Technical Documentation

# 1. Azure Event Hub Onboarding

Azure Event Hub is a highly scalable data streaming platform and event ingestion service. It can ingest millions of events per second and stream them to multiple consumers.

## Key Components

- Namespace: Logical container for Event Hubs.  
- Event Hubs: The actual streaming endpoints.  
- Partitions: Used to enable parallel processing and load distribution.  
- Consumer Groups: Enable multiple consumers to read independently.

## Configuration Details

|  |  |
| --- | --- |
| Configuration | Value/Guidelines |
| Namespace | eh-<project-name>-<env> |
| Event Hub Name | eh-<topic-name>-<env> |
| Partition Strategy | Based on throughput and parallel processing needs. Typical range: 4–32 |
| Retention Period | Default: 1 day; Configurable up to 90 days based on use case |
| Throughput Units / Capacity | Auto-Inflate enabled (up to X units) |
| Capture | Enabled (optional); writes to Azure Blob or ADLS for archiving |
| Access | Shared Access Policies (Send, Listen, Manage) |
| Schema Registry | Enabled if Avro/Protobuf formats are used |
| Private Endpoint | Enabled for secure communication |
| Point of Contact | CloudOps team: cloudops@company.com |

# 2. Kafka to Azure Event Hub Integration

Enable seamless data replication or ingestion from Kafka topics to Azure Event Hub for further processing or analytics on Azure.

## Option A: Confluent Kafka Connect with MirrorSourceConnector

Setup Overview:  
- Use Confluent’s Kafka Connect with MirrorSourceConnector.  
- Configure Azure Event Hub as a Kafka-compatible endpoint.

Sample Connector Config:

{  
 "name": "mirror-source-connector",  
 "connector.class": "org.apache.kafka.connect.mirror.MirrorSourceConnector",  
 "source.cluster.alias": "source-cluster",  
 "target.cluster.alias": "eventhub",  
 "topics": "topic1,topic2",  
 "tasks.max": "4"  
}

Pros:

* - Managed solution with high reliability.  
  - Native topic and offset sync capabilities.  
  - Scalable and supports failover.

Cons:

* - Requires Confluent Kafka Connect setup.  
  - Slightly higher latency due to mirroring process.  
  - Limited flexibility for transformations.

## Option B: Kafka Consumer to Event Hub Producer (Spark or Python)

Spark-Based Producer:  
- Consume from Kafka using spark.readStream.  
- Write to Event Hub using Event Hubs connector for Spark.

Python-Based Producer:

from kafka import KafkaConsumer  
from azure.eventhub import EventHubProducerClient, EventData  
  
consumer = KafkaConsumer('topic1', bootstrap\_servers='broker1:9092')  
producer = EventHubProducerClient.from\_connection\_string("Endpoint=sb://...")  
  
for msg in consumer:  
 event\_data = EventData(msg.value)  
 producer.send\_batch([event\_data])

Pros:

* - Full control over transformations and logic.  
  - Easy to deploy in Python-based workflows.

Cons:

* - More error-prone, requires custom retry/error handling.  
  - Less scalable unless parallelized (e.g., with Spark Structured Streaming).

# 3. Azure Databricks Onboarding

## Cluster Configuration

|  |  |
| --- | --- |
| Config Item | Value |
| Cluster Type | Job/All-Purpose |
| Databricks Runtime | 12.2 LTS ML or latest supported |
| Worker Type | Standard\_D4\_v2 (modifiable) |
| Autoscaling | Enabled |
| Init Scripts | For installing required libraries |
| Cluster Mode | Standard / High Concurrency (for REST API access) |
| Tagging | Environment, Owner, Project |

## Required Libraries

- Azure Event Hub Connector: com.microsoft.azure:azure-eventhubs-spark\_2.12:2.3.22  
- Delta Lake: Included in runtime  
- Kafka (optional): org.apache.spark:spark-sql-kafka-0-10\_2.12:<spark-version>  
- Monitoring: Add Prometheus or custom metrics if needed

## Authentication

- Azure Key Vault-backed secrets for connection strings  
- Managed Identity (preferred) for access to Event Hub and storage

## Point of Contact

|  |  |
| --- | --- |
| Area | Contact |
| Cluster Configuration | Data Engineering Team (dataeng@company.com) |
| Library Installations | Platform Engineering |
| IAM and Secrets | Security Team |

# 4. Event Hub to Databricks Integration

Ingest events from Azure Event Hub into Databricks for real-time processing or transformation using Spark Structured Streaming.

## Method: Event Hub Connector for Apache Spark

ehConf = {  
 'eventhubs.connectionString': sc.\_jvm.org.apache.spark.eventhubs.EventHubsUtils.encrypt(connection\_str),  
 'eventhubs.consumerGroup': "$Default"  
}  
  
streamingDF = (  
 spark.readStream  
 .format("eventhubs")  
 .options(\*\*ehConf)  
 .load()  
)  
  
transformed = streamingDF.selectExpr("CAST(body AS STRING) as message")  
  
query = (  
 transformed.writeStream  
 .format("delta")  
 .outputMode("append")  
 .option("checkpointLocation", "/mnt/checkpoints/eh\_to\_delta")  
 .start("/mnt/datalake/raw/events")  
)

## Considerations

- Schema Evolution: Use Delta Lake for schema enforcement  
- Checkpointing: Required for fault tolerance  
- Monitoring: Leverage Databricks metrics UI or logs to monitor  
- Scalability: Scale cluster based on Event Hub throughput  
- Backpressure: Enable Structured Streaming backpressure settings