# 趨勢科技：基於 MongoDB Change Stream 建立事件驅動系統

William Luo
Staff Software Engineer
Trend Micro

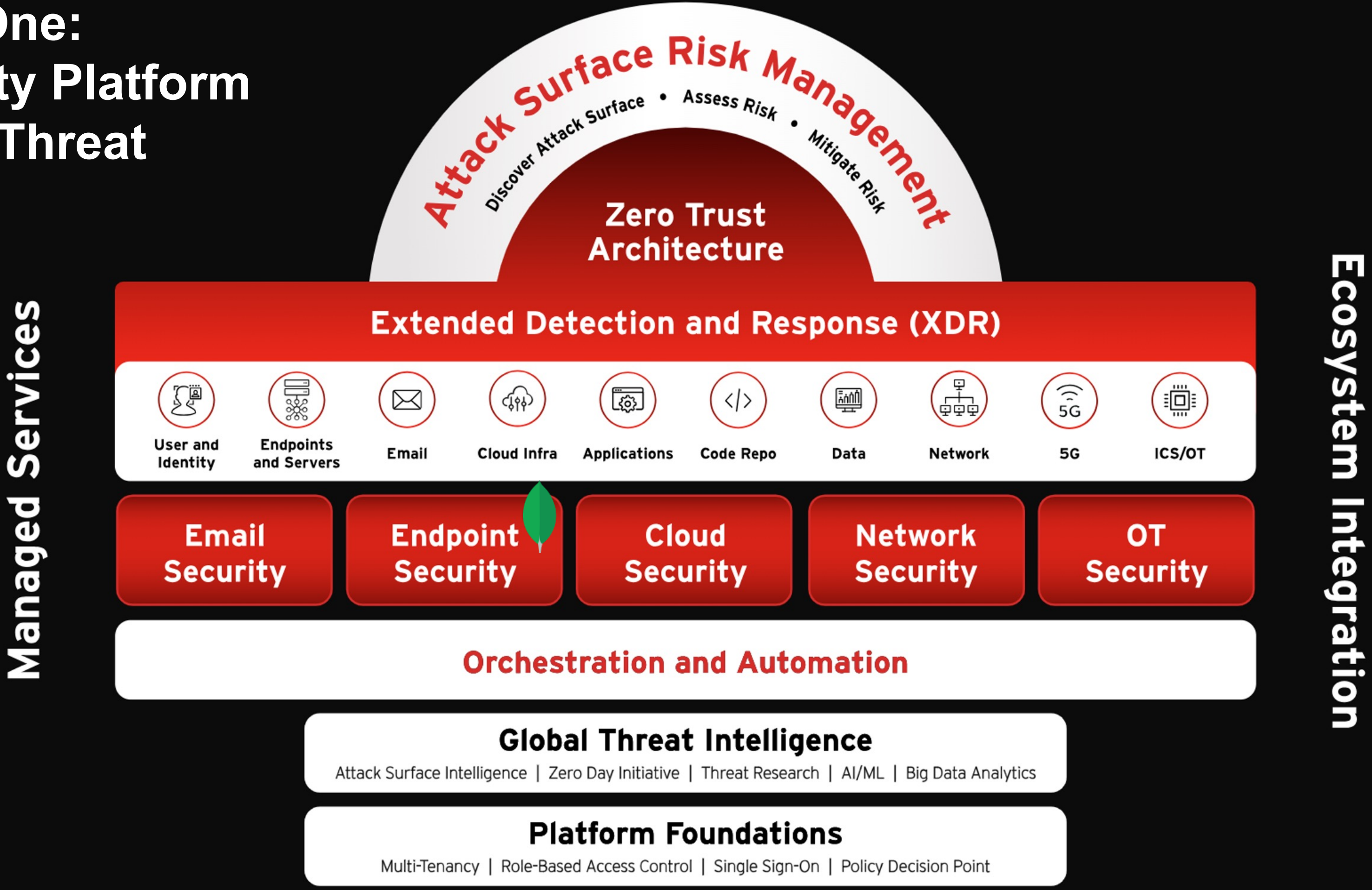
Jim Chi
Sr. Staff Software
Engineer Trend Micro

## Agenda

- Leverage MongoDB Change Stream to build an Event-Driven System
- Introduction to MongoDB Change Stream
- Performance improvements in Change Stream from MongoDB 4.0 to MongoDB 6.0
- Most Common Scenarios
- Best Practices

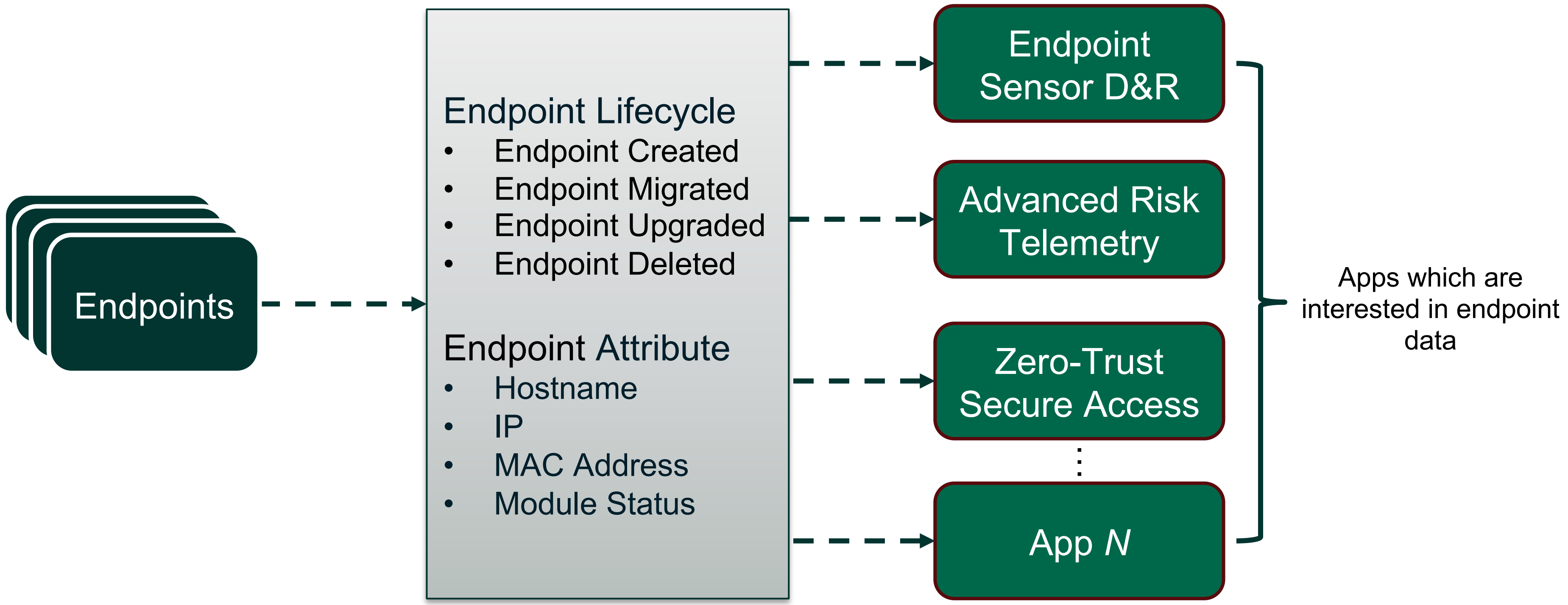
## Leverage MongoDB Change Stream to build an Event-Driven System

# Trend Vision One: A Cybersecurity Platform for Enterprise Threat Defense

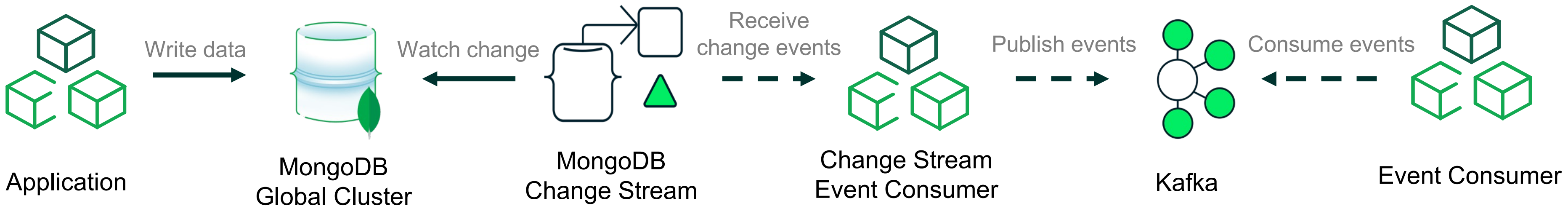

Attack Surface Risk Management
Discover Attack Surface • Assess Risk • Mitigate Risk
Zero Trust Architecture
Extended Detection and Response (XDR)
User and Identity
Endpoints and Servers
Email
Cloud Infra
Applications
Code Repo
Data
Network
5G
ICS/OT
Email Security
Endpoint Security
Cloud Security
Network Security
OT Security
Orchestration and Automation
Global Threat Intelligence
Attack Surface Intelligence | Zero Day Initiative | Threat Research | AI/ML | Big Data Analytics
Platform Foundations
Multi-Tenancy | Role-Based Access Control | Single Sign-On | Policy Decision Point
Managed Services
Ecosystem Integration

TREND MICRO™

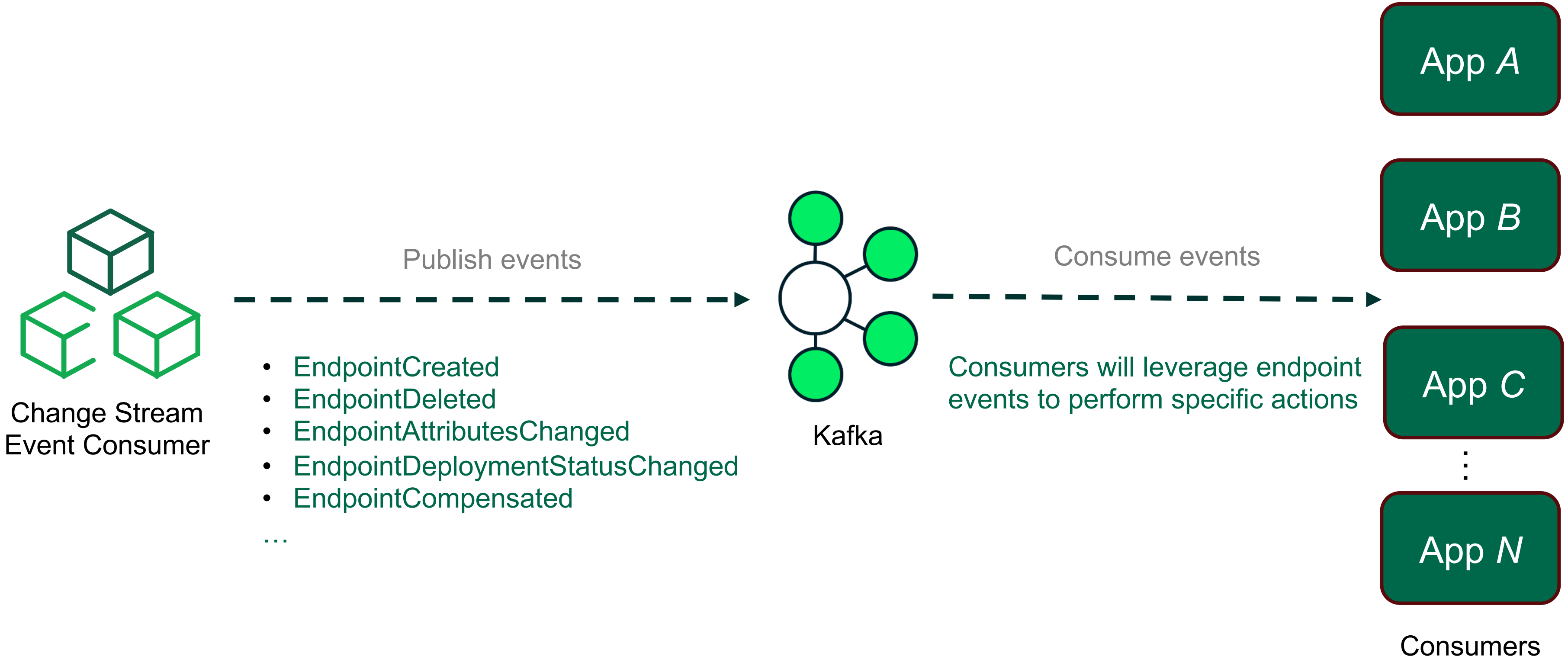
# Share Endpoint Data/Lifecycle between Apps



# Event Driven Architecture



# Event Driven Architecture



- EndpointCreated
- EndpointDeleted
- EndpointAttributesChanged
- EndpointDeploymentStatusChanged
- EndpointCompensated

…

Consumers will leverage endpoint events to perform specific actions

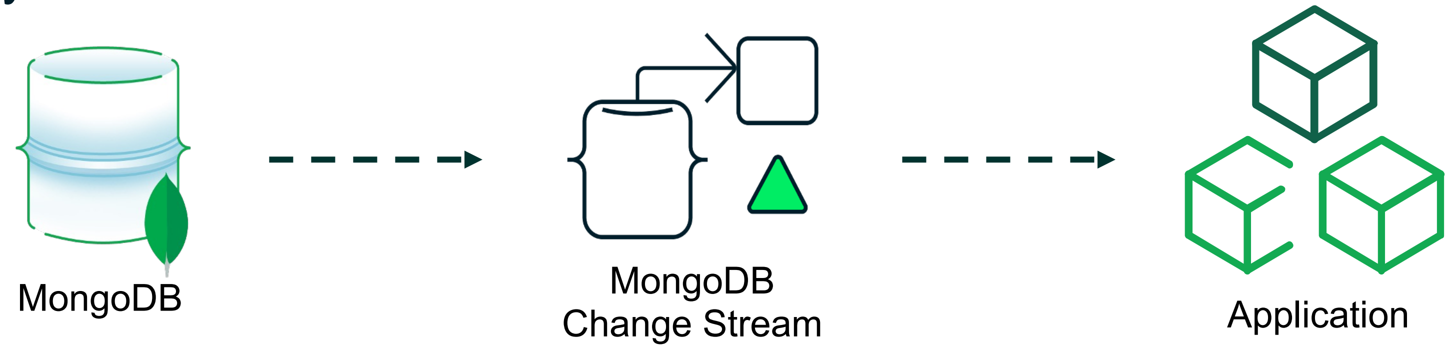
## Introduction to MongoDB Change Stream

## What is a MongoDB Change Stream?

Change Stream is a technology that enables real-time streaming of data changes in a database to applications. Through Change Stream, your application can instantly react to data modifications on a single collection, an entire database, or across the entire deployment. For applications reliant on data change notifications, Change Stream plays a pivotal role.

# Advantages of MongoDB Change Stream

Change Stream enables applications to access real-time data changes without the need to deal with the complexity and risks of tracking oplog. Since Change Stream utilizes the Aggregation Framework, applications can selectively filter specific changes to notify them about data modifications as needed.



## Change Stream Example Code in Go

```go
resumeToken := original.ResumeToken()
pipeline := mongo.Pipeline{bson.D{{“$match”, bson.D{{“$or”,
    bson.A{
      bson.D{{“operationType”, ”update“}}}}},
}}}
cs, err := coll.Watch(ctx, pipeline, options.ChangeStream().SetResumeAfter(resumeToken).
SetFullDocument(options.UpdateLookUp))
if err != nil {
   return err
}
defer cs.Close(ctx)
ok = cs.Next(ctx)

var event changeEvent
decodeErr = cs.Decode(&event)
if decodeErr != nil {
  return decodeErr
}
```

## Change Event Example

```
{
  “_id”: { <Resume Token> },
  “operationType”: “update”,
  “clusterTime”: <Timestamp>,
  “wallTime”: <ISODate>,
  “ns”: {
    “db”: “engineering”,
    “coll”: “users”
  },
  “documentKey”: {
    “_id”: ObjectId(“58a4eb4a30c75625e00d2820”)
  },
  “updateDescription”: {
    “updatedFields”: {
      “name”: “Alice”
    },
  },
  “fullDocument”: {
    “_id”: ObjectId(“58a4eb4a30c75625e00d2820”),
    “name”: “Alice”,
    “email”: “alice@10gen.com”,
  }
}
```

# Features of MongoDB Change Stream

Filterable

Resumable

In order

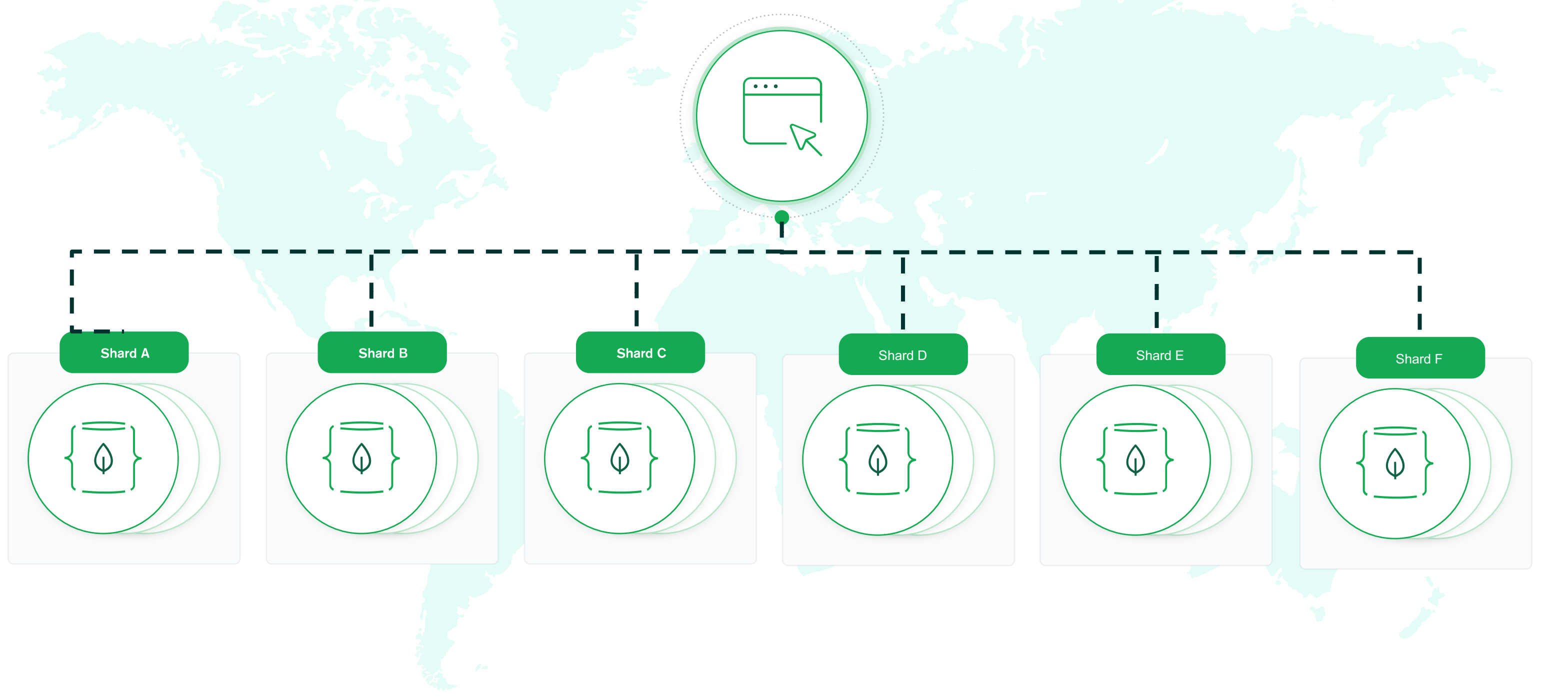
Durable

Secure

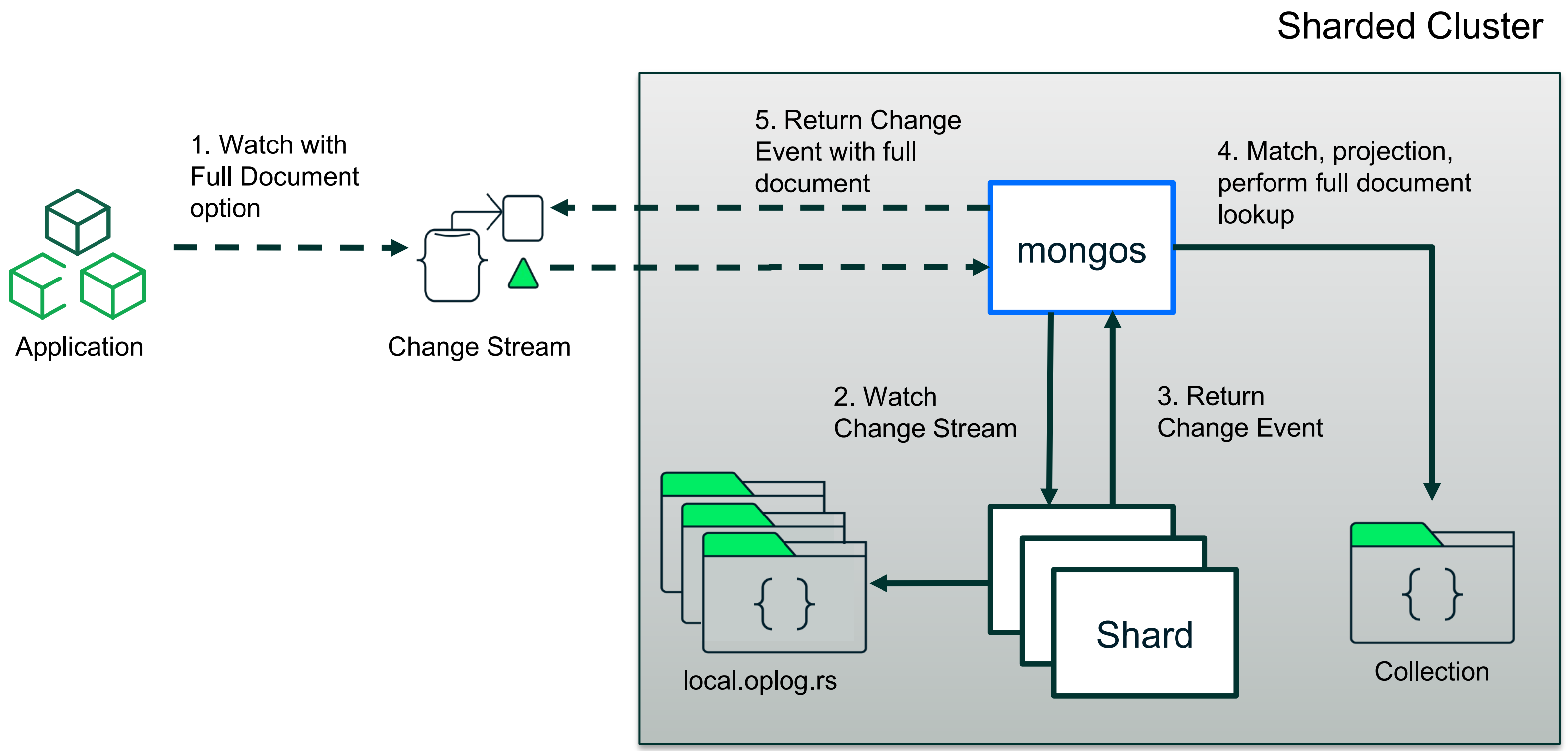
Easy to use

## Performance improvements in Change Stream from MongoDB 4.0 to MongoDB 6.0

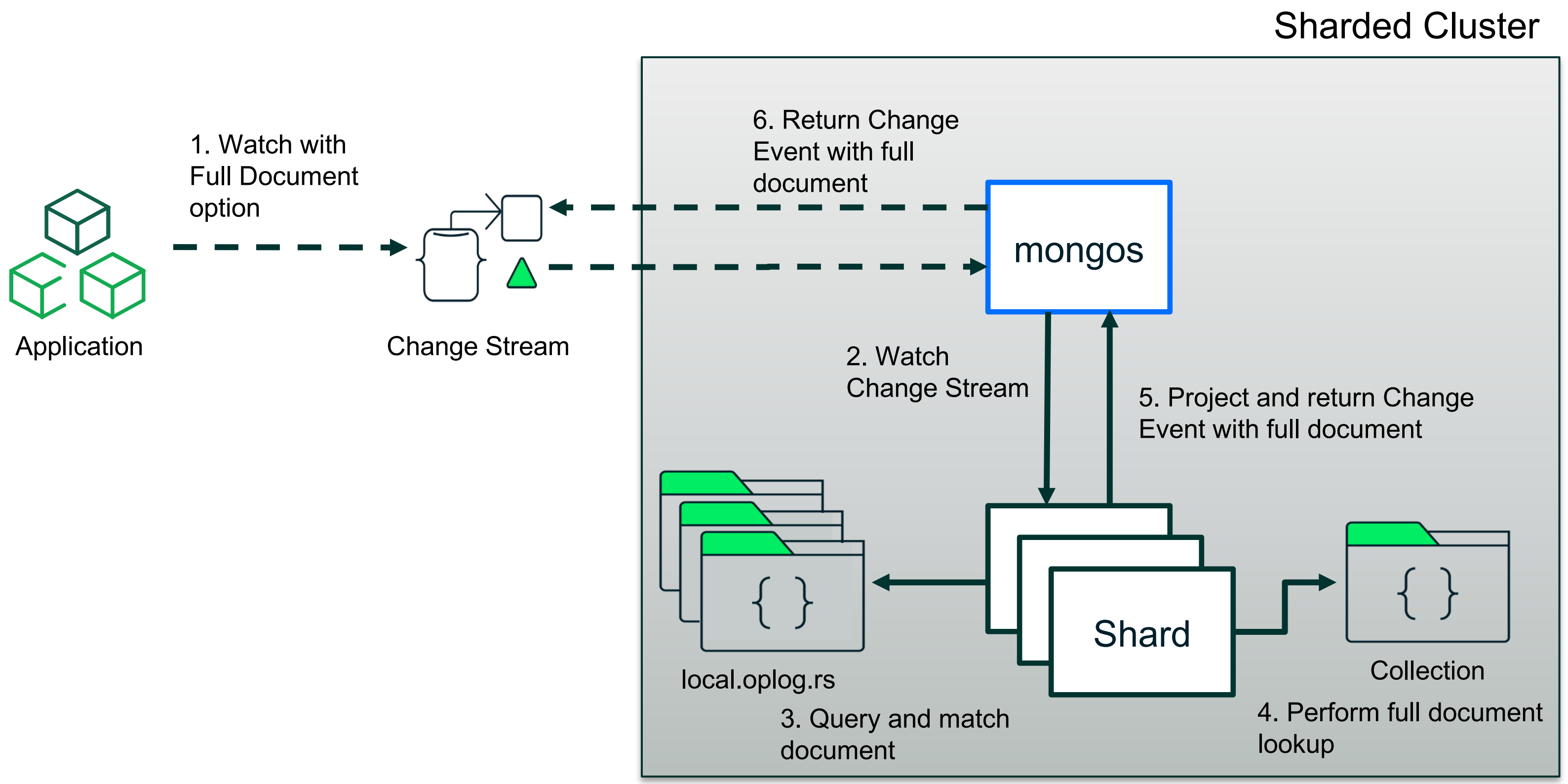
# MongoDB Atlas Global Cluster in Different Regions



# MongoDB 4.x ~ 5.x – Change Stream Flow


Sharded Cluster
Application
1. Watch with Full Document option
Change Stream
5. Return Change Event with full document
mongos
4. Match, projection, perform full document lookup
2. Watch Change Stream
3. Return Change Event
{ }
local.oplog.rs
Shard
{ }
Collection

# MongoDB 6.0 – Change Stream Flow


Sharded Cluster
1. Watch with Full Document option
Application
Change Stream
6. Return Change Event with full document
mongos
2. Watch Change Stream
5. Project and return Change Event with full document
{ }
local.oplog.rs
Shard
{ }
Collection
3. Query and match document
4. Perform full document lookup

## Most Common Scenarios

## 若下游服務因Bug造成event資料處理上出現問題或是掉event時該怎麼辦？

## What is an Event Compensation?

Event compensation refers to a mechanism or process used to correct or reverse the effects of an event when errors, failures, or unexpected outcomes occur during its execution.

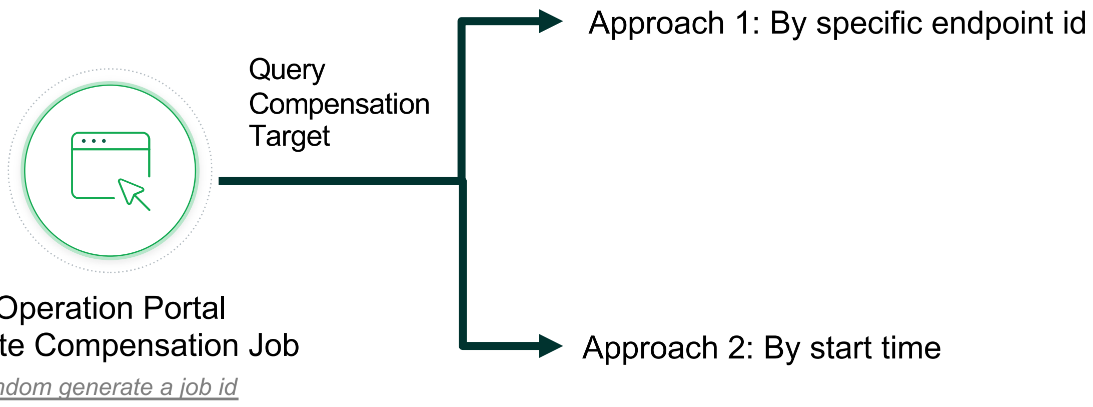
**The goal of a compensation mechanism is to ensure that the system can recover appropriately in the face of errors or failures, thus preserving overall stability and data consistency.**

# Use Change Stream to Achieve Event Compensation

## Background

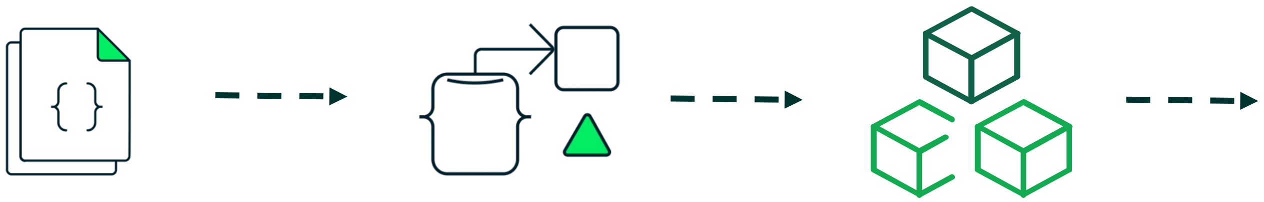
Each document includes an "**update_time**" field, which is used to record the update time of document. Whenever any field within the document is updated, the "update_time" field is also updated to the current time.

## When performing the operation for compensation


Query Compensation Target
Approach 1: By specific endpoint id
Approach 2: By start time
Operation Portal
Create Compensation Job
Random generate a job id

# Use Change Stream to Achieve Event Compensation

## When performing the operation for compensation



Update the job id to compensation_job_id field

Watch compensation_job_id field

When receiving the Change Event containing compensation_job_id, the app publishes the compensation event according to status

*EndpointCompensated*
*EndpointDeletionCompensated*

When services receive a compensation event, they will execute the corresponding actions.

## 如何透過Change Event拿到已刪除資料的Full Document？

## Approach 1：Soft Deletion

When you need to delete a document, you can mark it as deleted by adding a designated field. At the time of updating the document, an additional field is included to record the deletion timestamp. When a Change Event is received containing this newly added field, the event is interpreted as a deletion operation.

Utilize a TTL Index to automatically delete documents based on the "deleted time" field after a period of time.

## Example

### Document

```
{
  _id: ObjectId<ObjectId>,
  name: ‘Bilbo Baggins’,
  status: ‘deleted’,
  deleted_time: ISODate(
"2023-08-08T13:33:46.369+0000")
}
```

### Create TTL index

```
db.runCommand({
  "collMod": <collName>,
  "index": {
    "keyPattern": {"deleted_time": 1},
    "expireAfterSeconds": 86400
  }
})
```

## Approach 2：MongoDB 6.0 Document Pre-Image

**Pre-Image**

The pre-image is the document before it was replaced, updated, or deleted. There is no pre-image for an inserted document.

**Post-Image**

The post-image is the document after it was inserted, replaced, or updated. There is no post-image for a deleted document.

## Example

### Enable Change Stream pre- and post-images

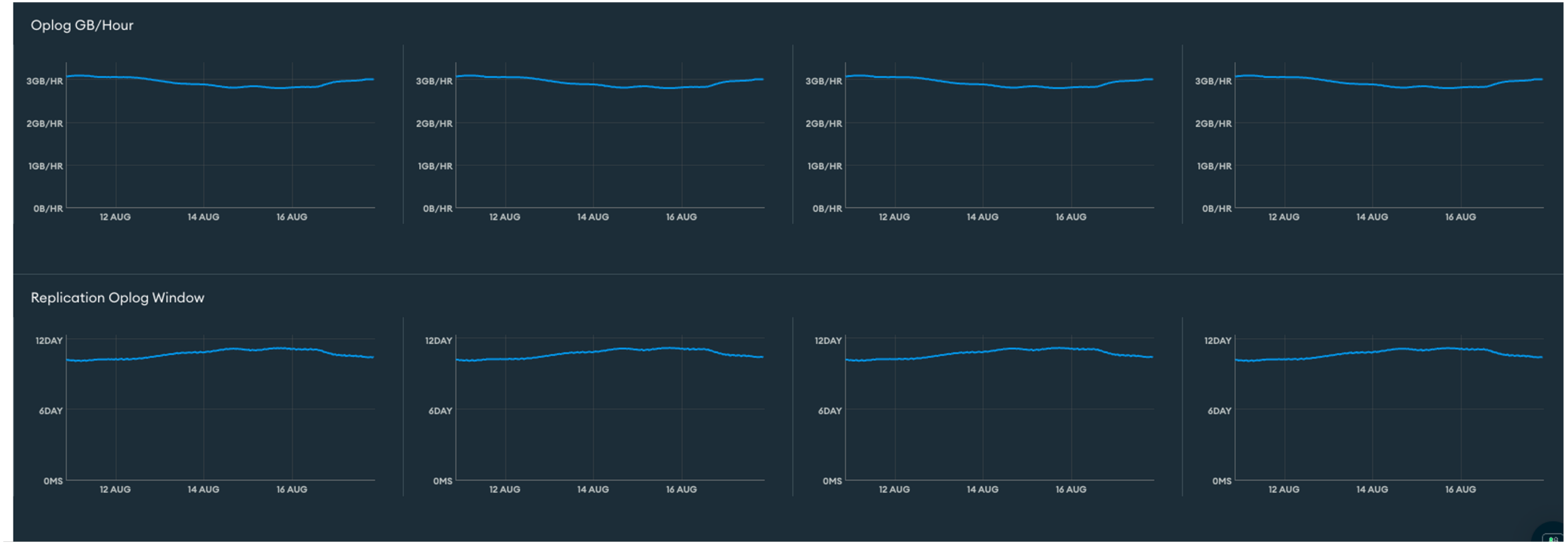
```
db.runCommand({
    collMod: <collName>,
    changeStreamPreAndPostImages: {
      enabled: true
    }
})
```

### Change Event output

```
"fullDocumentBeforeChange":{
  _id: ObjectId<ObjectId>,
  name: ‘wonderful’,
  email: ‘mongodb_taipei@mongodb.com’
}
```

## Best Practices

# Monitor the usage of oplog size



## Change Event 16 MB Size Limitation

**Prior to MongoDB 7.0**

If a change stream has large events that exceed 16 MB, a BSONObjectTooLarge exception is returned. Use a $project stage to include only the fields necessary for your application

**Starting to MongoDB 7.0**

You can use a $changeStreamSplitLargeEvent stage to split the events into smaller fragments.

Thank you for your time. ☺