Event Hub & IoT Hub

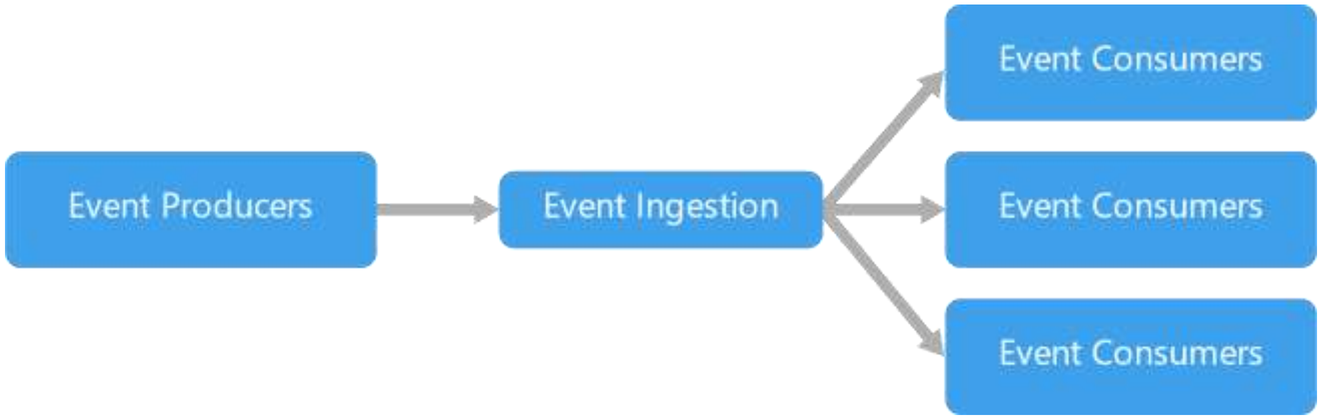
Lecture 09

Deep Azure @ McKesson

Zoran B. Djordjević

Event Driven Architectural Style

- Large number of applications requires collection and analysis of “events”, typically small textual notification about value of a system parameter or a log or audit entry.
- Such applications are frequently assembled in what is know as the event-driven architecture consists of **event producers** that generate a stream of events, and **event consumers** that listen for the events.



- Events are delivered in near real time, so consumers can respond immediately to events as they occur. Producers are decoupled from consumers — a producer doesn't know which consumers are listening. Consumers are also decoupled from each other, and every consumer sees all of the events.
- In some systems, such as IoT, events must be ingested at very high volumes. This differs from a Competing Consumers Architectural pattern, where consumers pull messages from a queue and a message is processed just once.

Models of Event-Driven Architecture

An event driven architecture can use a pub/sub model or an event stream model.

- **Pub/sub**: The messaging infrastructure keeps track of subscriptions. When an event is published, it sends the event to each subscriber. After an event is received, it cannot be replayed, and new subscribers do not see the event.
- **Event streaming**: Events are written to a log. Events are strictly ordered (within a partition) and durable. Clients don't subscribe to the stream, instead a client can read from any part of the stream. The client is responsible for advancing its position in the stream. That means a client can join at any time, and can replay events.

On the consumer side, there are some common variations:

- **Simple event processing**. An event immediately triggers an action in the consumer. For example, you could use Azure Functions with a Service Bus trigger, so that a function executes whenever a message is published to a Service Bus topic.
- **Complex event processing**. A consumer processes a series of events, looking for patterns in the event data, using a technology such as Azure Stream Analytics or Apache Storm. For example, you could aggregate readings from an embedded device over a time window, and generate a notification if the moving average crosses a certain threshold.
- **Event stream processing**. Use a data streaming platform, such as Azure IoT Hub or Apache Kafka, as a pipeline to ingest events and feed them to stream processors. The stream processors act to process or transform the stream. There may be multiple stream processors for different subsystems of the application. This approach is a good fit for IoT workloads.

Use Cases for Event Driven Architecture

We use Event Driver Architecture when

- Multiple subsystems must process the same events.
- Real-time processing with minimum time lag.
- Complex event processing, such as pattern matching or aggregation over time windows.
- High volume and high velocity of data, such as IoT.

Benefits of Event Driven Architecture

- Producers and consumers are decoupled.
- No point-to point-integrations. It's easy to add new consumers to the system.
- Consumers can respond to events immediately as they arrive.
- Highly scalable and distributed.
- Subsystems have independent views of the event stream.

Challenges

- Guaranteed delivery. In some systems, especially in IoT scenarios, it's crucial to guarantee that events are delivered.
- Processing events in order or exactly once. Each consumer type typically runs in multiple instances, for resiliency and scalability. Hard to process events in order.

Azure Services for Events Driven Systems

- One can implement Event Driven Systems in many ways in Azure.
- Two main services specifically designed for constructing event driven systems are:
  - Azure IoT service and
  - Azure Event Hubs

Azure IoT Solutions

Azure IoT offers several options, each appropriate for different sets of customer requirements:

- Azure IoT Suite is an enterprise-grade collection of preconfigured solutions built on Azure Platform-as-a-Service that enable you to accelerate the development of custom IoT solutions.
- Microsoft IoT Central is a SaaS solution that uses a model-based approach to enable you to build enterprise-grade IoT solutions without requiring cloud solution development expertise.
- Azure IoT Hub is the core Azure Platform-as-a-Service that both Microsoft IoT Central and Azure IoT Suite make use of. IoT Hub enables reliable and securely bidirectional communications between millions of IoT devices and a cloud solution. IoT Hub helps you meet IoT implementation challenges such as:+
  - High-volume device connectivity and management.
  - High-volume telemetry ingestion.
  - Command and control of devices.
  - Device security enforcement.

Azure IoT Hub

- Azure IoT Hub provides an easy and secure way to connect, provision and manage millions of IoT devices sending and receiving billions of messages per month.
- IoT Hub is the bridge between your devices and Cloud applications that analyze generated data.
- IoT Hub allowing them to store, analyze and act on that data in real time.
- IoT Hub enables secure, reliable, two-way communication — from device to cloud and cloud to device — over open protocols such as MQTT, HTTPS and AMQPS. These protocols are already widely used in IoT.
- Developer are looking at three main objectives:
  - connecting devices to IoT Hub,
  - managing the IoT Hub service itself and
  - integrating IoT Hub into your overall IoT solution in the cloud.
- To achieve all three, the Azure IoT Hub service exposes REST APIs along with AMQP and MQTT communication support, but implementing communication protocols is not trivial by nature.
- Some other Cloud providers place emphasis on MQTT. Azure IoT apparently places emphasis on AMQP

AMQP.org

- AMQP, which stands for Advanced Message Queuing Protocol, was designed as an open replacement for existing proprietary messaging middleware.
- Two of the most important reasons to use AMQP are reliability and interoperability. As the name implies, it provides a wide range of features related to messaging, including reliable queuing, topic-based publish-and-subscribe messaging, flexible routing, transactions, and security. AMQP exchanges route messages directly—in fanout form, by topic, and also based on headers.
- You can restrict access to queues, manage their depth, and more. Features like message properties, annotations and headers make it a good fit for a wide range of enterprise applications.
- This protocol was designed for reliability at the many large companies who depend on messaging to integrate applications and move data around their organisations.
- AMQP is a binary wire protocol which was designed for interoperability between different vendors. Where other protocols have failed, AMQP adoption has been strong.
- Companies like JP Morgan use it to process 1 billion messages a day. NASA uses it for Nebula Cloud Computing. Google uses it for complex event processing. AMQP is used in one of the world's largest biometric databases India's Aadhar project—home to 1.2 billion identities. AMQP is used in the Ocean Observatories Initiative—an architecture that collects 8 terabytes of data per day.

MQTT.org

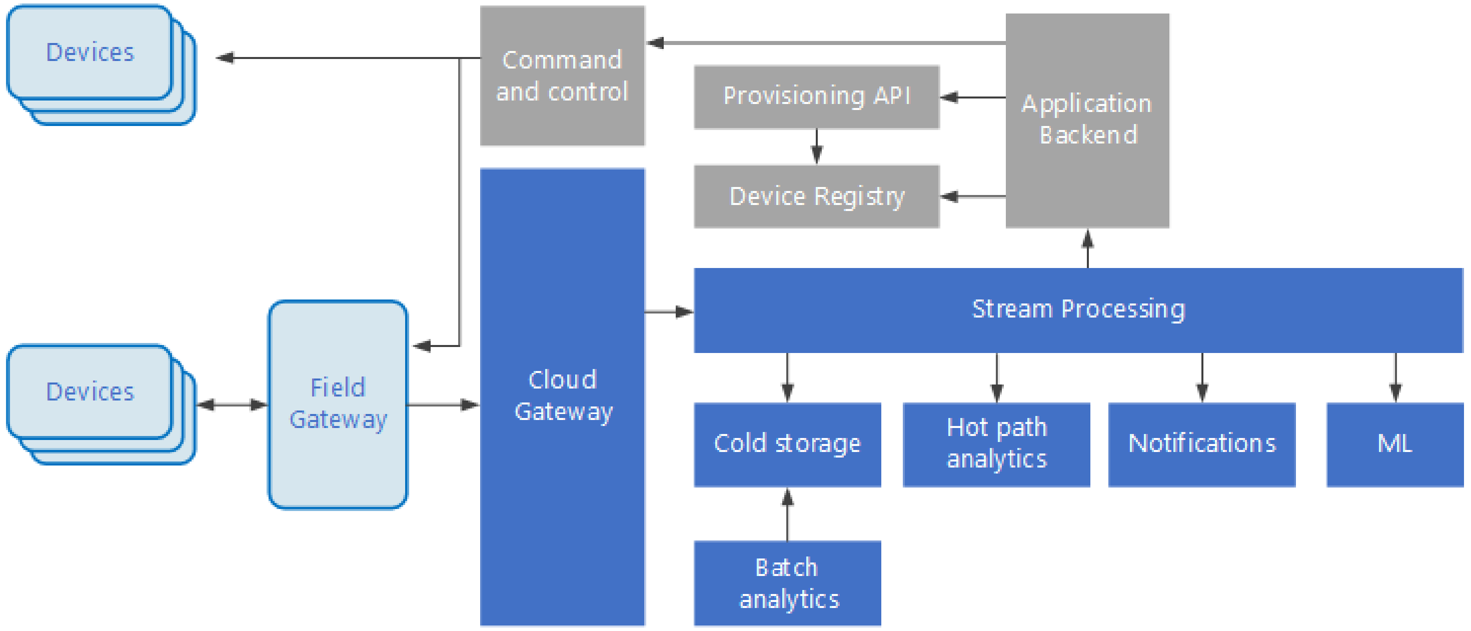
- MQTT (Message Queue Telemetry Transport) was originally developed by IBM’s pervasive computing team and their partners in the industrial sector. Over the past couple of years the protocol has been moved into the open source community, seen significant growth in popularity as mobile applications have taken off.
- The design principles and aims of MQTT are much more simple and focused than those of AMQP—it provides publish-and-subscribe messaging (no queues, in spite of the name) and was specifically designed for resource-constrained devices and low bandwidth, high latency networks such as dial up lines and satellite links, for example. Basically, it can be used effectively in embedded systems.
- One of the advantages MQTT has over more full-featured “enterprise messaging” brokers is that its intentionally low footprint makes it ideal for today’s mobile and developing “Internet of Things” style applications. For example, Facebook is using MQTT as part of its mobile applications because it has such a low power draw and is light on network bandwidth.
- Some of the MQTT-based brokers support many thousands of concurrent device connections. MQTT offers three qualities of service: 1) fire-and-forget / unreliable; 2) “at least once” to ensure it is sent a minimum of one time (but might be sent more than one time), and 3) “exactly once”.
- MQTT’s strengths are simplicity (just five API methods), a compact binary packet payload (no message properties, compressed headers, much less verbose than something text-based like HTTP). MQTT makes a good fit for simple push messaging scenarios such as temperature updates, stock price tickers, oil pressure feeds or mobile notifications. It is also very useful for connecting machines together, such as connecting an Arduino device to a web service with MQTT.

Developer's Environment

- Both the service and the device client sides of Azure IoT are supported with open source SDKs with simple and straightforward APIs. All Azure IoT client SDKs are open sourced on GitHub and offer the following:
- Device client:
  - Support for C, Node.js, Python, Java and C#
  - Support for AMQP, AMQP over WebSockets, MQTT and HTTP/REST for the device–to-cloud communication
  - Support for SSL (using third-party dependencies such as openSSL or WolfSSL)
  - Simple APIs to:
    - Establish a secure connection to IoT Hub
    - Send messages to IoT Hub
    - Receive messages from IoT Hub
- Service client:
  - Support for C, Node.js, Python, Java and C#
  - Simple APIs to:
    - Manage the device registry (Create, Remove, Update, Delete)
    - Read data from IoT Hub
    - Send messages to specific devices

IoT Architecture

- Event-driven architectures are central to IoT solutions. The following is a possible logical architecture for IoT, using a reliable, low latency messaging system.
- Devices might send events directly to the cloud gateway, or through a field gateway. Gateways might also preprocess the raw device events, performing functions such as filtering, aggregation, or protocol transformation.
- After ingestion, events go through one or more stream processors that can route the data (for example, to storage) or perform analytics and other processing.



Security of IoT Communications

- Azure IoT Hub offers secure mechanisms for communication with connecting devices.
- Azure IoT Hub service uses a set of permissions to grant access to endpoints. Developers can set up permissions using shared access policies for services, apps and devices and can create per-device security credentials leveraging the device identity registry which is provided with the service.
- Then Azure IoT Hub will authenticate endpoints by verifying a token against the shared access policies and device identity registry security credentials.
- Security credentials, such as symmetric keys, are never sent over the wire. The security token generation is implemented in the device client SDKs.
- All endpoints connect to Azure IoT Hub over TLS, ensuring no endpoint is ever exposed on unencrypted or unsecured channels.
- The device client SDKs implements all of this under the hoods for you, so as a developer you can focus on the development of the actual solution trusting that the plumbing is robust and secured.
- The service client SDKs also implement secure connection between processing applications and Azure IoT Hub.

Sending and Receiving Messages

- Once a secure connection is established from devices to Azure IoT Hub, all that's left to do is to send and receive messages.
- Azure IoT Hub offers a raw messaging infrastructure, meaning you can put whatever you want and need in the body of your messages.
- Sending messages is as simple as the C code snippet below:

```
msgHandle = IoTHubMessage_CreateFromByteArray(msgText, strlen(msgText));
IoTHubClient_SendEventAsync(iotHubClientHandle, msgHandle,
SendConfirmationCallback, NULL);
```

- And if you want to receive messages from Azure IoT Hub on the device, you need to register a callback like in the below C example:

```
IoTHubClient_SetMessageCallback(iotHubClientHandle, ReceiveMessageCallback,
NULL);
```

- The SDKs also allow to know and notify when a message has been received on the other side.

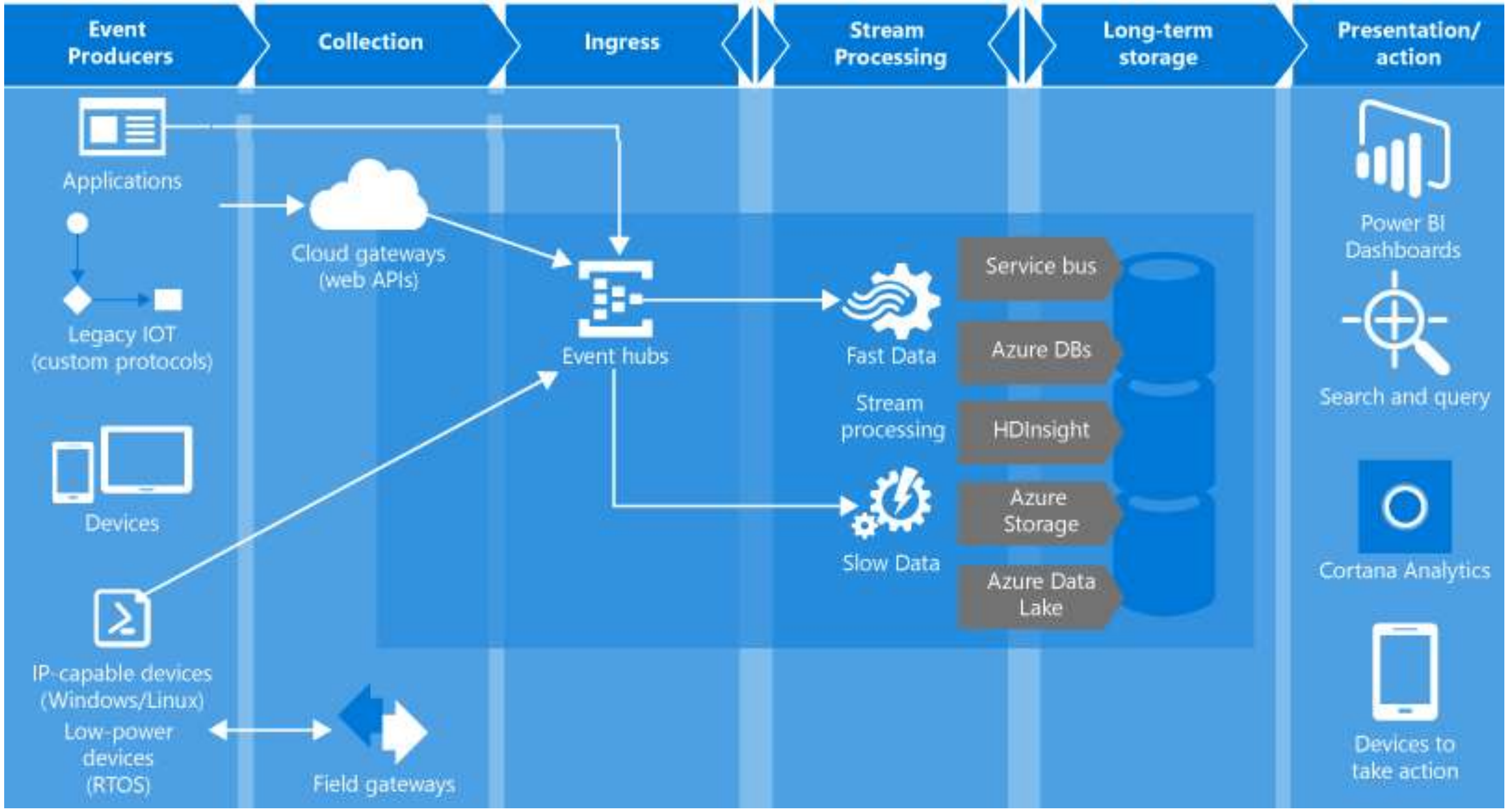
Azure Event Hub

- Azure Event Hubs is a highly scalable data streaming platform and event ingestion service, capable of receiving and processing millions of events per second.
- Event Hubs can process and store events, data, or telemetry produced by distributed software and devices.
- Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters.
- With the ability to provide publish-subscribe capabilities with low latency and at massive scale, Event Hubs serves as the "on ramp" for Big Data.

- Event Hubs event and telemetry handling capabilities make it especially useful for:
  - Application instrumentation
  - User experience or workflow processing
  - Internet of Things (IoT) scenarios
- For example, Event Hubs enables behavior tracking in mobile apps, traffic information from web farms, in-game event capture in console games, or telemetry collected from industrial machines, connected vehicles, or other devices.

Event Hub Architectural Environment

- The common role that Event Hubs plays in solution architectures is the "front door" for an event pipeline, often called an *event ingestor*. An event ingestor is a component or service that sits between event publishers and event consumers to decouple the event stream from the consumption of those events.



Key Features of Event Hub

- Event Hubs provides message stream handling capability but has characteristics that are different from traditional enterprise messaging.
- Event Hubs capabilities are built around high throughput and event processing scenarios.
- Event Hubs is different from Azure Service Bus messaging, and does not implement some of the capabilities that are available for Service Bus messaging entities, such as topics.
- Event Hubs contains additional key features:
  - **Event producers/publishers** exchange messages via AMQP 1.0 or HTTPS.
  - **Capture e**nables you to capture Event Hubs streaming data and store it in an Azure Blob storage account.
  - **Partitions** are subsets of, or partition of, of the event stream which can be made visible to separate consumers.
  - **SAS tokens i**dentify and authenticate the event publisher.
  - **Event consumers are** entities that reads event data from an event hub. Event consumers connect to Event Hub via AMQP 1.0.
  - **Consumer groups p**rovides each multiple of consuming applications with a separate view of the event stream, enabling those consumers to act independently.
  - **Throughput units** are-purchased units of capacity. A single partition has a maximum scale of 1 throughput unit.

Comparison of IoT Hub and Event Hub

- It is not uncommon to use both IoT Hub and Event Hubs in the same solution.
- IoT Hub handles the device-to-cloud communication, and Event Hubs handles later-stage event ingress into real-time processing engines.
- The following table list differences between two services:

Area	IoT Hub	Event Hubs
Communication patterns	Enables device-to-cloud communications (messaging, file uploads, and reported properties) and cloud-to-device communications (direct methods, desired properties, messaging).	Only enables event ingress (usually considered for device-to-cloud scenarios).
Device state information	Device twins can store and query device state information.	No device state information can be stored.
Device protocol support	MQTT, MQTT over WebSockets, AMQP, AMQP over WebSockets, and HTTPS. Additionally, IoT Hub works with the Azure IoT protocol gateway, a customizable protocol gateway implementation to support custom protocols.	Supports AMQP, AMQP over WebSockets, and HTTPS.
Security	Per-device identity and revocable access control. See the Security section of the IoT Hub developer guide.	Event Hubs-wide shared access policies, with limited revocation support through publisher's policies. IoT solutions are often required to implement a custom solution to support per-device credentials and anti-spoofing measures.

Comparison of IoT Hub and Event Hub

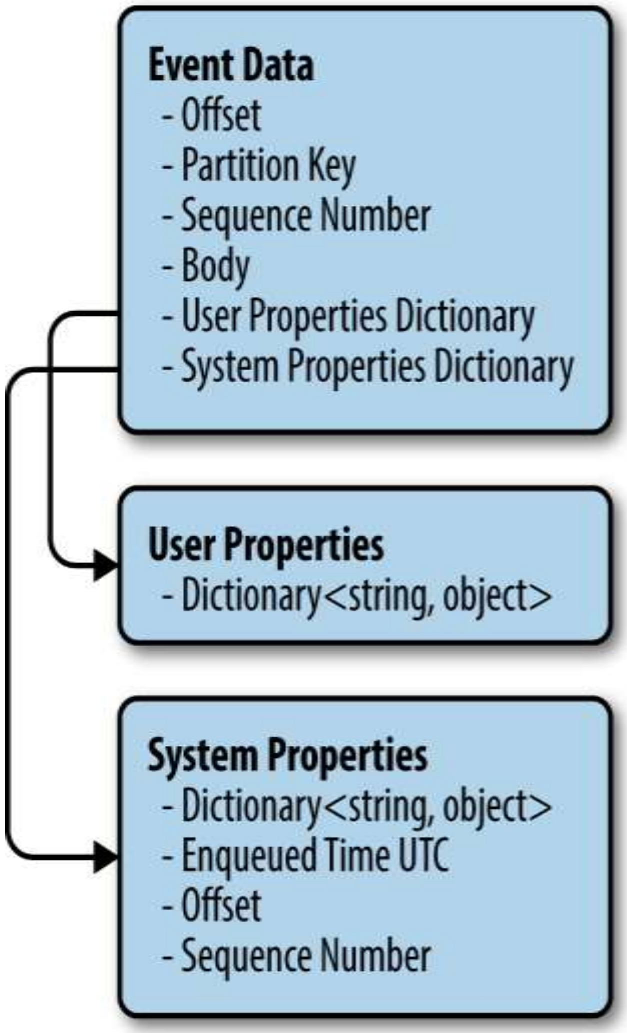
Area	IoT Hub	Event Hubs
Operations monitoring	Enables IoT solutions to subscribe to a rich set of device identity management and connectivity events such as individual device authentication errors, throttling, and bad format exceptions. These events enable you to quickly identify connectivity problems at the individual device level.	Exposes only aggregate metrics.
Scale	Is optimized to support millions of simultaneously connected devices.	Meters the connections as per Azure Event Hubs quotas. On the other hand, Event Hubs enables you to specify the partition for each message sent.
Device SDKs	Provides device SDKs for a large variety of platforms and languages, in addition to direct MQTT, AMQP, and HTTPS APIs.	Is supported on .NET, Java, and C, in addition to AMQP and HTTPS send interfaces.
File upload	Enables IoT solutions to upload files from devices to the cloud. Includes a file notification endpoint for workflow integration and an operations monitoring category for debugging support.	Not supported.
Route messages to multiple endpoints	Up to 10 custom endpoints are supported. Rules determine how messages are routed to custom endpoints. For more information, see Send and receive messages with IoT Hub.	Requires additional code to be written and hosted for message dispatching.

Ingest and Storage with Event Hubs

- We will dive one layer further into the ingest storage aspects of the highly scalable, multi-consumer messaging system, the Event Hubs.
- Subsequently, we will examine the other side of ingest, which is the consumption and processing of messages pulled from Event Hubs.
- The sender can use any of several of software development kits (SDKs) to communicate with Event Hubs. There are client SDKs for .NET, C, Node.js, and Java.
- Alternately, the REST API can be used directly from any platform that supports making RESTful calls.
- The sender creates an event, which represents the “message” that is sent to the Event Hub.
- In the .NET and Java SDK for Event Hubs, a sender creates an instance of the `EventData` class. That class has the properties shown on the next slide.

Structure of Message, EventData Class

- There are three types of properties that are the most important: user, system, and body.
- System properties are set by Event Hubs itself.
- User properties can include key/value pairs that contain string data that is useful for downstream message processing (identifying the importance of a message, capturing the ID of the sender, etc.).
- Body properties are ultimately always serialized to a binary payload. When sending from the .NET/Java SDK, the body is represented as an array byte[].
- When sending message by REST API, it is a challenge to prepare a binary payload, so Event Hubs accepts JSON as the payload format.
- The most common format is to have the body contain binary serialized JSON.
- The Offset, Enqueued Time, and Sequence Number properties are set by Event Hubs upon event ingest.
- These properties have an important role to play with consumers of the Event Hub.

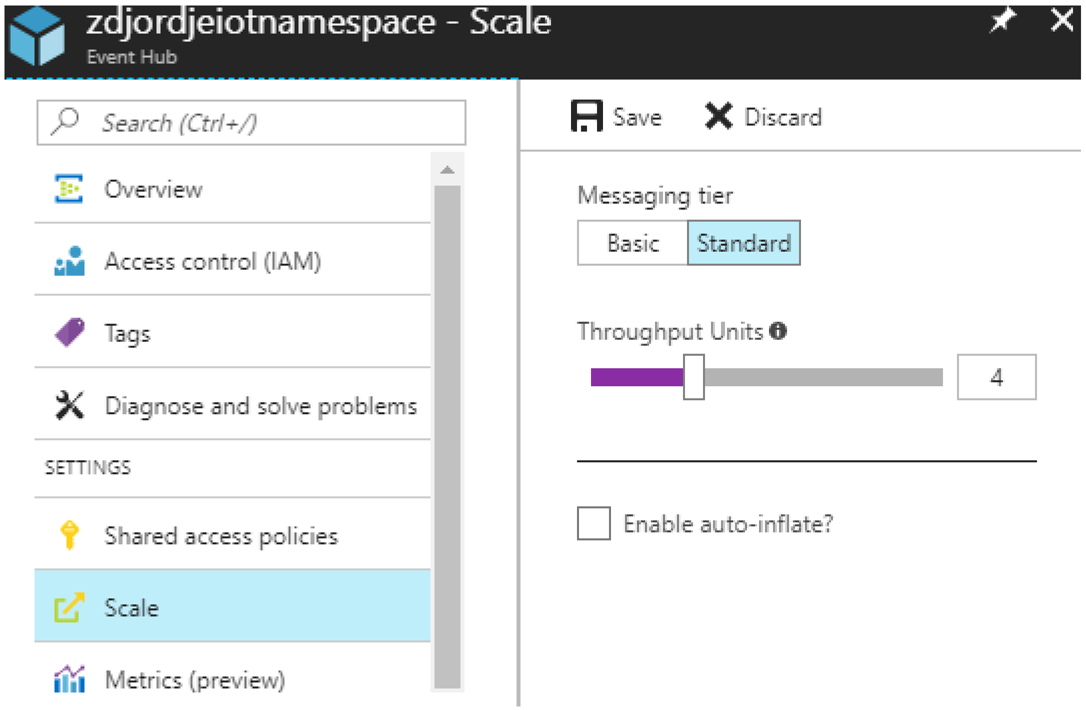


Capacity

- Each event instance can be at most 256 KB in size.
- For improved sending performance, senders can batch a list of events to send in a single go so long as the total size of all events in the batch does not exceed 256 KB

Throughput

- The scale of an Event Hub is controlled via throughput units (TUs), as shown in the figure. Each TU controls the volume of data ingress and egress that can be handled by the Event Hubs instance.
- For the senders, the event ingress throughput scales at 1 MB/s per TU or 1,000 events/s per TU. If either limit is triggered, message ingress is throttled. By default, there is a per-subscription quota of 20 TUs, but this is a soft limit that can be raised via a request to Azure support.
- When requesting additional TUs, request them in batches of 20 up to a total of 100 TUs. Beyond 100 TUs, you can request additional TUs in batches of 100.
- TUs apply at the level of the service bus namespace (a scoping container for a set of service bus messaging entities like Event Hubs). Your allocated TUs can be shared among multiple Event Hubs



Messaging tier: Basic [✓ Standard]

Throughput Units: 4

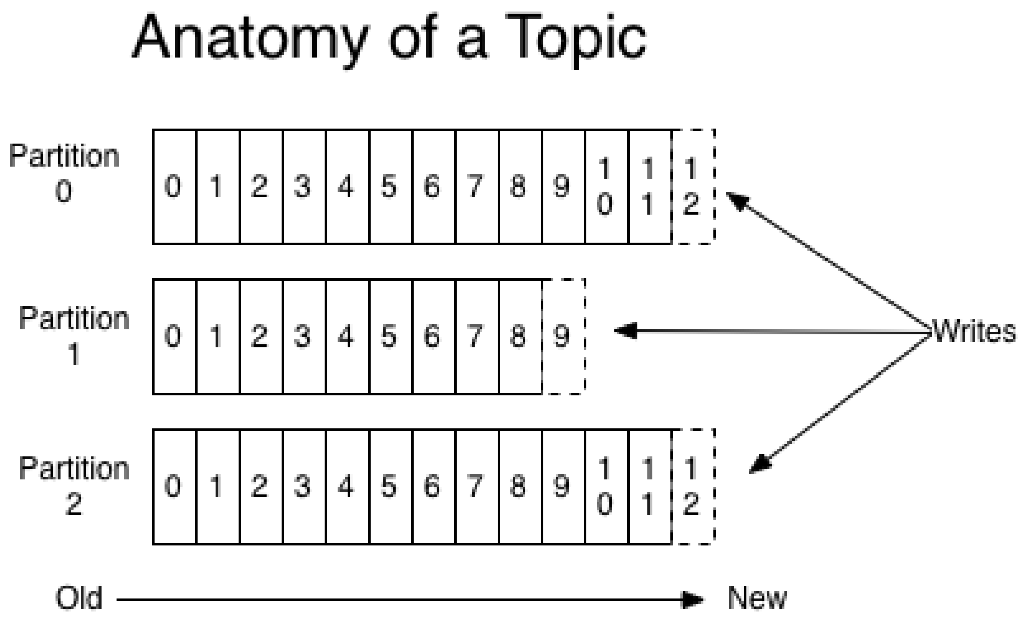
Enable auto-inflate? ☐

Partitions

- Event Hubs supports partitioning the ingested events into partitions—in other words, spreading messages across different “buckets” for storage.
- When an Event Hub is provisioned, the number of partitions, which can be 1–32, is specified.
- The number of partitions cannot be changed after provisioning.
- When you send events to Event Hubs, the default approach is for the messages to be distributed among partitions in a round-robin fashion. However, if a partition key is provided, this value can be used to influence the selection of the actual partition instead.
- The partition key is a string, and any events sent to Event Hubs that share the same partition key value (or more precisely, whose hash of the partition key is the same) are delivered in order to the same partition.
- From the sender’s perspective, beyond using the partition key (which does not identify a partition directly), individual partitions are not usually targeted by a sender.
- Partitions play a very important role for consumers of Event Hubs.

Partitions from Kafka Manuals

- Each message within a partition has an identifier called its *offset*. The offset the ordering of messages as an immutable sequence. Kafka maintains this message ordering for you. Consumers can read messages starting from a specific offset and are allowed to read from any offset point they choose, allowing consumers to join the cluster at any point in time they see fit. Given these constraints, each specific message in a Kafka cluster can be uniquely identified by a tuple consisting of the message's topic, partition, and offset within the partition.



Ingres Limit per Partition

- There is also an ingress limit that applies to each partition. Each partition within an Event Hub can utilize at most 1 TU.
- If you had an Event Hubs instance with 32 partitions, and had allocated 32 TUs to the service bus namespace that contains the Event Hubs instance, you are basically ensuring that each partition has access to its full potential of 1 MB/s ingress.
- If you allocated 33 TUs, this would not benefit the partitions any further (albeit the extra capacity would benefit other Event Hubs instances in the service bus namespace).

Message Storage

- The total storage capacity for an Event Hubs instance is not limited. Out of the box, 84 GB of storage per TU is included free of charge. Any storage you consume beyond 84 GB is billed at the rates for using Azure Storage in the locally redundant storage (LRS) mode.
- The way Event Hubs manages storage (since events are not deleted by the consumer when they are retrieved) is by applying a retention policy.
- Event Hubs has a configurable retention policy where messages older than the retention period are automatically purged. The retention period can be set in units of days, between 1 and 7 days.

Concurrent Connections, Limits

- There’s one final important consideration for message ingest: the number of concurrent connections has limitations depending on how the senders are communicating with Event Hubs.
- If you are using HTTPS, there is no limit on the number of concurrent connections;
- If you are using AMQP, then there is a service bus namespace– wide limit of 5,000 concurrent connections.

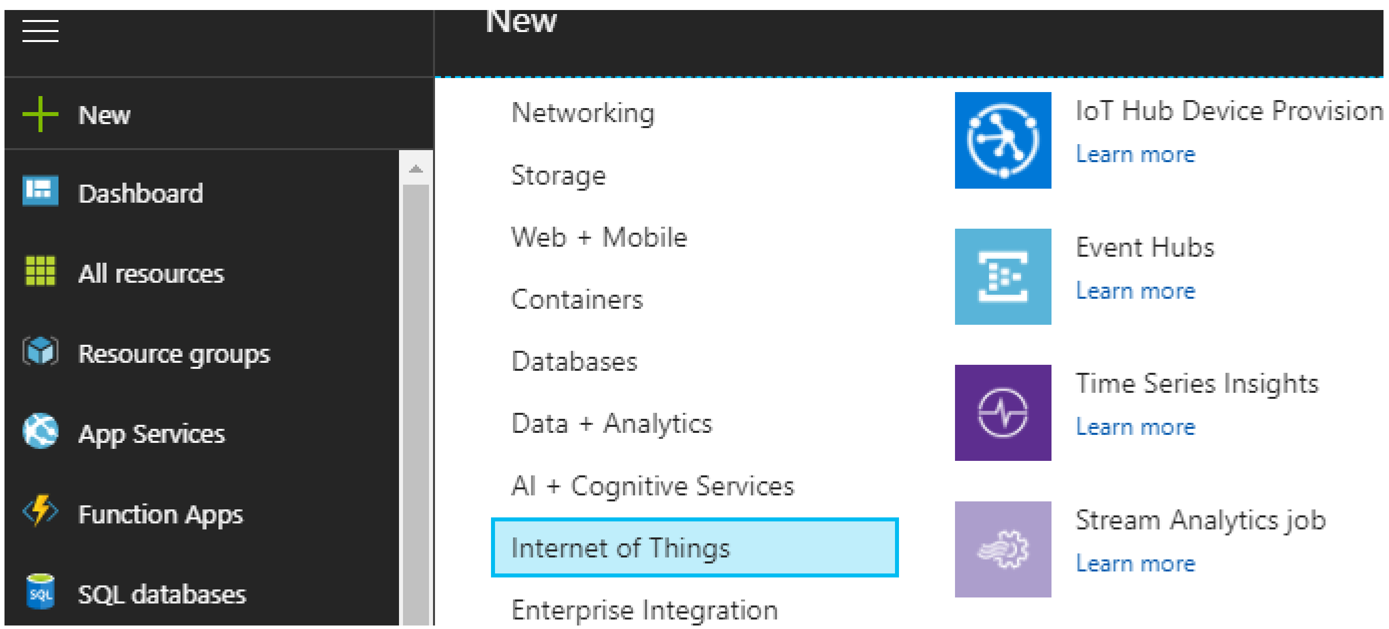
Item	Limits
Throughput units	Default soft limit of 20 TU per subscription
Ingress Throughput	1 MB/s per TU and 1,000 events/s per TU
Total Storage Capacity	No limit (~ 500 TB)
Message Retention	Min 1 day, max 7 days
Partitions	Between 1 and 32 partitions
Max event size	256 KB
Max batchsize	256 KB

Event Hub Sensor Simulation

- Event Hubs is a highly scalable ingestion system that can ingest millions of events per second, enabling an application to process and analyze the massive amounts of data produced by your connected devices and applications.
- Once collected into an event hub, you can transform and store data using any real-time analytics provider or storage cluster.

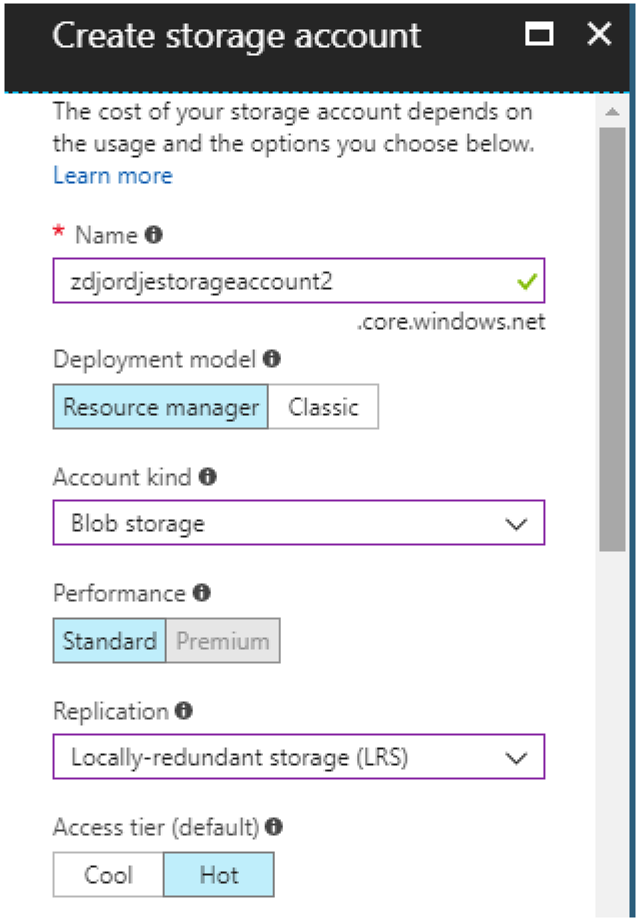
Create an Event Hub in Azure Portal

- In Azure Portal click on + New and select `Internet of Things > Event Hubs`



Create a Storage Account

- Before creating an Azure Event Hub, check whether you have a Storage Account.
- On the left service bar, select Storage accounts and, if you do not have a storage account already, create a new one. As Account kind, select Blob storage:



Create storage account

The cost of your storage account depends on the usage and the options you choose below. Learn more

* Name: zdjordjestorageaccount2 .core.windows.net

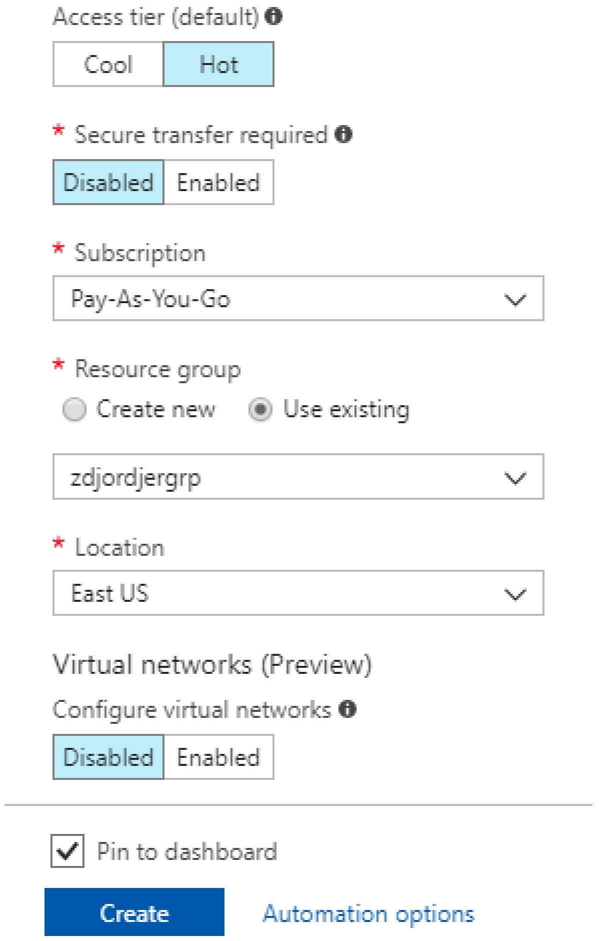
Deployment model: [✓ Resource manager] Classic

Account kind: Blob storage

Performance: [✓ Standard] Premium

Replication: Locally-redundant storage (LRS)

Access tier (default): Cool [✓ Hot]



Access tier (default): Cool [✓ Hot]

* Secure transfer required: [✓ Disabled] Enabled

* Subscription: Pay-As-You-Go

* Resource group: Create new [✓ Use existing]

zdjordjergrp

* Location: East US

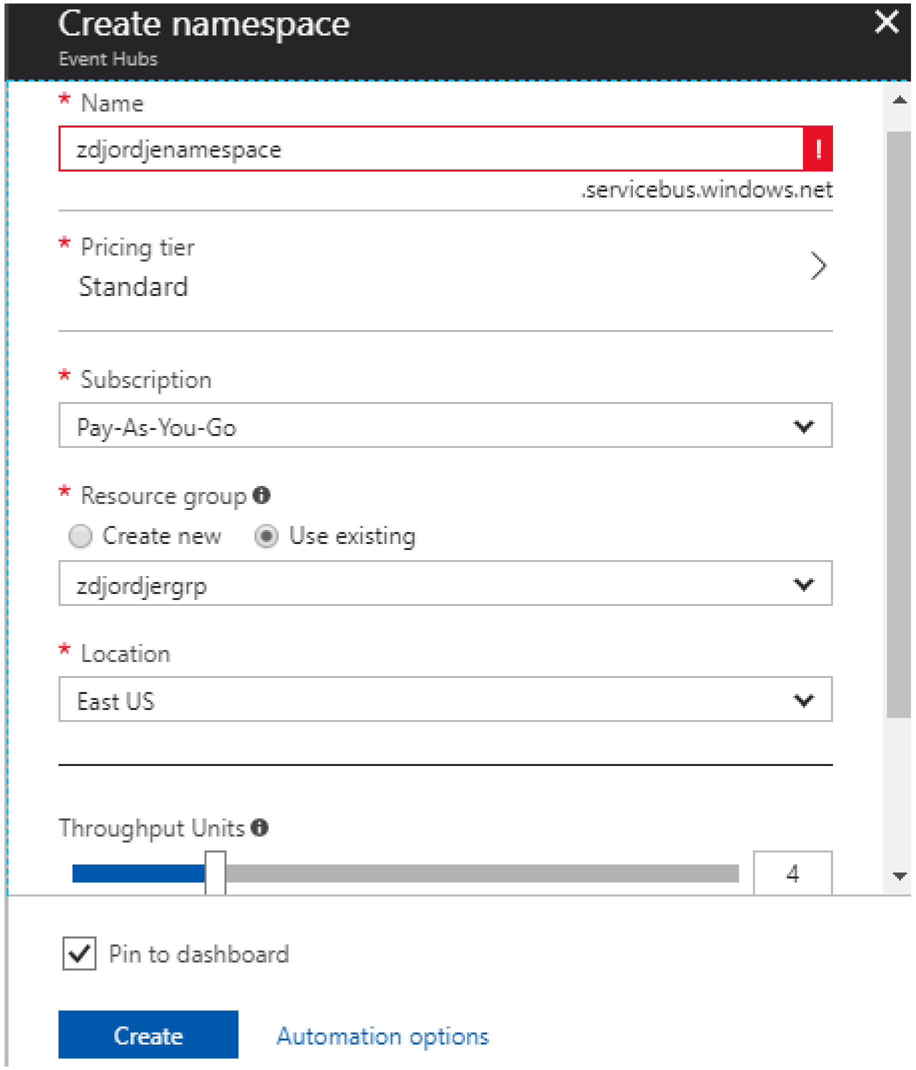
Virtual networks (Preview)

Configure virtual networks: [✓ Disabled] Enabled

[✓] Pin to dashboard

Create IoT namespace

- On Create Namespace blade enter a unique all lowercase name of your new IoT namespace.
- As Service, select Standard.
- Use an existing resource group or create new.
- We do not need much Throughput. Just to demonstrate, move the slider to the right. Number of assigned TU units will show in the box on the right.
- Below TU slide, there is checkbox "Enable auto-inflate". This would enable auto scaling of Event Hub capacity. We do not need that feature now.
- Select `Create.` After a few minutes the namespace will be created.



Create namespace
Event Hubs

* Name
zdjordjenamespace
.servicebus.windows.net

* Pricing tier
Standard

* Subscription
Pay-As-You-Go

* Resource group
( ) Create new [✓ Use existing]
zdjordjergrp

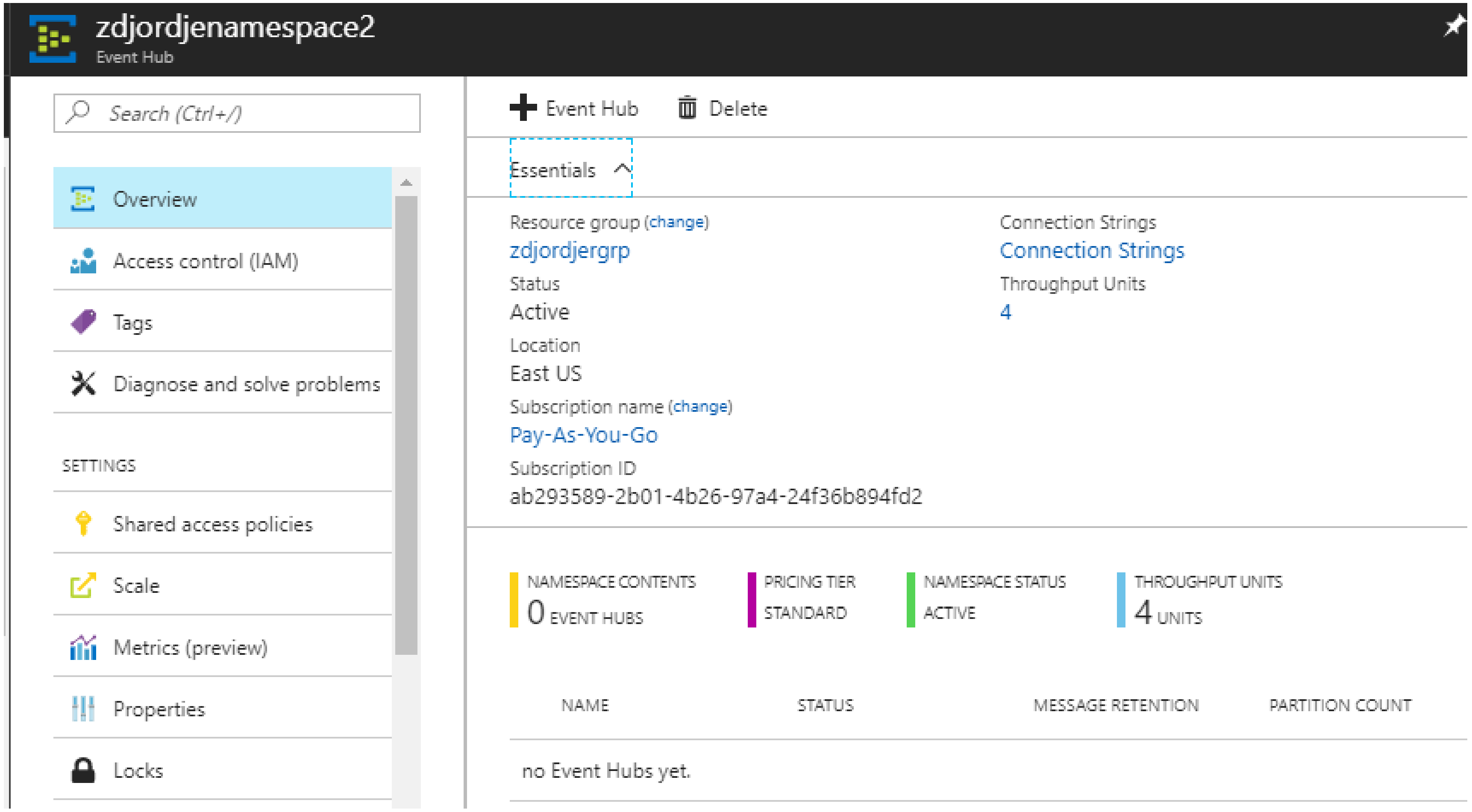
* Location
East US

Throughput Units
4

[✓] Pin to dashboard

Create Event Hub

- On the blade that opens, on the top bar, hit + next to Event Hub



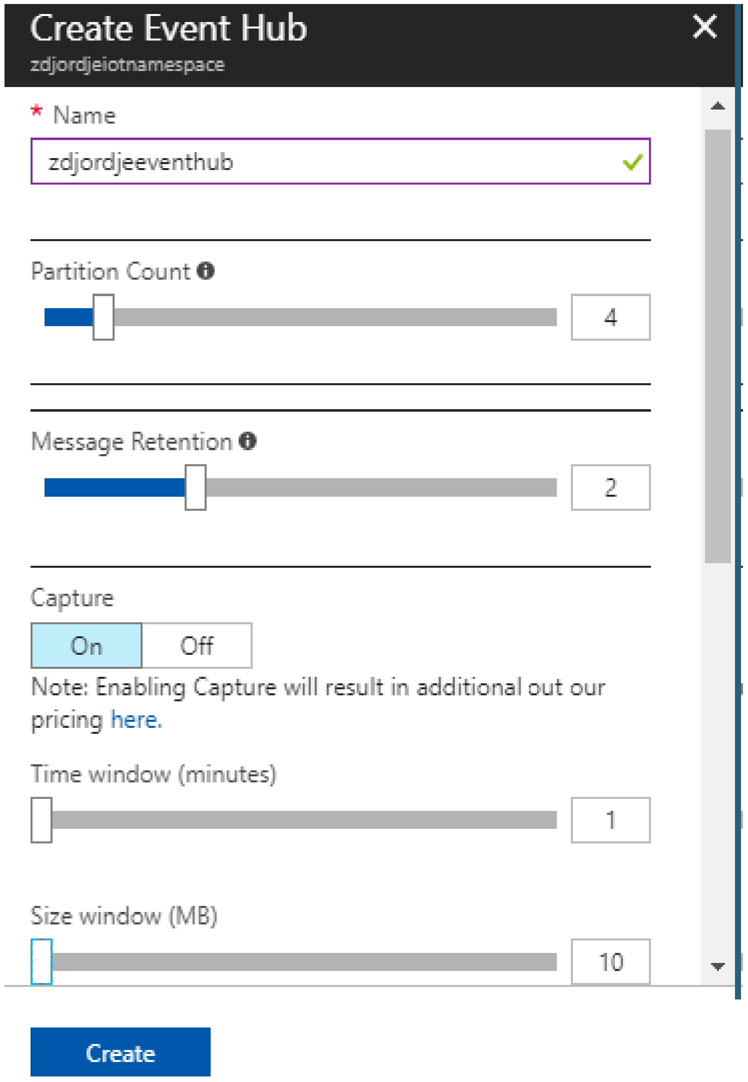
Resource group (change)
zdjordjergrp
Status
Active
Location
East US
Subscription name (change)
Pay-As-You-Go
Subscription ID
ab293589-2b01-4b26-97a4-24f36b894fd2

Connection Strings
Connection Strings
Throughput Units
4

NAMESPACE CONTENTS	PRICING TIER	NAMESPACE STATUS	THROUGHPUT UNITS
0 EVENT HUBS	STANDARD	ACTIVE	4 UNITS

NAME	STATUS	MESSAGE RETENTION	PARTITION COUNT
no Event Hubs yet.			

Parameters of Event Hub



Create Event Hub
zdjordjeiotnamespace

* Name
zdjordjeeventhub

Partition Count
4

Message Retention
2

Capture
[✓ On] Off

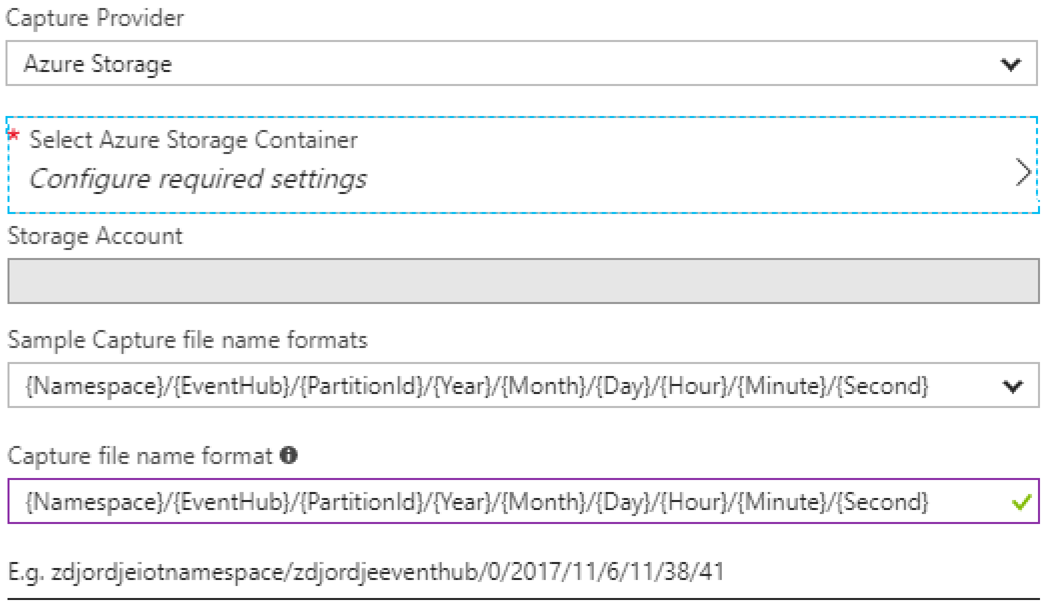
Note: Enabling Capture will result in additional out our pricing here.

Time window (minutes)
1

Size window (MB)
10

Create

- Partition Count can be the default value of 2 or 4 up to 32
- Message retention between 1 and 7 days
- Capture On
- Capture provider: Azure Storage
- Select: Azure Storage Container, *Configure*



Capture Provider
Azure Storage

* Select Azure Storage Container
Configure required settings

Storage Account

Sample Capture file name formats
{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}

Capture file name format
{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}

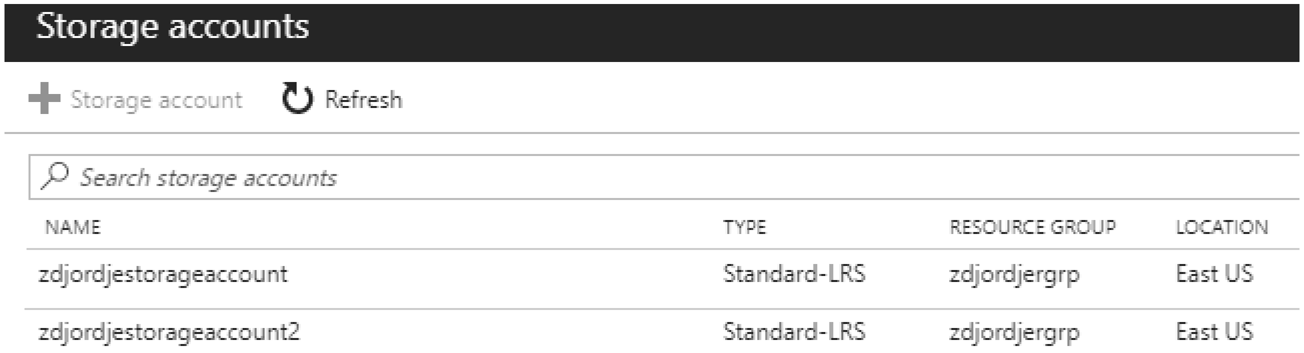
E.g. zdjordjeiotnamespace/zdjordjeeventhub/0/2017/11/6/11/38/41

Configure Event Hubs, Partitions

- Event Hubs leverage the partitioned consumer model. This contrast with the competing consumer model, leveraged by several other message brokering systems. Earlier, we touched on partitions. Partitions are essentially channels through which events flow.
- Adding partitions increases Event Hub's degree of availability by providing additional channels to distribute events. It's important to understand the partitioned consumer model because partitions are immutable. That is, when we configure an Event Hub to leverage a specific number of partitions, we cannot change that amount at a later stage.
- Events published to a partition remain intact and readable for a predetermined time. The time is configurable, the default is 24 hours. In other words, events published to an Event Hub persist for this time, regardless of whether they are read, or the amount of times they are read.
- We define a partition key that is assigned to events. Event Hubs employs a hashing mechanism on the property that we elect partition key in order to determine the partition to which the event will be sent. While it is possible to publish events to specific partitions, it's favorable to allow the SDK to calculate this at runtime in order to achieve balanced distribution of events across partitions.
- Any given event will persist to a single partition only. Thus, the event will be consumed by a single consumer only, given that partitions are subscribed to by single consumers.
- When a consumer subscribes to an Event Hub it acquires a lease on any number of partitions. The exact number is controlled by the Event Hub's SDK and is dependent on the total number of subscribing consumers.
- If an Event Hub has two partitions, when a single consumer is connected, that consumer is granted a lease on both partitions. Should a second consumer subscribe, then the first consumer will lose its lease on one of the partitions, which will in turn be granted to the second consumer. This holds true for any number of partitions. Leases are split evenly among consumers, which is why you must assign an even number of partitions when creating an Event Hub. This is a balancing mechanism, built into the Event Hub's SDK, allowing even distribution when reading events.

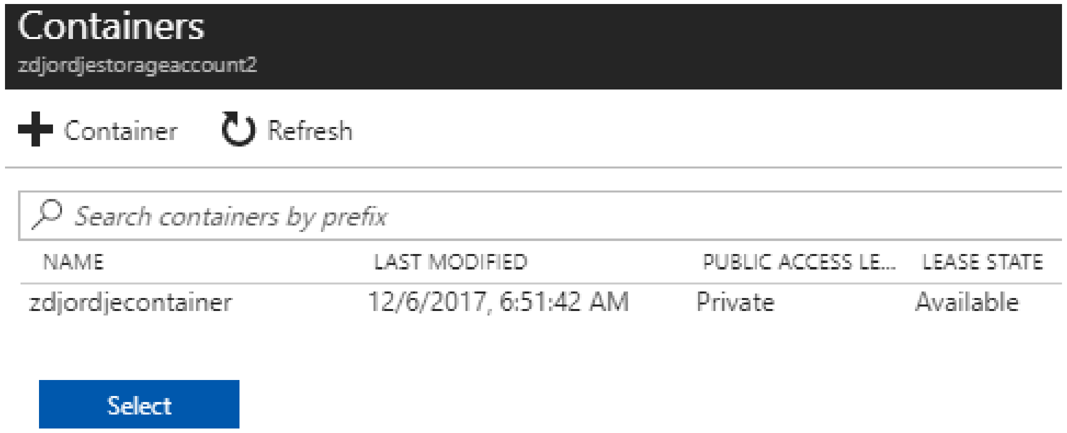
Configure Storage

- On Storage accounts blade, select an existing storage account or create (+) new



NAME	TYPE	RESOURCE GROUP	LOCATION
zdjordjestorageaccount	Standard-LRS	zdjordjergrp	East US
zdjordjestorageaccount2	Standard-LRS	zdjordjergrp	East US

- Once you select an account, the Containers blade appears. Hit + Container

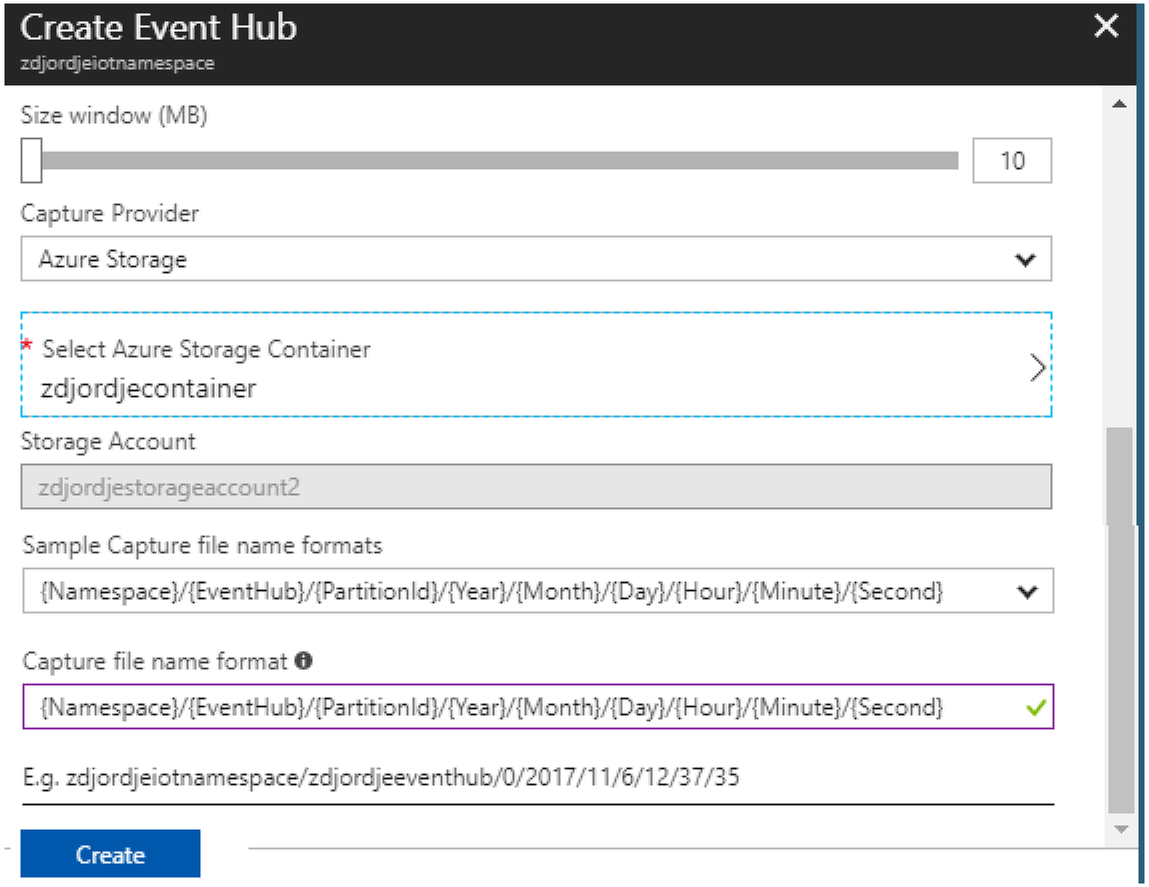


NAME	LAST MODIFIED	PUBLIC ACCESS LE...	LEASE STATE
zdjordjecontainer	12/6/2017, 6:51:42 AM	Private	Available

- At the bottom of Containers blade hit Select. That associates this container with your Event Hb.

Back on Event Hubs blade

- In the Even Hub blade, storage container name and storage account name appear.
- Now you can hit Create and the Event Hub will be created:



Create Event Hub
zdjordjeiotnamespace

Size window (MB)
10

Capture Provider
Azure Storage

* Select Azure Storage Container
zdjordjecontainer

Storage Account
zdjordjestorageaccount2

Sample Capture file name formats
{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}

Capture file name format
{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second} ✓

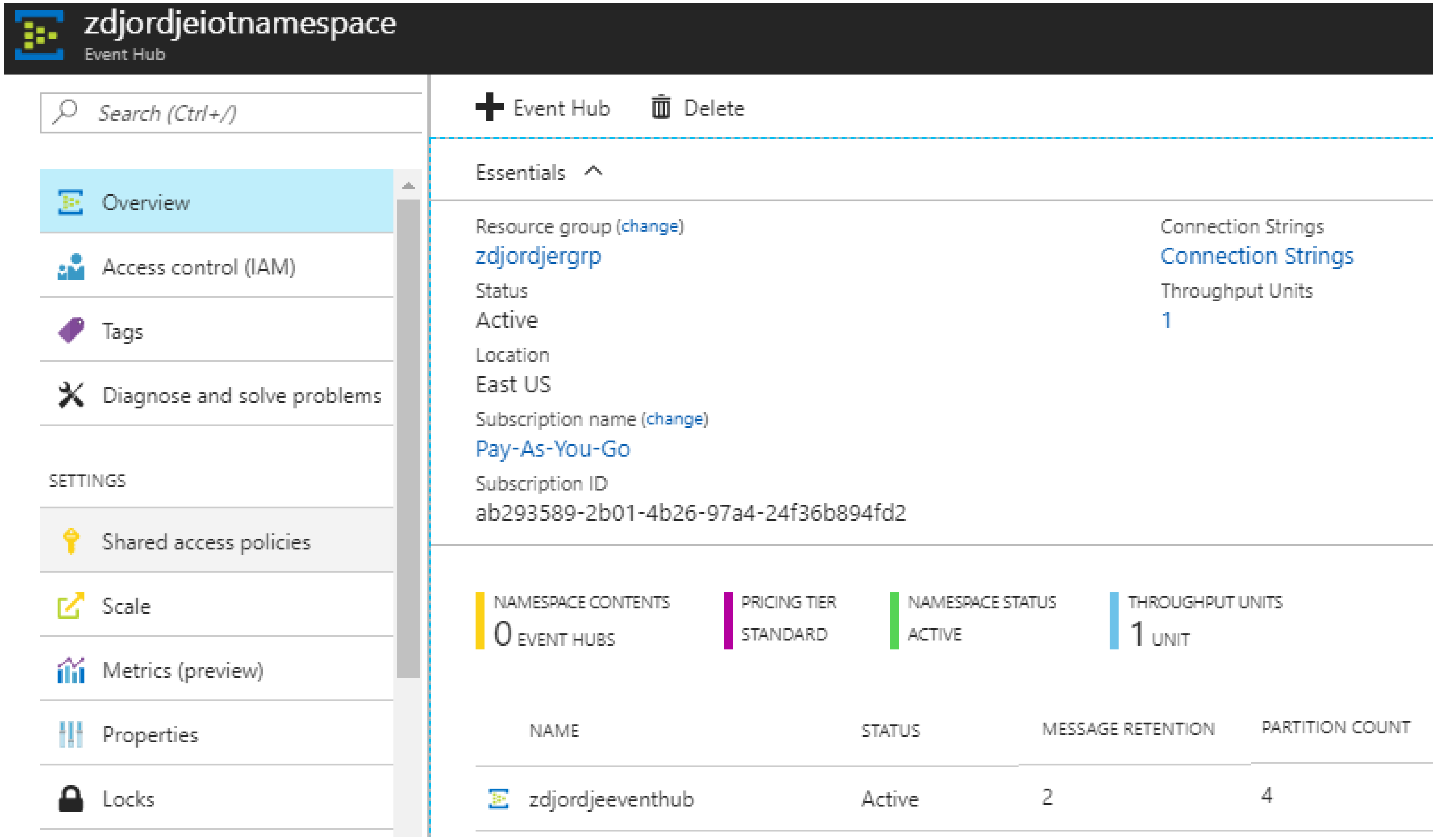
E.g. zdjordjeiotnamespace/zdjordjeeventhub/0/2017/11/6/12/37/35

Storage and Capture

- We have no specific requirement in terms of the volume of traffic that we must manage within our simulation. As a general rule of thumb, if you don't have any specific design requirements and an explicit number of partitions, it's a good idea to start with four.
- Event Hubs Capture allows us to funnel the raw data from our Event Hub to Blob storage for detailed analysis after the fact.
- Event Hubs leverage a bring-your-own-storage mechanism allowing you to plug in a Blob storage account to receive the streaming data. To enable it, take the following steps.
- Toggle Capture mode on, and configure Time and Size windows. These are interval-based settings that determine when Capture events occur, that is, when a segment of our Event Hub will be streamed to our Blob storage account.
- The Capture event itself operates on a first-wins policy. Let's say you configure the Size window to be 1 MG and the Time window to be 10 minutes. Our Event Hub receives data at a rate or 1 MB per minute. Our size window will win here, given that after one minute the size threshold will win, invoking the Capture event. Capture formats are structured intuitively.Select an appropriate file format and customize it if necessary.
- Create button.

Share access policies

- Once new Even Hub appears , select `Shared access policies` on the left



zdjordjeiotnamespace
Event Hub

Resource group (change)
zdjordjergrp
Status
Active
Location
East US
Subscription name (change)
Pay-As-You-Go
Subscription ID
ab293589-2b01-4b26-97a4-24f36b894fd2
Connection Strings
Connection Strings
Throughput Units
1

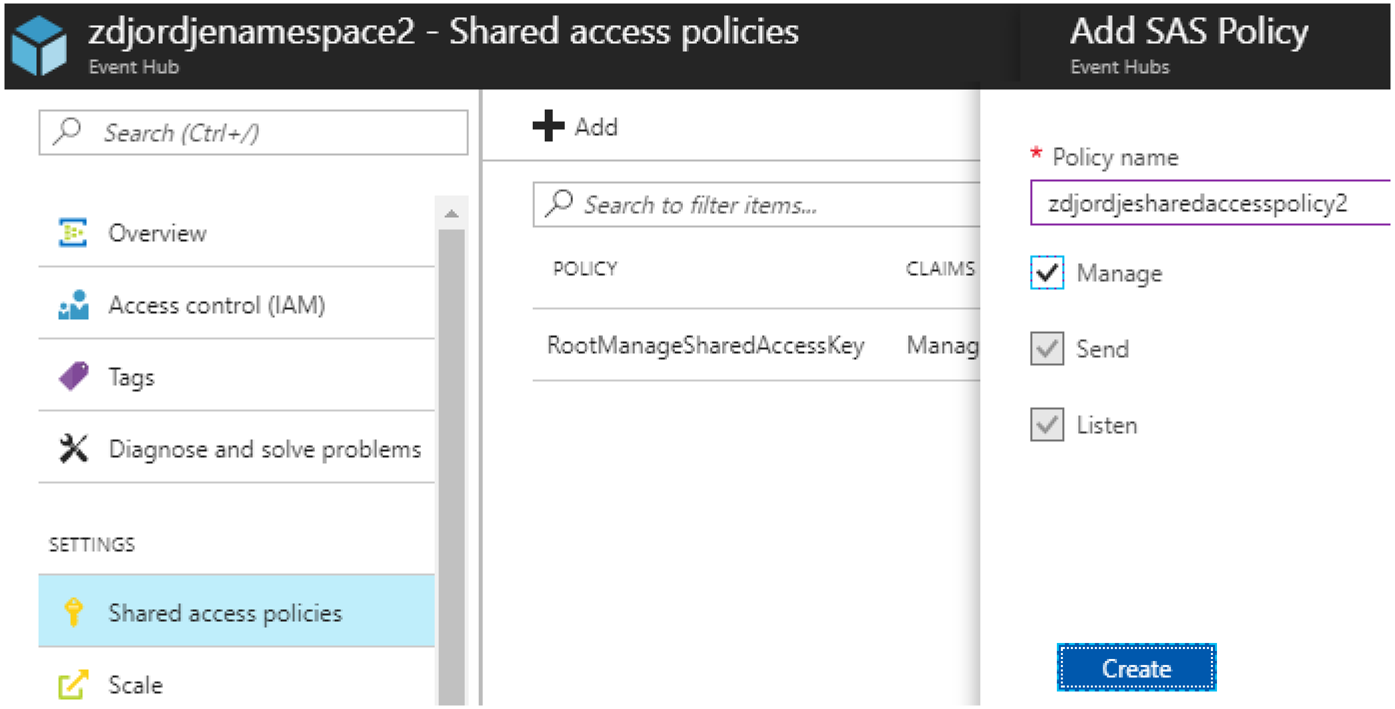
NAMESPACE CONTENTS 0 EVENT HUBS | PRICING TIER STANDARD | NAMESPACE STATUS ACTIVE | THROUGHPUT UNITS 1 UNIT

NAME	STATUS	MESSAGE RETENTION	PARTITION COUNT
zdjordjeeventhub	Active	2	4

- On the next screen select + Shared access policy

Create Shared Access Policy

- It appear to be important to create a named shared access policy.



zdjordjenamespace2 - Shared access policies
Event Hub

POLICY	CLAIMS
RootManageSharedAccessKey	Manag

Add SAS Policy
Event Hubs

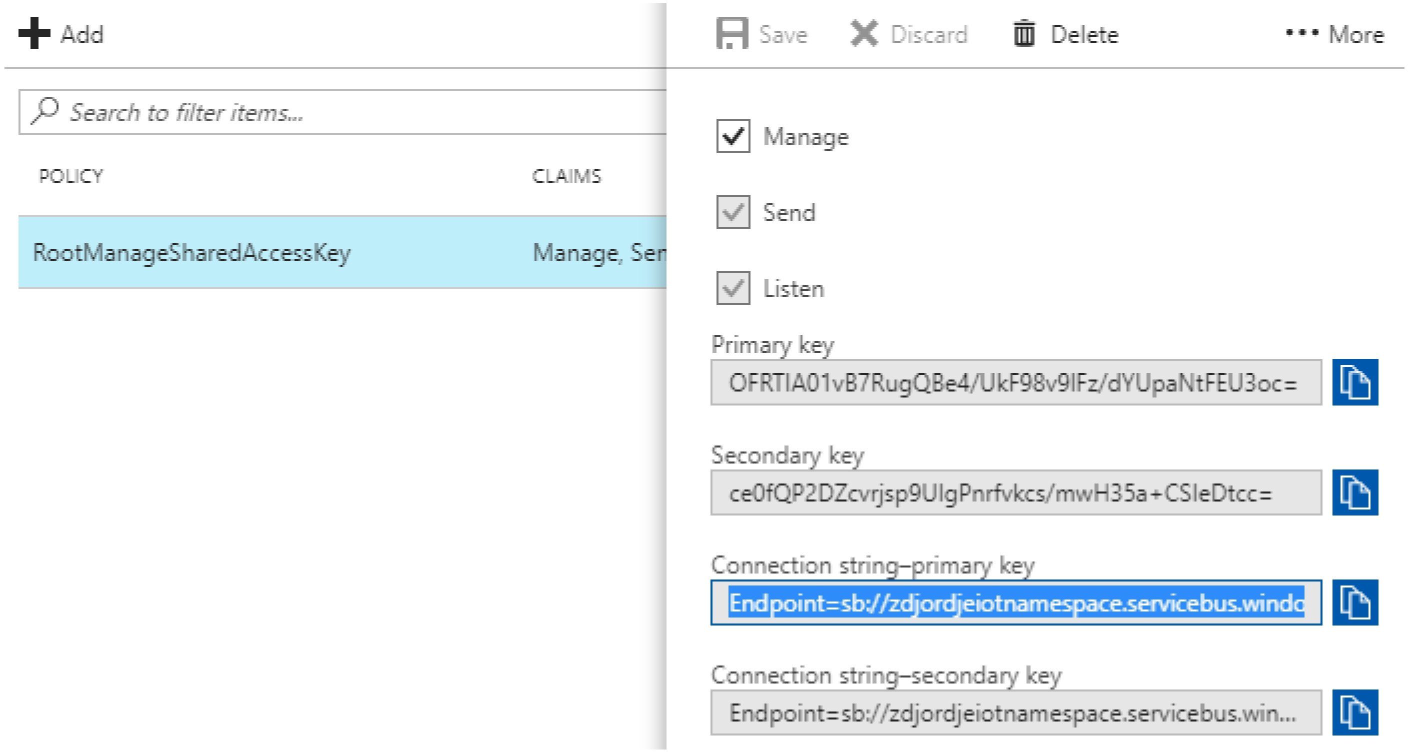
* Policy name: zdjordjesharedaccesspolicy2

✓ Manage
✓ Send
✓ Listen

- Provide unique name and select Manage, Send and Listen.
- Select Create
- This policy will allow you Event Hub to be Managed by API calls, sent data to and listened for incoming data.

Record and Save Primary key and Connection String

- Copy the primary key and connection string-primary key to text file and save. Those values are not going anywhere and you can always fetch them again.



POLICY	CLAIMS
RootManageSharedAccessKey	Manage, Ser

Manage [✓]
Send [✓]
Listen [✓]

Primary key
OFRTIA01vB7RugQBe4/UkF98v9IFz/dYUpaNtFEU3oc=

Secondary key
ce0fQP2DZcvrjsp9UlgPnrfvkcs/mwH35a+CSleDtcc=

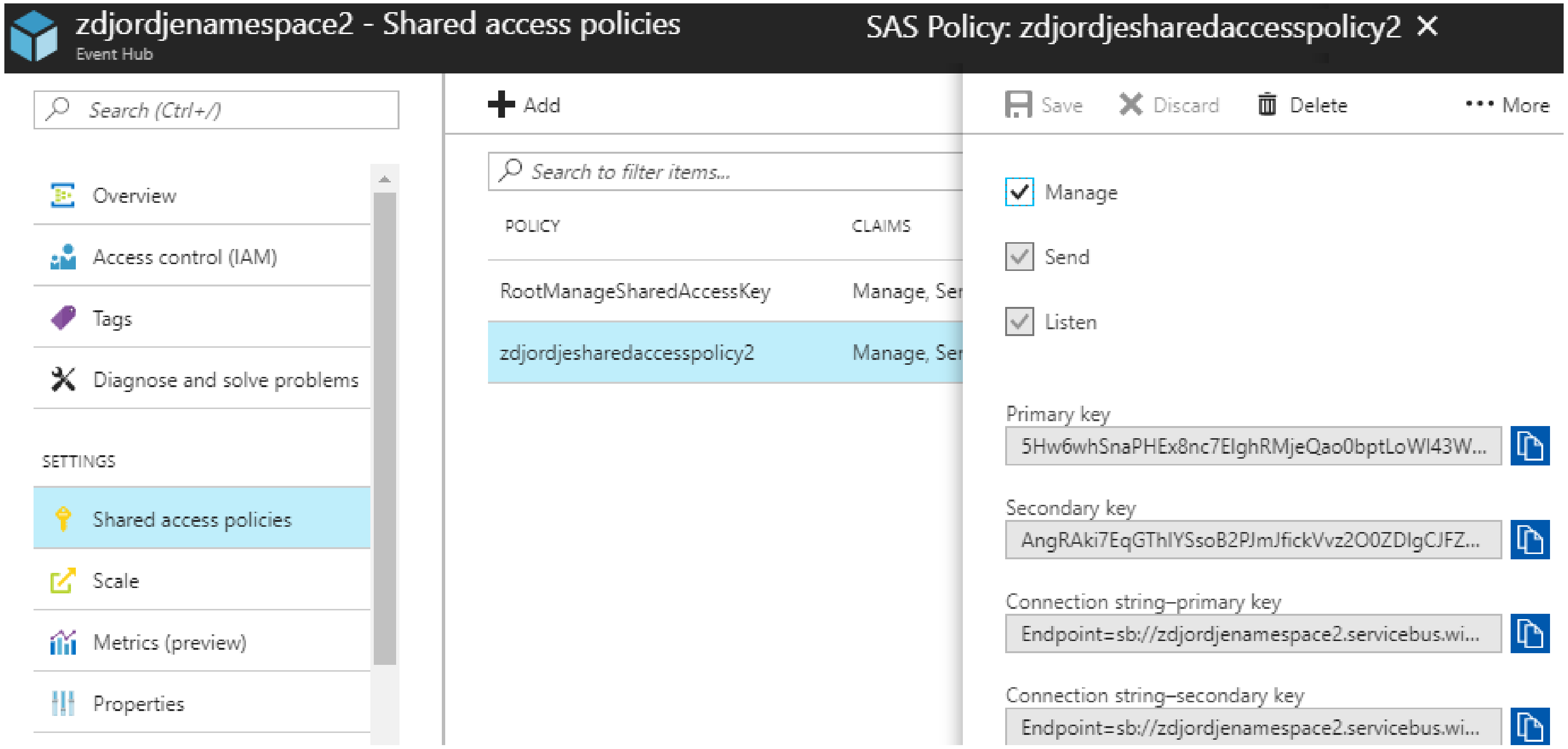
Connection string–primary key
Endpoint=sb://zdjordjeiotnamespace.servicebus.windo

Connection string–secondary key
Endpoint=sb://zdjordjeiotnamespace.servicebus.win...

- Notice, that by the default those keys could be used by API calls to `Manage`, `Send and Listen` to messages to and from the Event Hub

With named Shared access policy

- If you have assigned a names shared access policy to your Event Hub, you could use its Primary key, Connection string-primary key or secondary SAS for access to the Hub.



zdjordjenamespace2 - Shared access policies
Event Hub

SAS Policy: zdjordjesharedaccesspolicy2

POLICY	CLAIMS
RootManageSharedAccessKey	Manage, Ser
zdjordjesharedaccesspolicy2	Manage, Ser

✓ Manage
✓ Send
✓ Listen

Primary key
5Hw6whSnaPHEx8nc7ElghRMjeQao0bptLoWl43W...

Secondary key
AngRAki7EqGThlYSsoB2PJmJfickVvz2O0ZDlgCJFZ...

Connection string–primary key
Endpoint=sb://zdjordjenamespace2.servicebus.wi...

Connection string–secondary key
Endpoint=sb://zdjordjenamespace2.servicebus.wi...

Build Project with Maven

- To build Java clients that would send and receive messages from Event Hub you need:
  - Eclipse (recent JEE version)
  - Java 1.7+
  - Maven 3+
- We build the initial Maven project with

Create a Maven Project

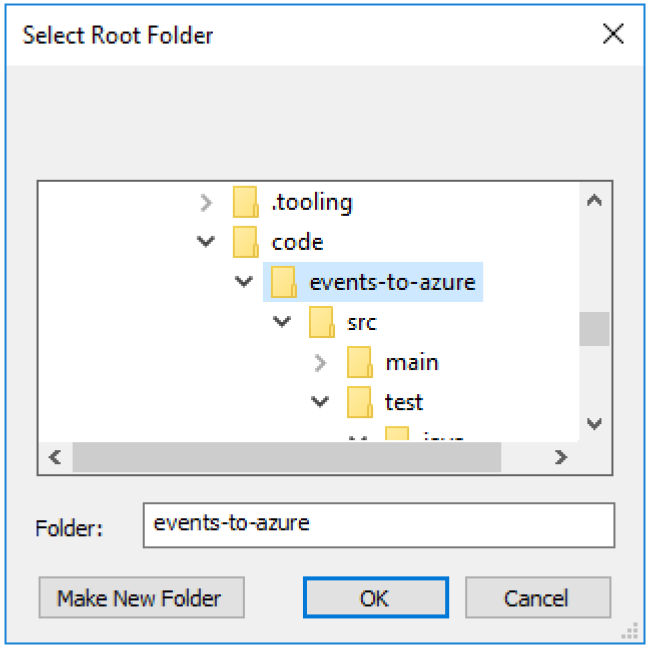
- Open a directory of your choice and test whether Maven is installed:

```
C:\Users\073621\code>mvn --version
Apache Maven 3.5.2 (138edd61fd100ec658bfa2d307c43b76940a5d7d; 2017-10-
18T02:58:13-05:00)
Maven home: C:\Programs\maven-3.5.2\bin\..
```

- Then run the following Maven command, all on one line:

```
mvn archetype:generate -DgroupId=com.mck.app -DartifactId=events-to-azure
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

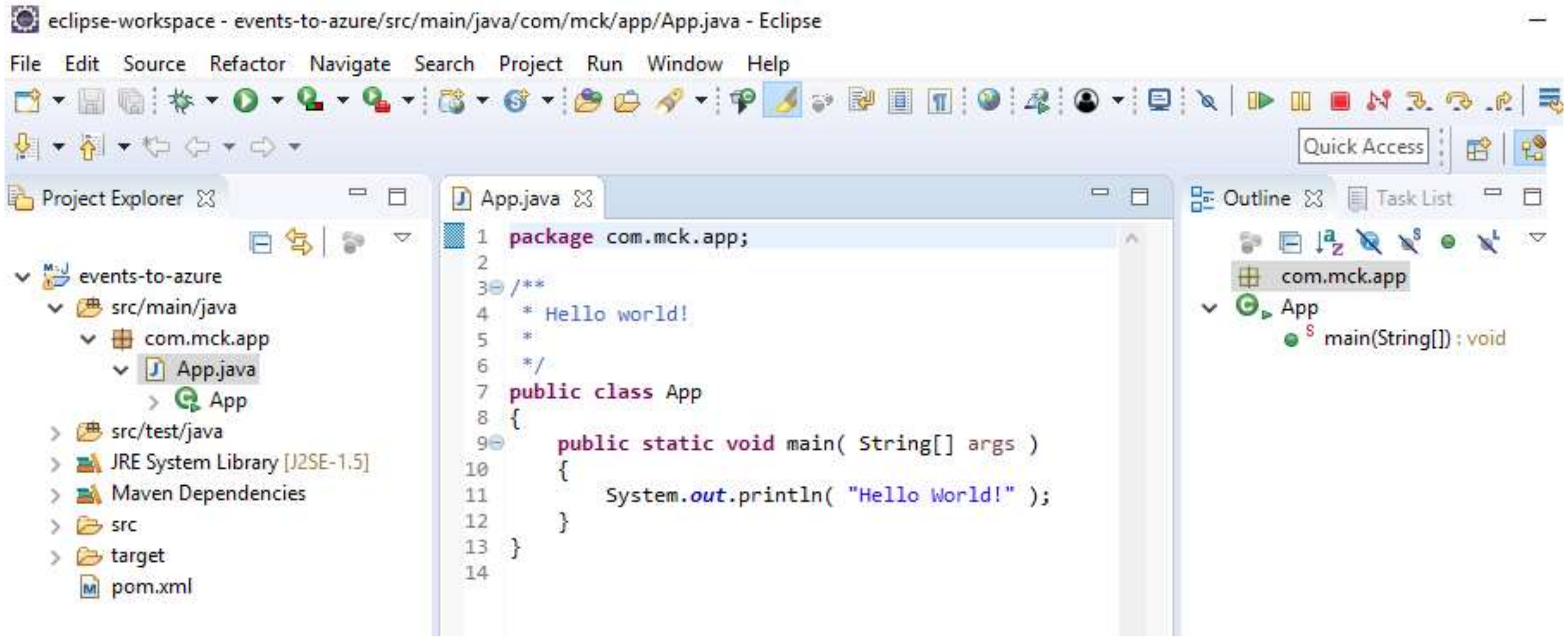
- The result will be a directory `events-to-azure` (in my case; you can set your `artifiactId` to any value) which contains a basic skeleton of a Java project:
- To bring that project to Eclipse, open Eclipse and do:
- `File > New > Import > Existing Maven Project`
- On Maven Project wizard, select `Browse`, to the right of `Root Directory`
- Navigate to the directory Maven directory just created. Hit OK, Accept selected pom.xml, click Finish.


Select Root Folder

Folder: events-to-azure

`events-to-azure` Project

- Actions so far resulted with new Java (Maven) project in Eclipse with a simple class App.java. Note that parameters of our Maven command: `artifactId` got adopted as the project name, and `groupId` got adopted as the package name `com.mck.app`



```java
package com.mck.app;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }
}
```

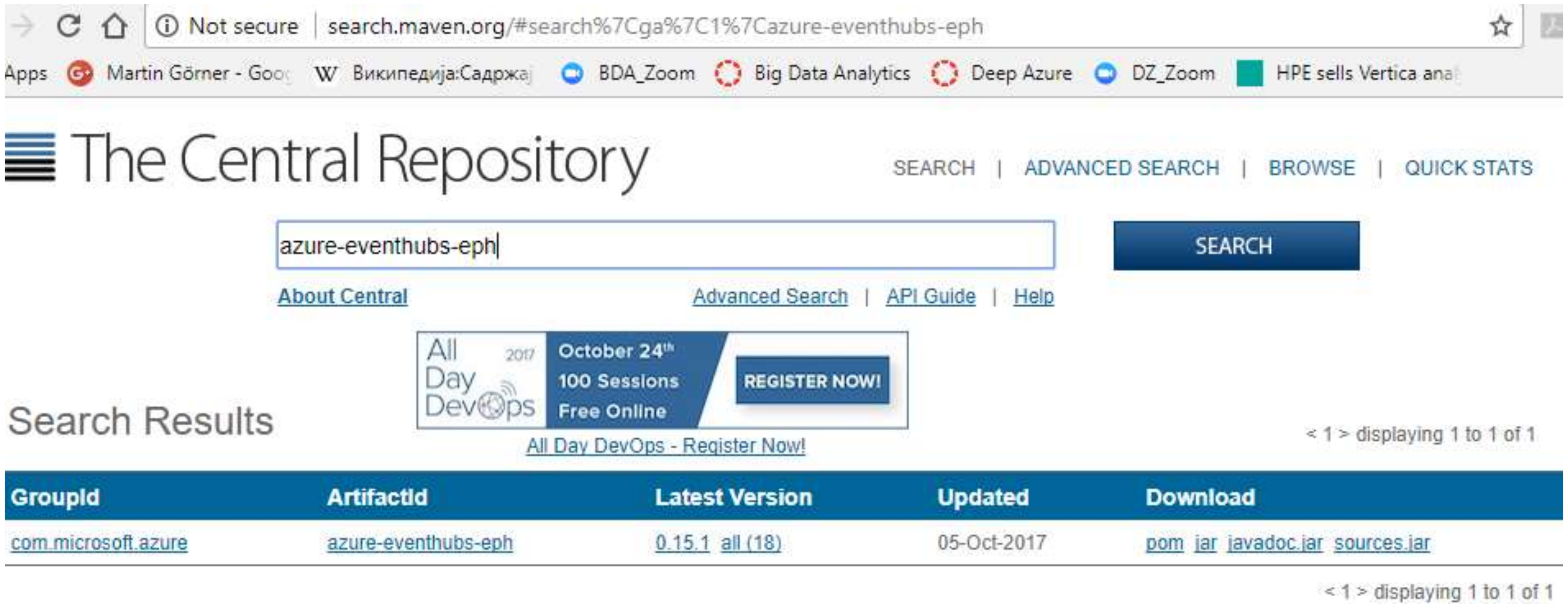
Maven `pom.xml` file

- What Maven does, compile, deletes, fetches dependencies, etc. is all determined by the content of file `pom.xml`. In our case, that file is coming from a preconfigured Maven project called: `maven-archetype-quickstart`, and is rather generic.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mck.app</groupId>
  <artifactId>events-to-azure</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>events-to-azure</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Additional Dependences

- Since we read MS documentation, we know that we need `azure-eventhubs-eph` API added to the project.
- We need to find out what is the latest or perhaps appropriate version of that API.
- Maven has a repository and a search utility at `search.maven.org/#search`



Not secure | search.maven.org/#search%7Cga%7C1%7Cazure-eventhubs-eph

The Central Repository

SEARCH | ADVANCED SEARCH | BROWSE | QUICK STATS

azure-eventhubs-eph

Search Results

< 1 > displaying 1 to 1 of 1

GroupId	ArtifactId	Latest Version	Updated	Download
com.microsoft.azure	azure-eventhubs-eph	0.15.1 all (18)	05-Oct-2017	pom jar javadoc.jar sources.jar

< 1 > displaying 1 to 1 of 1

- Enter the name of the API in the search field and you will get the name, i.e. `groupId, ArtifactId, Latest Version, and` ability to download pom.xml file, jar itself, Javadoc and source code. Our API has version 0.15.1

In pom.xml, add `azure-eventhubs-eph 0.15.1`

- In Eclipse, we open `pom.xml` file and within the `<dependencies>` …`</dependencies>` element add a new dependency pointing to `azure-eventhubs` API.

```
<dependencies>
   <dependency>
     <groupId>junit</groupId>
     <artifactId>junit</artifactId>
     <version>3.8.1</version>
     <scope>test</scope>
   </dependency>
  <dependency>
     <groupId>com.microsoft.azure</groupId>
     <artifactId>azure-eventhubs-eph</artifactId>
     <version>0.15.1</version>
 </dependency>
 </dependencies>
```

- Save `pom.xml` file. Open Maven Dependencies folder in your project.

Additional Dependencies for slf4j

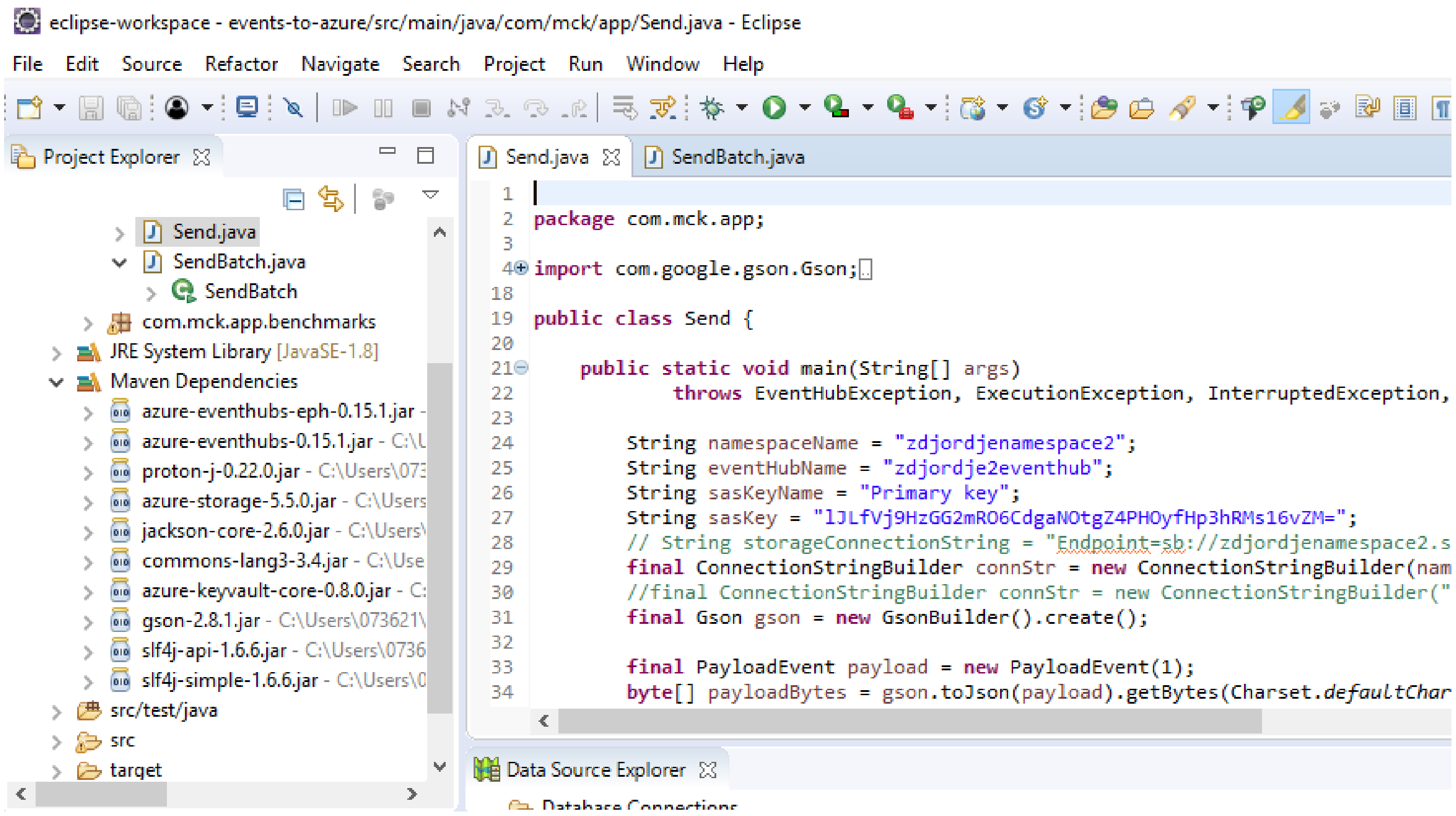
- It appears that `log4j` or `slf4j` requires two more dependencies. (At least, I needed them). Slf4j stands for Simple Logging Façade 4 Java and is supposed to simplify your work with log4j. Anyway, add these 2 dependencies inside `<dependencies>..</dependencies>` element in `pom.xml`:

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.6.6</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.6.6</version>
</dependency>
```

- Now, save pom.xml and take a look at Maven Dependencies folder in your Eclipse project

Maven Dependencies Folder

- Reading the instructions you provided in pom.xml and following the trail of dependencies, Maven brought from its repository various jar-s that your project needs.



eclipse-workspace - events-to-azure/src/main/java/com/mck/app/Send.java - Eclipse

```
Send.java
SendBatch.java
  SendBatch
com.mck.app.benchmarks
JRE System Library [JavaSE-1.8]
Maven Dependencies
  azure-eventhubs-eph-0.15.1.jar -
  azure-eventhubs-0.15.1.jar - C:\U
  proton-j-0.22.0.jar - C:\Users\073
  azure-storage-5.5.0.jar - C:\Users
  jackson-core-2.6.0.jar - C:\Users\
  commons-lang3-3.4.jar - C:\Use
  azure-keyvault-core-0.8.0.jar - C:
  gson-2.8.1.jar - C:\Users\073621\
  slf4j-api-1.6.6.jar - C:\Users\0736
  slf4j-simple-1.6.6.jar - C:\Users\0
src/test/java
src
target
```

```java

package com.mck.app;

import com.google.gson.Gson;

public class Send {

    public static void main(String[] args)
            throws EventHubException, ExecutionException, InterruptedException,

        String namespaceName = "zdjordjenamespace2";
        String eventHubName = "zdjordje2eventhub";
        String sasKeyName = "Primary key";
        String sasKey = "lJLfVj9HzGG2mRO6CdgaNOtgZ4PHOyfHp3hRMs16vZM=";
        // String storageConnectionString = "Endpoint=sb://zdjordjenamespace2.s
        final ConnectionStringBuilder connStr = new ConnectionStringBuilder(nam
        //final ConnectionStringBuilder connStr = new ConnectionStringBuilder("
        final Gson gson = new GsonBuilder().create();

        final PayloadEvent payload = new PayloadEvent(1);
        byte[] payloadBytes = gson.toJson(payload).getBytes(Charset.defaultChar
```

Class `com.mck.app.Send`

- We will demonstrate a class Send.java that will construct a string, simulating an event, or message generated by a telemetry device, establish connection with Event Hub in Azure and send the message to the Event Hub.
- We have enabled Storage option on our Event Hub, meaning that messages will not only be stored in hub’s partitions, but will rather be also deposited for safe keeping and later analysis into a container in a Storage Blob.
- Subsequently we will demonstrate how we could write a modification of this class, SendBatch, which will send events in burst.
- Finally, we will write a client that will retrieve messages from the Event Hub and print them for our reading pleasure.

Sand.java

```
package com.mck.app;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.microsoft.azure.eventhubs.ConnectionStringBuilder;
import com.microsoft.azure.eventhubs.EventData;
import com.microsoft.azure.eventhubs.EventHubClient;
import com.microsoft.azure.eventhubs.PartitionSender;
import com.microsoft.azure.eventhubs.EventHubException;
import java.io.IOException;
import java.nio.charset.Charset;
import java.time.Instant;
import java.util.Random;
import java.util.concurrent.ExecutionException;
import java.util.function.BiConsumer;

public class Send {
    public static void main(String[] args)
            throws EventHubException, ExecutionException, InterruptedException, IOException {
           String namespaceName = "zdjordjenamespace2";
           String eventHubName = "zdjordje2eventhub";
           String sasKeyName = "Primary key";
           String sasKey = "lJLfVj9HzGG2mRO6CdgaNOtgZ4PHOyfHp3hRMs16vZM=";
           // String storageConnectionString =
"Endpoint=sb://zdjordjenamespace2.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccess
Key;SharedAccessKey=39Ft6vRcGi1SA4rFnj2SWVSDR6EM9MWT5zMFcUtOor4=";
        //final ConnectionStringBuilder connStr = new ConnectionStringBuilder(namespaceName,
eventHubName, sasKeyName, sasKey);
        //final ConnectionStringBuilder connStr = new
ConnectionStringBuilder("Endpoint=sb://zdjordjenamespace2.servicebus.windows.net/;SharedAccessKeyNam
e=RootManageSharedAccessKey;SharedAccessKey=39Ft6vRcGi1SA4rFnj2SWVSDR6EM9MWT5zMFcUtOor4=");
        final ConnectionStringBuilder connStr = new
ConnectionStringBuilder("Endpoint=sb://zdjordjenamespace2.servicebus.windows.net/;SharedAccessKeyNam
e=zdjordjesharedaccesspolicy;SharedAccessKey=lJLfVj9HzGG2mRO6CdgaNOtgZ4PHOyfHp3hRMs16vZM=;EntityPath
=zdjordje2eventhub");
```

Sand.java

```
final ConnectionStringBuilder connStr = new
ConnectionStringBuilder("Endpoint=sb://zdjordjenamespace2.servicebus.windows.net/;SharedAccessK
eyName=zdjordjesharedaccesspolicy;SharedAccessKey=lJLfVj9HzGG2mRO6CdgaNOtgZ4PHOyfHp3hRMs16vZM=;
EntityPath=zdjordje2eventhub");
        final Gson gson = new GsonBuilder().create();

        final PayloadEvent payload = new PayloadEvent(1);
        byte[] payloadBytes = gson.toJson(payload).getBytes(Charset.defaultCharset());
        final EventData sendEvent = new EventData(payloadBytes);

        final EventHubClient ehClient =
EventHubClient.createFromConnectionStringSync(connStr.toString());;
        PartitionSender sender = null;

        try {
            // senders
            // Type-1 - Basic Send - not tied to any partition
            ehClient.send(sendEvent).get();

            // Advanced Sends
            // Type-2 - Send using PartitionKey - all Events with Same partitionKey will land
on the Same Partition
            final String partitionKey = "partitionTheStream";
            ehClient.sendSync(sendEvent, partitionKey);

            // Type-3 - Send to a Specific Partition
            sender = ehClient.createPartitionSenderSync("0");
            sender.sendSync(sendEvent);
            System.out.println(Instant.now() + ": Send Complete...");
           System.in.read();
```

Sand.java

```
} finally {
            if (sender != null)
                sender.close().whenComplete(new BiConsumer<Void, Throwable>() {
                    public void accept(Void t, Throwable u) {
                        if (u != null) {
                            // wire-up this error to diagnostics infrastructure
                            System.out.println(String.format("closing failed with
error: %s", u.toString()));
                        }
                        try {
                            ehClient.closeSync();
                        } catch (EventHubException sbException) {
                            // wire-up this error to diagnostics infrastructure
                            System.out.println(String.format("closing failed with
error: %s", sbException.toString()));
                        }
                    }
                }).get();
            else if (ehClient != null)
                ehClient.closeSync();
        }
    }
```

Sand.java

```
    /**
     * actual application-payload, ex: a telemetry event
     */
    static final class PayloadEvent {
        PayloadEvent(final int seed) {
            this.id = "telemetryEvent1-critical-eventid-2345" + seed;
            this.strProperty = "This is a sample payloadEvent, which could be
wrapped using eventdata and sent to eventhub." +
                    " None of the payload event properties will be looked-at by
EventHubs client or Service." +
                    " As far as EventHubs service/client is concerted, it is
plain bytes being sent as 1 Event.";
            this.longProperty = seed * new Random().nextInt(seed);
            this.intProperty = seed * new Random().nextInt(seed);
        }

        public String id;
        public String strProperty;
        public long longProperty;
        public int intProperty;
    }
}
```

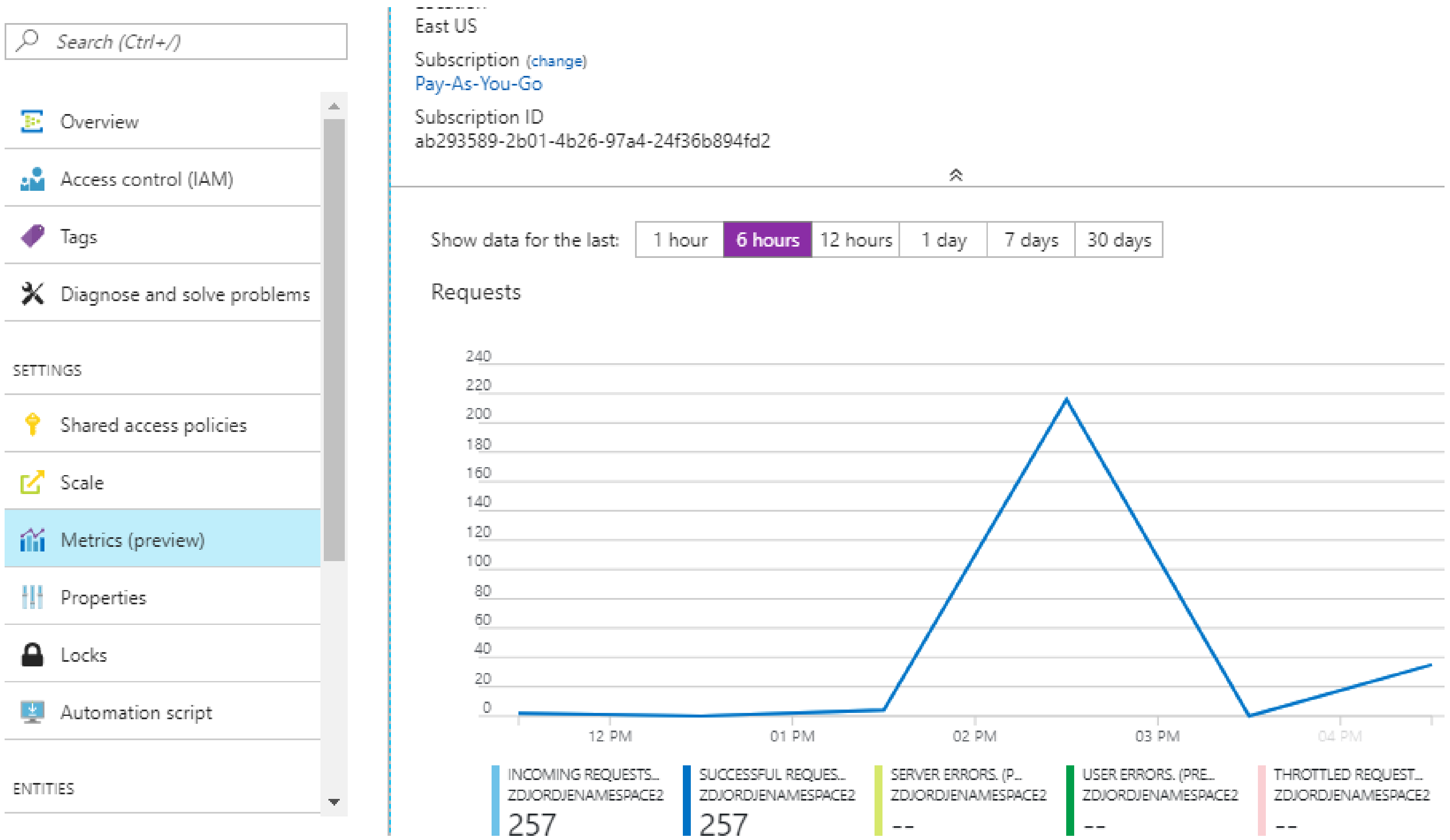
Run as Java Application

- Class Send.java is run as a Java Application and produces an unremarkable console output:

```
187 [main] INFO com.microsoft.azure.eventhubs.Timer - Starting ScheduledThreadPoolExecutor with
coreThreadPoolSize: 4
245 [Thread-0] INFO com.microsoft.azure.eventhubs.MessagingFactory - starting reactor instance.
341 [Thread-0] INFO com.microsoft.azure.eventhubs.amqp.ReactorHandler - reactor.onReactorInit
2064 [Thread-0] INFO com.microsoft.azure.eventhubs.amqp.ConnectionHandler - onConnectionRemoteOpen:
hostname[zdjordjenamespace2.servicebus.windows.net:5671, b48d58f1d4d54cc98f721a8d422ad0b0_G29]
2116 [Thread-0] INFO com.microsoft.azure.eventhubs.amqp.ReceiveLinkHandler - linkName[cbs:receiver],
localSource[Source{address='$cbs', durable=NONE, expiryPolicy=SESSION_END, timeout=0, dynamic=false,
dynamicNodeProperties=null, distributionMode=null, filter=null, defaultOutcome=null, outcomes=null,
capabilities=null}]
2227 [Thread-0] INFO com.microsoft.azure.eventhubs.amqp.SessionHandler - entityName[cbs-session],
sessionIncCapacity[1048576], sessionOutgoingWindow[2147483647]
2365 [Thread-0] INFO com.microsoft.azure.eventhubs.amqp.SendLinkHandler - linkName[cbs:sender], 2818
[Thread-0] INFO com.microsoft.azure.eventhubs.amqp.SessionHandler - entityName[zdjordje2eventhub],
sessionIncCapacity[1048576], sessionOutgoingWindow[2147483647]
2934 [Thread-0] INFO com.microsoft.azure.eventhubs.amqp.SendLinkHandler -
linkName[70eb03_0b0_G29_1512601359753], remoteTarget[Target{address='zdjordje2eventhub', durable=NONE,
expiryPolicy=SESSION_END, timeout=0, dynamic=false, dynamicNodeProperties=null, capabilities=null}]
3855 [Thread-0] INFO com.microsoft.azure.eventhubs.amqp.SessionHandler -
entityName[zdjordje2eventhub/Partitions/0], sessionIncCapacity[1048576],
sessionOutgoingWindow[2147483647]
3975 [Thread-0] INFO com.microsoft.azure.eventhubs.amqp.SendLinkHandler -
linkName[a14284_0b0_G29_1512601360788], remoteTarget[Target{address='zdjordje2eventhub/Partitions/0',
durable=NONE, expiryPolicy=SESSION_END, timeout=0, dynamic=false, dynamicNodeProperties=null,
capabilities=null}]
2017-12-06T23:02:41.195Z: Send Complete...
```

- However,if we run it many time and then visit, Azure portal, we will see that Event Hub was hit by messages.
- We can also navigate to our storage container (storage account > Blob > Container) and we will see that messages are also saved in the Container.

Select Metrics on Event Hub



East US

Subscription (change)
Pay-As-You-Go

Subscription ID
ab293589-2b01-4b26-97a4-24f36b894fd2

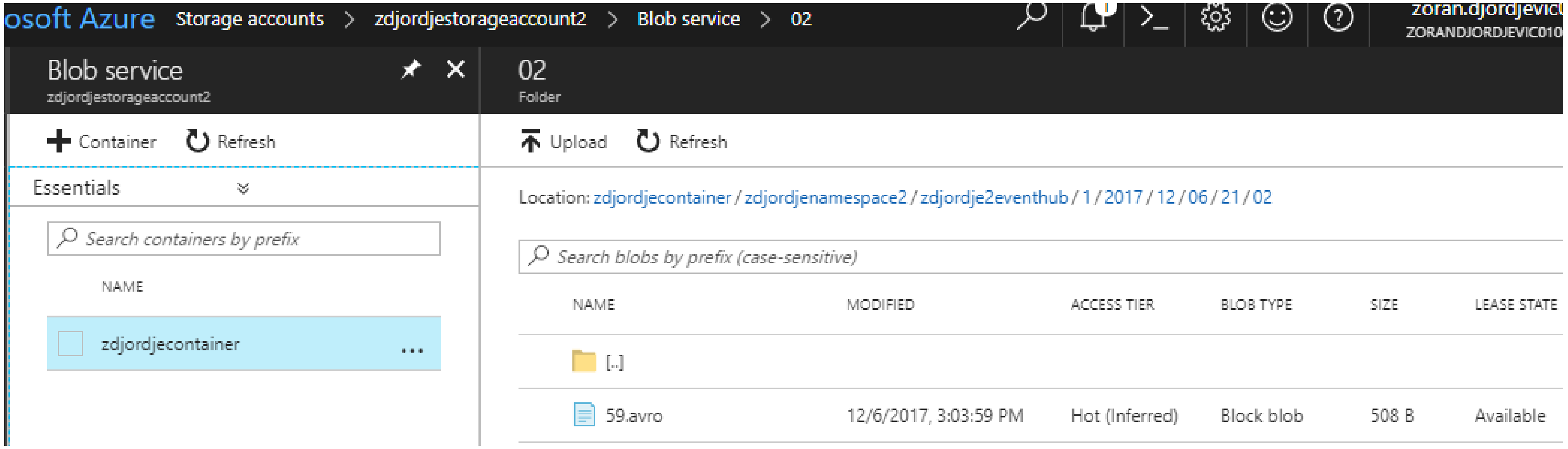
Show data for the last: 1 hour [✓ 6 hours] 12 hours 1 day 7 days 30 days

Requests

INCOMING REQUESTS... ZDJORDJENAMESPACE2	SUCCESSFUL REQUES... ZDJORDJENAMESPACE2	SERVER ERRORS. (P... ZDJORDJENAMESPACE2	USER ERRORS. (PRE... ZDJORDJENAMESPACE2	THROTTLED REQUEST... ZDJORDJENAMESPACE2
257	257	--	--	--

Navigate to Stored Messages

- With some patience you can navigate through many levels of directories used when messages are stored and eventually discover atomic messages that contain actual fake telemetry data:


osoft Azure
Storage accounts > zdjordjestorageaccount2 > Blob service > 02
zoran.djordjevic
ZORANDJORDJEVIC010
Blob service
zdjordjestorageaccount2
Container
Refresh
Essentials
Search containers by prefix
02
Folder
Upload
Refresh
Search blobs by prefix (case-sensitive)

NAME
zdjordjecontainer

Location: zdjordjecontainer / zdjordjenamespace2 / zdjordje2eventhub / 1 / 2017 / 12 / 06 / 21 / 02

NAME	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE	LEASE STATE
[..]					
59.avro	12/6/2017, 3:03:59 PM	Hot (Inferred)	Block blob	508 B	Available

- Curiously enough, messages are stored in (Hadoop) Avro format.
- **Avro** is a remote procedure call and data serialization framework developed within Apache's Hadoop project. It uses JSON for defining data types and protocols, and serializes data in a compact binary format. Its primary use is in Apache Hadoop, where it can provide both a serialization format for persistent data, and a wire format for communication between Hadoop nodes, and from client programs to the Hadoop services.

Note on ConnectionStringBuilder

- Pay attention to two ways how connection string is built and them used to instantiate `EventHubClient` object:

```
String namespaceName = "zdjordjenamespace2";
     String eventHubName = "zdjordje2eventhub";
     String sasKeyName = "RootManageSharedAccessKey";
     String sasKey = "lJLfVj9HzGG2mRO6CdgaNOtgZ4PHOyfHp3hRMs16vZM=";
//final ConnectionStringBuilder connStr = new ConnectionStringBuilder( namespaceName, eventHubName, sasKeyName, sasKey);
final ConnectionStringBuilder connStr = new ConnectionStringBuilder(
"Endpoint=sb://zdjordjenamespace2.servicebus.windows.net/;SharedAccess
KeyName=zdjordjesharedaccesspolicy;SharedAccessKey=lJLfVj9HzGG2mRO6Cdg
aNOtgZ4PHOyfHp3hRMs16vZM=;EntityPath=zdjordje2eventhub");

          final Gson gson = new GsonBuilder().create();
          final PayloadEvent payload = new PayloadEvent(1);
          byte[] payloadBytes =
gson.toJson(payload).getBytes(Charset.defaultCharset());
          final EventData sendEvent = new EventData(payloadBytes);
          final EventHubClient ehClient =
EventHubClient.createFromConnectionStringSync(connStr.toString());;
          PartitionSender sender = null;
```

Note on `ConnectionStringBuilder`

- In the first case, we invoke a constructor of ConnectionStringBuilder with four arguments:

```
final ConnectionStringBuilder connStr = new
ConnectionStringBuilder( namespaceName, eventHubName, sasKeyName,
sasKey);
```

- Alternatively, we can construct the connection string as a String and pass it to an overloaded constructor:

```
final ConnectionStringBuilder connStr = new
ConnectionStringBuilder(
"Endpoint=sb://zdjordjenamespace2.servicebus.windows.net/;SharedA
ccessKeyName=zdjordjesharedaccesspolicy;SharedAccessKey=lJLfVj9Hz
GG2mRO6CdgaNOtgZ4PHOyfHp3hRMs16vZM=;EntityPath=zdjordje2eventhub"
);
```

- Note that the connection string ends with the Event Hub name:

```
EntityPath=zdjordje2eventhub
```

ReceiveUsingOffset.java

- Class ReceiveUsingOffset create its client object in a similar fashion, fetches events from Event Hub and prints them on console:

```
 final ConnectionStringBuilder connStr = new
ConnectionStringBuilder("Endpoint=sb://zdjordjenamespace2.servicebus.window
s.net/;SharedAccessKeyName=zdjordjesharedaccesspolicy;SharedAccessKey=lJLfV
j9HzGG2mRO6CdgaNOtgZ4PHOyfHp3hRMs16vZM=;EntityPath=zdjordje2eventhub");
        final EventHubClient ehClient =
EventHubClient.createFromConnectionStringSync(connStr.toString());

        final EventHubRuntimeInformation eventHubInfo =
ehClient.getRuntimeInformation().get();
        final String partitionId = eventHubInfo.getPartitionIds()[0]; //
get first partition's id

        final PartitionReceiver receiver =
ehClient.createEpochReceiverSync(
                EventHubClient.DEFAULT_CONSUMER_GROUP_NAME,
                partitionId,
                PartitionReceiver.START_OF_STREAM,
                false,
                1);

        try {
            Iterable<EventData> receivedEvents = receiver.receiveSync(100);
```

Output of ReceiveUsingOffset.java

```
Offset: 153200, SeqNo: 714, EnqueueTime: 2017-12-06T20:06:08.080Z
Message Payload: {"id":"telemetryEvent1-critical-eventid-
23456","strProperty":"I am a mock telemetry event from
JavaClient.","longProperty":30,"intProperty":18}
Offset: 153416, SeqNo: 715, EnqueueTime: 2017-12-06T20:06:08.080Z
Message Payload: {"id":"telemetryEvent1-critical-eventid-
23457","strProperty":"I am a mock telemetry event from
JavaClient.","longProperty":35,"intProperty":42}
Offset: 153632, SeqNo: 716, EnqueueTime: 2017-12-06T20:06:08.080Z
Message Payload: {"id":"telemetryEvent1-critical-eventid-
23458","strProperty":"I am a mock telemetry event from
JavaClient.","longProperty":48,"intProperty":32}
Offset: 153848, SeqNo: 717, EnqueueTime: 2017-12-06T20:06:08.080Z
Message Payload: {"id":"telemetryEvent1-critical-eventid-
23459","strProperty":"I am a mock telemetry event from
JavaClient.","longProperty":9,"intProperty":72}
Offset: 154064, SeqNo: 718, EnqueueTime: 2017-12-06T20:06:08.080Z
ReceivedBatch Size: 8
```