensure that scaling actions do not disrupt application usage.

Relevant portions of the app files are shown below. Line numbers are included for reference only and include a two-character prefix that denotes the specific file to which they belong.

```
EventGridController.cs
EG01 public class EventGridController : Controller
EG02 {
EG03    public static AsyncLocal<string> EventId = new AsyncLocal<string>();
EG04    public IactionResult Process([FromBody] string eventsJson)
EG05    {
EG06      var events = JArray.Parse(eventsJson);
EG07
EG08      foreach (var @event in events)
EG09      {
EG10        EventId.Value = @event["id"].ToString();
EG11        if (@event["topic"].ToString(). Contains("providers/Microsoft.Storage"))
EG12        {
EG13        SendToAnomalyDetection Service(@event["data"]["url"].ToString());
EG14          }
EG15
EG16          {
EG17        EnsureLogging(@event["subject"].ToString());
EG18          }
EG19      }
EG20      return null;
EG21    }
EG22    private void EnsureLogging(string resource)
EG23    {
EG24      . . .
EG25    }
EG26    private async Task SendToAnomalyDetectionService(string uri)
EG27    {
EG28    var content = GetLogData(uri)
EG29    var scoreRequest = new
EG30    {
EG31      Inputs = new Dictionary<string, List<Dictionary<string, string>>>()
EG32      {
EG33        {
EG34        "input1",
EG35        new List<Dictionary<string, string>>()
EG36        {
EG37        new Dictionary<string, string>()
EG38        {
EG39          {
EG40        "logcontent", content
EG41          }
EG42        }
EG43        }
EG44        },
EG45      },
EG46    GlobalParameters = new Dictionary<string, string>() { }
EG47    };
EG48    var result = await (new HttpClient()).PostAsJsonAsync(" …", scoreRequest);
EG49    var rawModelResult = await result.Content.ReadAsStringAsync();
EG50    var modelResult = JObject.Parse(rawModelResult);
EG51 if (modelResult ["notify"].HasValues)
EG52    {
EG53    . . .
EG54    }
EG55 }
EG56 private (string name, string resourceGroup) ParseResourceId(string resourceId)
EG57 {
EG58    . . .
EG59 }
EG60 private string GetLogData(string uri)
EG61 {
EG62    . . .
EG63 }
EG64    static string BlobStoreAccountSAS (string containerName)
EG65    {
EG66    . . .
EG67 }
EG68 }
```

Relevant portions of the app files are shown below. Line numbers are included for reference only and include a two-character prefix that denotes the specific file to which they belong.

```
LoginEvent.cs
LE01 public class LoginEvent
LE02 {
LE03
LE04 public string subject { get ; set ; }
LE05 public DateTime eventTime { get ; set ; }
LE06 public Dictionary<string, string> data { get; set; }
LE07 public string Serialize()
LE08 {
LE09     return JsonConvert.SerializeObject(this);
LE10 }
LE11 }
```

Drag and Drop Question
You need to ensure that PolicyLib requirements are met.
How should you complete the code segment? To answer, drag the appropriate code segments to the correct locations.
Each code segment may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content.
NOTE: Each correct selection is worth one point.

Code segments	Answer Area
Process	public class IncludeEventId: [_____]
Initialize	(
telemetry.Sequence	public void [_____]
ITelemetryProcessor	(ITelemetry telemetry)
ITelemetryInitializer	{
Telemetry.Context	[_____] .Properties["KventId"] =
EventGridController.EventId.Value	[_____]
[(EventTelemetry)telemetry.Properties["EventId"]	)
	)

Answer:

Code segments	Answer Area
Process	public class IncludeEventId: ITelemetryInitializer
	(
telemetry.Sequence	public void Initialize
ITelemetryProcessor	(ITelemetry telemetry)
	{
	Telemetry.Context .Properties["KventId"] =
	[(EventTelemetry)telemetry.Properties["EventId"]
EventGridController.EventId.Value	)
	)

Explanation:
Scenario: You have a shared library named PolicyLib that contains functionality common to all ASP.NET Core web services and applications. The PolicyLib library must:
Exclude non-user actions from Application Insights telemetry.
Provide methods that allow a web service to scale itself.
Ensure that scaling actions do not disrupt application usage.
Box 1: ITelemetryInitializer
Use telemetry initializers to define global properties that are sent with all telemetry; and to override selected behavior of the standard telemetry modules.
Box 2: Initialize
Box 3: Telemetry.Context
Box 4: [(EventTelemetry)telemetry.Properties("EventID")

QUESTION 85
Case Study 7
Policy service
You develop and deploy a stateful ASP.NET Core 2.1 web application named Policy service to an Azure App Service Web App. The application reacts to events from Azure Event Grid and performs policy actions based on those events. The application must include the Event Grid Event ID field in all Application Insights telemetry.
Policy service must use Application Insights to automatically scale with the number of policy actions that it is performing.

Other
Anomaly detection service
You have an anomaly detection service that analyzes log infrmation for anomalies. It is implemented as an Azure Machine Learning model. The model is deployed as a web service.
If an anomaly is detected, an Azure Function that emails administrators is called by using an HTTP WebHook.

Health monitoring
All web applications and services have health monitoring at the /health service endpoint.

Issues
Policy loss
When you deploy Policy service, policies may not be applied if they were in the process of being applied during the deployment.

Performance issue
When under heavy load, the anomaly detection service undergoes slowdowns and rejects connections.

Notification latency
Users report that anomaly detection emails can sometimes arrive several minutes after an anomaly is detected.

Policies
Log policy
All Azure App Service Web Apps must write logs to Azure Blob storage. All log files should be saved to a container named logdrop. Logs must remain in the container for 15 days.

Authentication events
Authentication events are used to monitor users signing in and signing out. All authentication events must be processed by Policy service. Sign outs must be processed as quickly as possible.

PolicyLib
You have a shared library named PolicyLib that contains functionality common to all ASP.NET Core web services and applications. The PolicyLib library must:

・Exclude non-user actions from Application Insights telemetry.

・Provide methods that allow a web service to scale itself.

・Ensure that scaling actions do not disrupt application usage.

Relevant portions of the app files are shown below. Line numbers are included for reference only and include a two-character prefix that denotes the specific file to which they belong.

```
EventGridController.cs
EG01 public class EventGridController : Controller
EG02 {
EG03    public static AsyncLocal<string> EventId = new AsyncLocal<string>();
EG04    public IactionResult Process([FromBody] string eventsJson)
EG05    {
EG06      var events = JArray.Parse(eventsJson);
EG07
EG08      foreach (var @event in events)
EG09      {
EG10        EventId.Value = @event["id"].ToString();
EG11        if (@event["topic"].ToString(). Contains("providers/Microsoft.Storage"))
EG12        {
EG13        SendToAnomalyDetection Service(@event["data"]["url"].ToString());
EG14          }
EG15
EG16          {
EG17        EnsureLogging(@event["subject"].ToString());
EG18          }
EG19      }
EG20      return null;
EG21 }
EG22    private void EnsureLogging(string resource)
EG23    {
EG24      . . .
EG25    }
EG26    private async Task SendToAnomalyDetectionService(string uri)
EG27    {
EG28    var content = GetLogData(uri)
EG29    var scoreRequest = new
EG30    {
EG31      Inputs = new Dictionary<string, List<Dictionary<string, string>>>()
EG32      {
EG33      {
EG34        "input1",
EG35       new List<Dictionary<string, string>>()
EG36       {
EG37         new Dictionary<string, string>()
EG38       {
EG39        {
EG40          "logcontent", content
EG41        }
EG42      }
EG43        }
EG44      },
EG45      },
EG46    GlobalParameters = new Dictionary<string, string>() ( )
EG47    };
EG48    var result = await (new HttpClient()).PostAsJsonAsync(" …", scoreRequest);
EG49    var rawModelResult = await result.Content.ReadAsStringAsync();
EG50    var modelResult = JObject.Parse(rawModelResult);
EG51 if (modelResult ["notify"].HasValues)
EG52    {
EG53      . . .
EG54      }
EG55 }
EG56    private (string name, string resourceGroup) ParseResourceId(string resourceId)
EG57    {
EG58      . . .
EG59 }
EG60    private string GetLogData(string uri)
EG61    {
EG62      . . .
EG63    }
EG64      static string BlobStoreAccountSAS (string containerName)
EG65    {
EG66      . . .
EG67    }
EG68 }
```

Relevant portions of the app files are shown below. Line numbers are included for reference only and include a two-character prefix that denotes the specific file to which they belong.
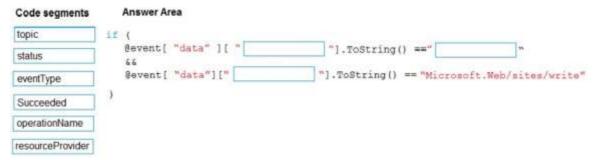
```
LoginEvent.cs
LE01 public class LoginEvent
LE02 {
LE03
LE04 public string subject { get ; set ; }
LE05 public DateTime eventTime { get ; set ; }
LE06 public Dictionary<string, string> data { get; set; }
LE07 public string Serialize()
LE08 {
LE09     return JsonConvert.SerializeObject(this);
LE10 }
LE11 }
```

Drag and Drop Question
You need to add code at line EG15 in EventGridController.cs to ensure that the Log policy applies to all services.
How should you complete the code? To answer, drag the appropriate code segments to the correct locations. Each code segment may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content.
NOTE: Each correct selection is worth one point.

Answer:

Explanation:
Scenario, Log policy: All Azure App Service Web Apps must write logs to Azure Blob storage.
Box 1: Status
Box 2: Succeeded
Box 3: operationName
Microsoft.Web/sites/write is resource provider operation. It creates a new Web App or updates an existing one,
References:
https://docs.microsoft.com/en-us/azure/role-based-access-control/resource-provider-operations

QUESTION 86
You are writing code to create and run an Azure Batch job.
You have created a pool of compute nodes.
You need to choose the right class and its method to submit a batch job to the Batch service.
Which method should you use?

300-410 Exam Dumps **300-410 Exam Questions** **300-410 PDF Dumps** **300-410 VCE Dumps**

https://www.braindump2go.com/300-410.html

A. JobOperations.EnableJobAsync(String,
    IEnumerable<BatchClientBehavior>,CancellationToken)
B. JobOperations.CreateJob()
C. CloudJob.Enable(IEnumerable<BatchClientBehavior>)
D. JobOperations.EnableJob(String,IEnumerable<BatchClientBehavior>)
E. CloudJob.CommitAsync(IEnumerable<BatchClientBehavior>, CancellationToken)

Answer: E
Explanation:
A Batch job is a logical grouping of one or more tasks. A job includes settings common to the tasks, such as priority and the pool to run tasks on. The app uses the BatchClient.JobOperations.CreateJob method to create a job on your pool. The Commit method submits the job to the Batch service. Initially the job has no tasks.
{
CloudJob job = batchClient.JobOperations.CreateJob();
job.Id = JobId;
job.PoolInformation = new PoolInformation { PoolId = PoolId }; job.Commit();
}
...
References:
https://docs.microsoft.com/en-us/azure/batch/quick-run-dotnet

QUESTION 87
Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.
After you answer a question in this question, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.
Margie's Travel is an international travel and bookings management service. The company is expanding into restaurant bookings. You are tasked with implementing Azure Search for the restaurants listed in their solution.
You create the index in Azure Search.
You need to import the restaurant data into the Azure Search service by using the Azure Search .NET SDK.
Solution:
1. Create a SearchServiceClient object to connect to the search index.
2. Create a DataContainer that contains the documents which must be added.
3. Create a DataSource instance and set its Container property to the DataContainer.
4. Set the DataSources property of the SearchServiceClient.
Does the solution meet the goal?

A. Yes
B. No

Answer: B
Explanation:
Use the following method:
1. Create a SearchIndexClient object to connect to the search index
2. Create an IndexBatch that contains the documents which must be added.
3. Call the Documents.Index method of the SearchIndexClient and pass the IndexBatch.
References:
https://docs.microsoft.com/en-us/azure/search/search-howto-dotnet-sdk

QUESTION 88
Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.
After you answer a question in this question, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.
Margie's Travel is an international travel and bookings management service. The company is expanding into restaurant

bookings. You are tasked with implementing Azure Search for the restaurants listed in their solution.
You create the index in Azure Search.
You need to import the restaurant data into the Azure Search service by using the Azure Search .NET SDK.
Solution:
1. Create a SearchServiceClient object to connect to the search index.
2. Create a DataContainer that contains the documents which must be added.
3. Create a DataSource instance and set its Container property to the DataContainer.
4. Call the Documents.Suggest method of the SearchIndexClient and pass the DataSource.
Does the solution meet the goal?

A. Yes
B. No

Answer: B
Explanation:
Use the following method:
1. Create a SearchIndexClient object to connect to the search index
2. Create an IndexBatch that contains the documents which must be added.
3. Call the Documents.Index method of the SearchIndexClient and pass the IndexBatch.
References:
https://docs.microsoft.com/en-us/azure/search/search-howto-dotnet-sdk

QUESTION 89
Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.
After you answer a question in this question, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.
Margie's Travel is an international travel and bookings management service. The company is expanding into restaurant bookings. You are tasked with implementing Azure Search for the restaurants listed in their solution.
You create the index in Azure Search.
You need to import the restaurant data into the Azure Search service by using the Azure Search .NET SDK.
Solution:
1. Create a SearchIndexClient object to connect to the search index
2. Create an IndexBatch that contains the documents which must be added.
3. Call the Documents.Index method of the SearchIndexClient and pass the IndexBatch.
Does the solution meet the goal?

A. Yes
B. No

Answer: A
Explanation:
1. The index needs to be populated. To do this, we will need a SearchIndexClient. There are two ways to obtain one: by constructing it, or by calling Indexes.GetClient on the SearchServiceClient. Here we will use the first method.
2. Create the indexBatch with the documents
Something like:
var hotels = new Hotel[];
{
new Hotel()
{
HotelId = "3",
BaseRate = 129.99,
Description = "Close to town hall and the river"
}
};
...
var batch = IndexBatch.Upload(hotels);

300-410 Exam Dumps **300-410 Exam Questions** **300-410 PDF Dumps** **300-410 VCE Dumps**

https://www.braindump2go.com/300-410.html

3. The next step is to populate the newly-created index
Example:
var batch = IndexBatch.Upload(hotels);
try
{
indexClient.Documents.Index(batch);
}
References:
https://docs.microsoft.com/en-us/azure/search/search-howto-dotnet-sdk

QUESTION 90
A company is implementing a publish-subscribe (Pub/Sub) messaging component by using Azure Service Bus. You are developing the first subscription application.
In the Azure portal you see that messages are being sent to the subscription for each topic. You create and initialize a subscription client object by supplying the correct details, but the subscription application is still not consuming the messages.
You need to complete the source code of the subscription client
What should you do?

A. await subscriptionClient.CloseAsync();
B. await subscriptionClient.AddRuleAsync(new RuleDescription (RuleDescription.DefaultRuleName, new TrueFilter()));
C. subscriptionClient.RegisterMessageHandler(ProcessMessagesAsync, messageHandlerOptions);
D. subscriptionClient = new SubscriptionClient(ServiceBusConnectionString, TopicName, SubscriptionName);

Answer: C
Explanation:
Using topic client, call RegisterMessageHandler which is used to receive messages continuously from the entity. It registers a message handler and begins a new thread to receive messages. This handler is waited on every time a new message is received by the receiver.
subscriptionClient.RegisterMessageHandler(ReceiveMessagesAsync, messageHandlerOptions);
References:
https://www.c-sharpcorner.com/article/azure-service-bus-topic-and-subscription-pub-sub/

QUESTION 91
You develop an Azure web app. You monitor performance of the web app by using Application Insights.
You need to ensure the cost for Application Insights does not exceed a preset budget.
What should you do?

A. Implement ingestions sampling using the Application Insights SDK.
B. Set a daily cap for the Application Insights instance.
C. Implement ingestion sampling using the Azure portal.
D. Implement adaptive sampling using the Azure portal.
E. Implement adaptive sampling using the Application Insights SDK.

Answer: E
Explanation:
Sampling is an effective way to reduce charges and stay within your monthly quota.
You can set sampling manually, either in the portal on the Usage and estimated costs page; or in the ASP.NET SDK in the .config file; or in the Java SDK in the ApplicationInsights.xml file, to also reduce the network traffic.
Adaptive sampling is the default for the ASP.NET SDK. Adaptive sampling automatically adjusts to the volume of telemetry that your app sends. It operates automatically in the SDK in your web app so that telemetry traffic on the network is reduced.
Incorrect Answers:
B: You can use the daily volume cap to limit the data collected.
To change the daily cap, in the Configure section of your Application Insights resource, in the Usage and estimated

costs pane, select Daily Cap.
References:
https://docs.microsoft.com/en-us/azure/azure-monitor/app/sampling

QUESTION 92
You are developing an ASP.NET Core Web API web service. The web service uses Azure Application Insights for all telemetry and dependency tracking. The web service reads and writes data to a database other than Microsoft SQL Server.
You need to ensure that dependency tracking works for calls to the third-party database.
Which two Dependency Telemetry properties should you store in the database? Each correct answer presents part of the solution.
NOTE: Each correct selection is worth one point.

A. Telemetry.Context.Operation.Id
B. Telemetry.Name
C. Telemetry.Context.Cloud.RoleInstance
D. Telemetry.Context.Session.Id
E. Telemetry.Id

Answer: AE
Explanation:
https://docs.microsoft.com/en-us/azure/azure-monitor/app/custom-operations-tracking

QUESTION 93
You use Azure Table storage to store customer information for an application. The data contains customer details and is partitioned by last name.
You need to create a query that returns all customers with the last name Smith.
Which code segment should you use?

A. TableQuery.GenerateFilterCondition("PartitionKey", Equals, "Smith")
B. TableQuery.GenerateFilterCondition("LastName", Equals, "Smith")
C. TableQuery.GenerateFilterCondition("PartitionKey",
   QueryComparisons.Equal, "Smith")
D. TableQuery.GenerateFilterCondition("LastName",
   QueryComparisons.Equal, "Smith")

Answer: C
Explanation:
Retrieve all entities in a partition. The following code example specifies a filter for entities where 'Smith' is the partition key. This example prints the fields of each entity in the query results to the console.
Construct the query operation for all customer entities where PartitionKey="Smith".
TableQuery<CustomerEntity> query = new TableQuery<CustomerEntity>().Where
(TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal, "Smith")); References:
https://docs.microsoft.com/en-us/azure/cosmos-db/table-storage-how-to-use-dotnet