.NET Conf TAIWAN 2024

初探 Drasi

微軟開源的 Data Change Processing Platform

Alan Tsai

蔡孟玹

- 後端工程師 - .NET 技術為主
- 喜歡學習不同東西
  - Azure
  - Data Science、Chatbot
  - Container、DevOps
  - 加强開發的Tools、架構
- 翻譯文章/軟體
  - 兩本翻譯書
- 三門線上課程
- 看小說、玩手游

教就是最好的學習方式 - 喜歡考證驗證所學

.NET Conf TAIWAN 2024

喜歡技術分享

2024 11 月 GDG 小聚
探索 GitHub Copilot 的新功能讓我們的應用程式開發更快一步

2024 08 月 TSMC IT Community Meetup
聊聊生成式 AI 衝擊下工程師可以做好什麼準備

2024 06 月 Global DevOps Experience Day

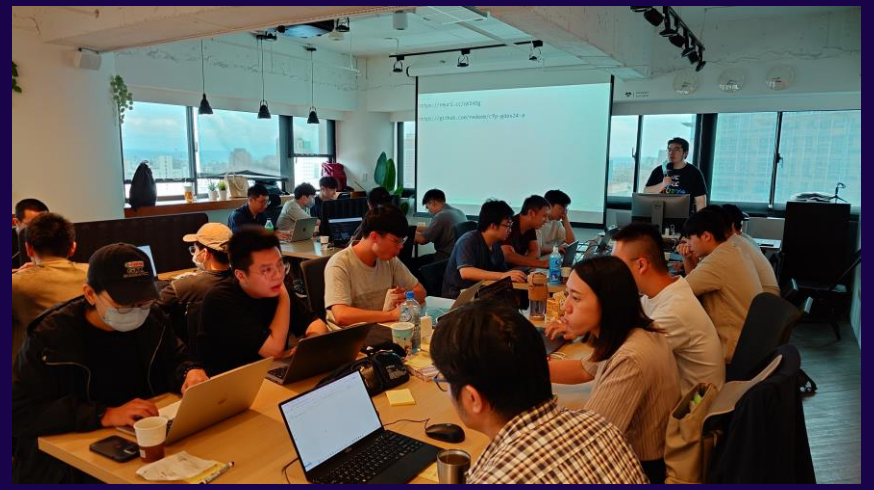
2023 .NET Conf Taiwan
探索 API 開發的挑戰與解決之道
從 GitHub Copilot 到 Enterprise Copilot：打造符合企業需求的智能開發助手之路

2023 10 月 GDG 小聚
開始邁上你的 Azure 成本管控大師之路

2022 .NET Conf Taiwan
談 Event Driven Architecture 之前是不是該把 Event 規格搞定?
CloudEvents 是什麼? ｜邁上 Cloud Native App 之路

.NET Conf TAIWAN 2024

STUDY4.TW 為學習而生



喜歡技術分享

Tibame
AZ-900、AZ-104、AZ-204

GitHub Copilot
企業導入，在製造業、金融業、服務業，已培訓近 2000+ 人次

Trainocate
AZ-104、AZ-204、AZ-400、DP-200、DP-201、PL-900、DA-100

台灣智慧自動化與機器人協會
運用Python進行大數據分析、機器學習基礎理論課程及人工智慧 – ML.NET

中華電信學院
DevOps與CI/CD實務研習班
Windows Container 技術實務班
使用Azure AI打造有人工智能的Line聊天機器人

教就是最好的學習方式 - 喜歡考證驗證所學

.NET Conf TAIWAN 2024

Alan Tsai 的學習筆記

https://blog.alantsai.net

contact@alantsai.net

FB 粉絲頁

http://fb.alantsai.net

contact@alantsai.net

影片

http://yt.alantsai.ne

http://bili.alantsai.net

.NET Conf TAIWAN 2024

Agenda

01

What is the problem

02

What is Drasi

03

How to use Drasi

What is the problem

.NET Conf TAIWAN 2024

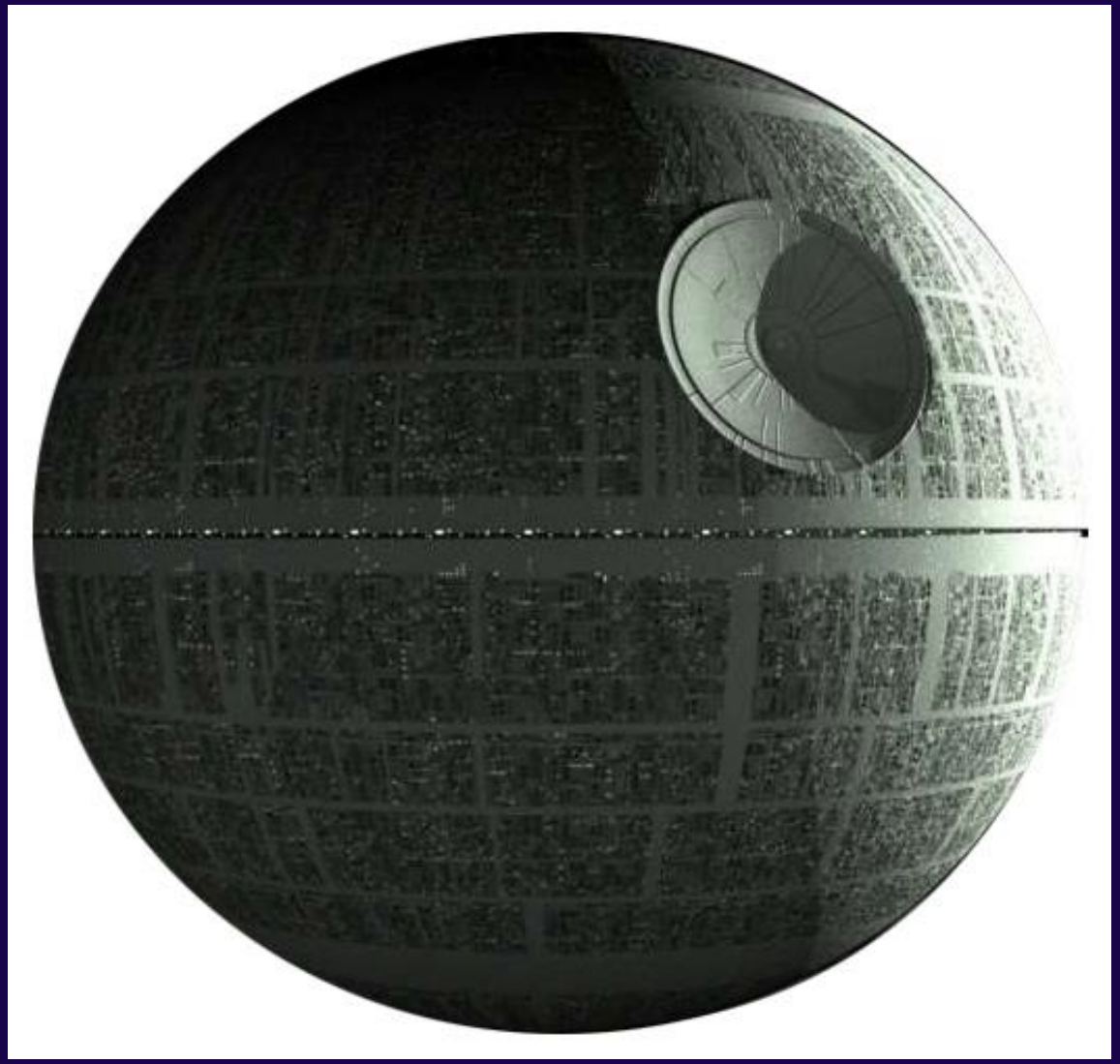
What is a Program

.NET Conf TAIWAN 2024

Data + Logic

.NET Conf TAIWAN 2024

Death Star Architecture

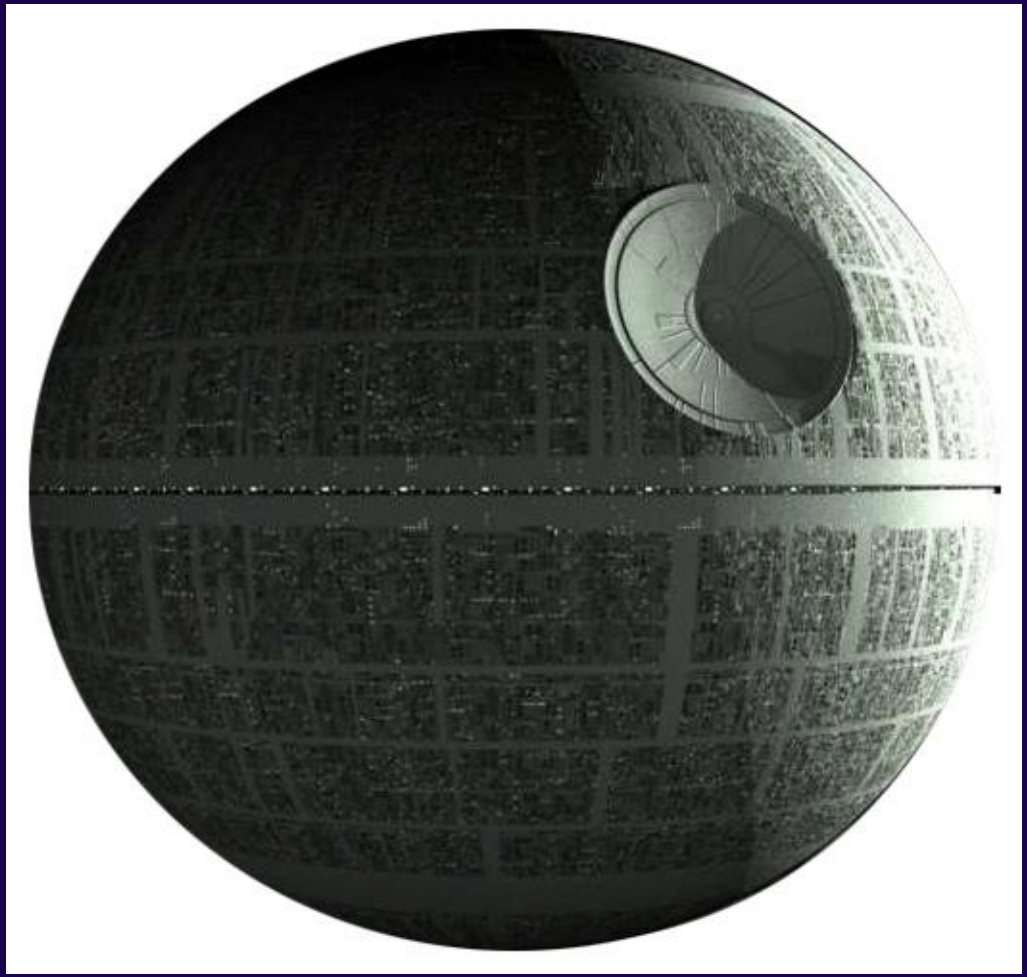


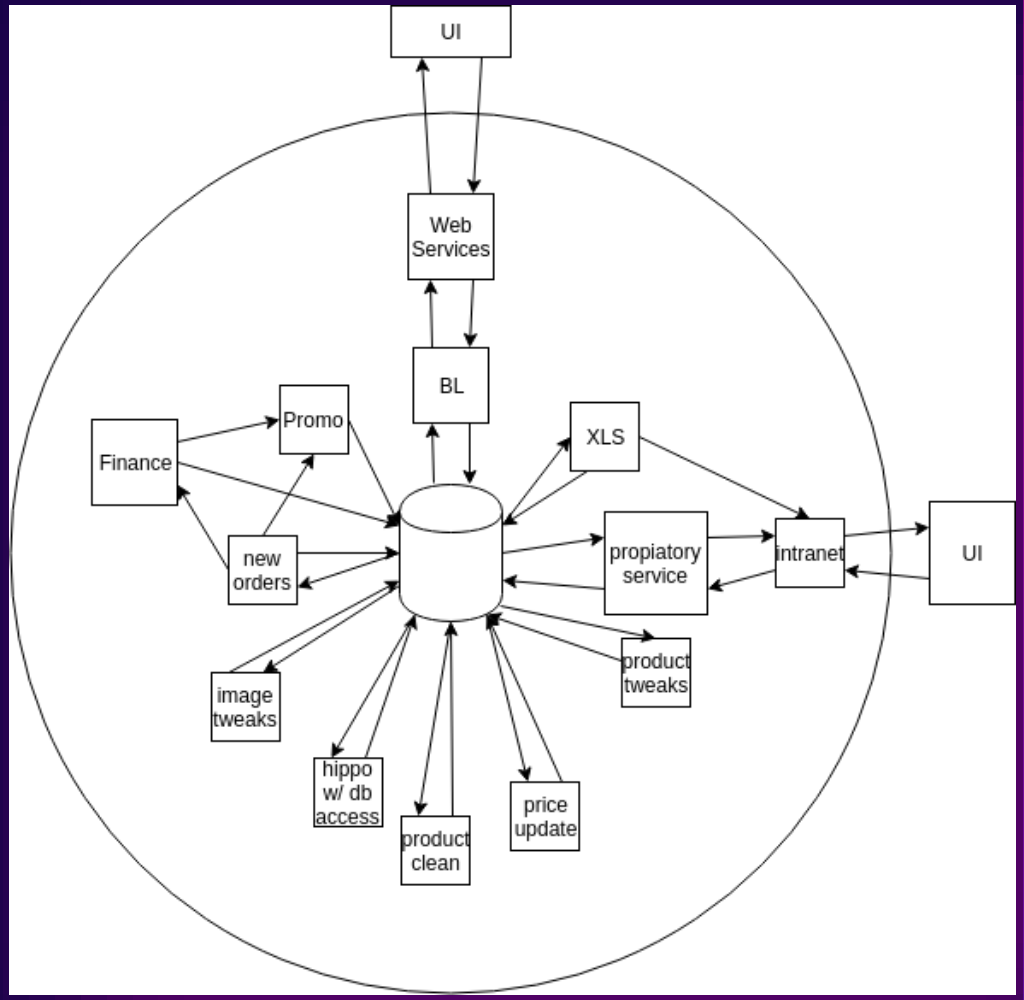
It appears powerful but is fundamentally vulnerable.

1.Take the cheaper faster option.

2.Take the cheaper faster option.

3.Goto 1.





[What is Death Star Architecture and why you should care]

改 A 壞 B => 有程式碼品質問題

Gate Keepers 變成英雄

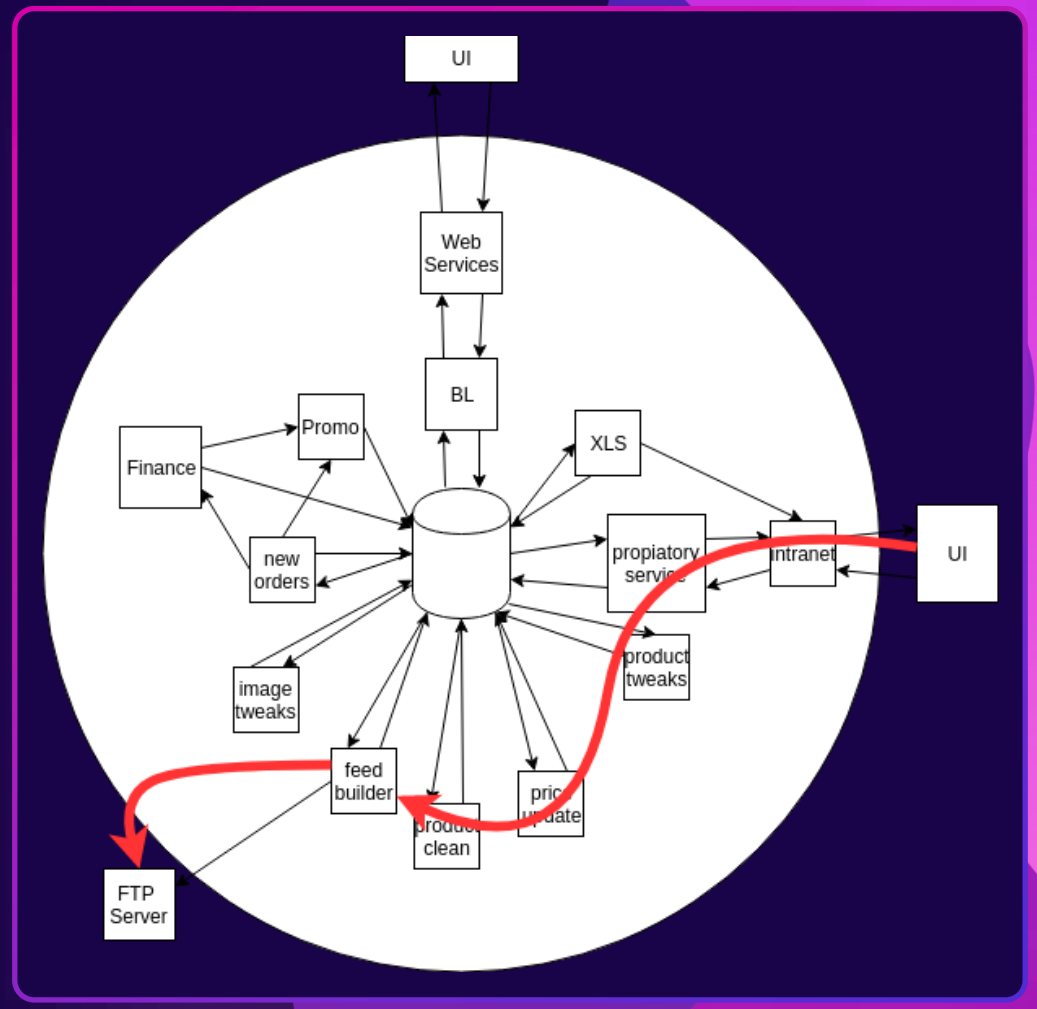
為了加速開發，我們導入 “敏捷” 專注在 Process, Documentation, Planning

整個糾結在一起，value stream 散落在各個解決方案

開始有很多分支 (但是 DB 是同一座...)

商業需求不會因為開發變慢而停下

各個部門解決自己的問題，而沒有思考整體公司。沒有一致步調前進



.NET Conf TAIWAN 2024

We need to share Data not Data Source

Three way of communication Logic and Data

01

Synchronous
API

02

Asynchronous
Event

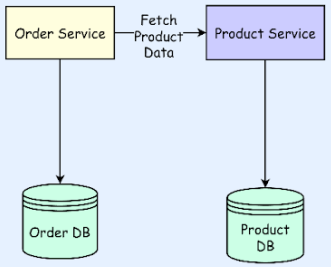
03

Data
CDC

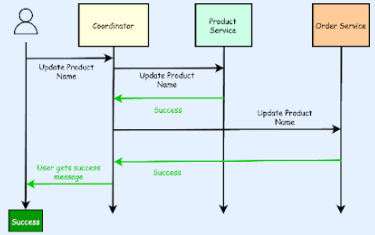
Organizations often need to take some action when data stored in a database changes, or, conversely, when it doesn't change as expected

A Cheatsheet On Data Sharing Between Microservices

Synchronous Data Sharing

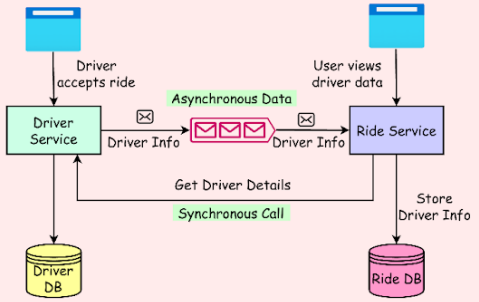


Synchronous approach involves real-time communication between services

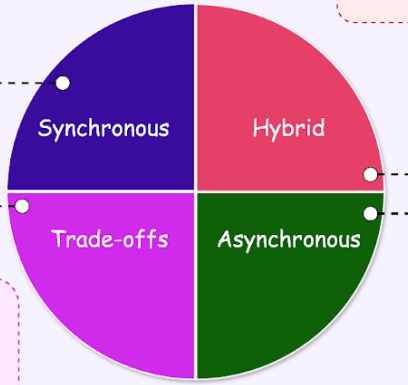


Coordinator that acts like a gateway initiates the update process across multiple services

Hybrid Approach



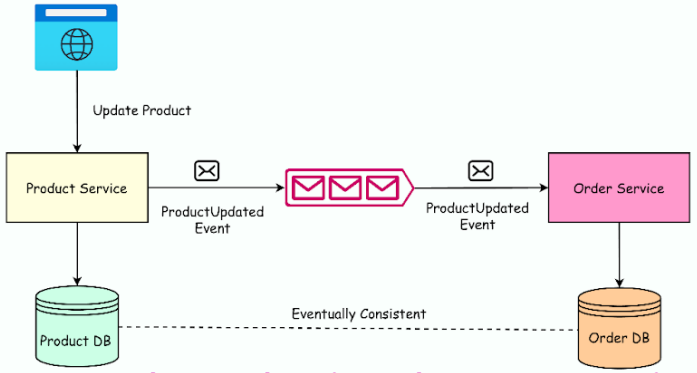
Hybrid approach contains elements of both synchronous and asynchronous methods



Trade-offs and Decisions

- Evaluate the consistency requirements
- Strong vs Eventual Consistency
- Performance considerations such as latency, throughput, and resource utilization
- Application scalability considerations such as horizontal scaling, data volume, and service evolution

Asynchronous Data Sharing



Asynchronous Data Sharing leads to eventual consistency

https://blog.bytebytego.com/p/data-sharing-between-microservices

What is the challenge?

Change Detection Challenges

Real-Time Response Issues

Developer and Business Impact

instantaneous query

What is the current solution

.NET Conf TAIWAN 2024

What is drasi

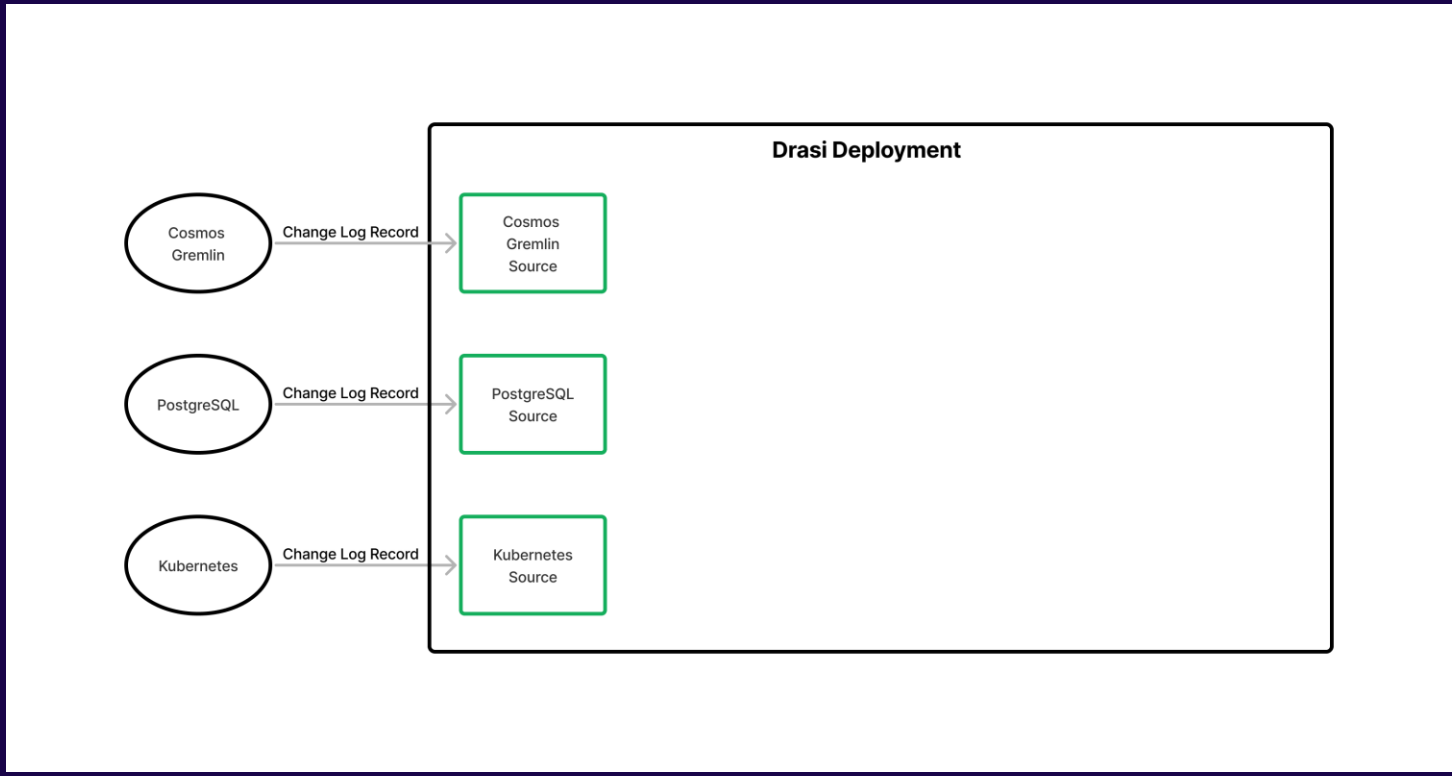
Propose to CNCF Sandbox Open Source Project


drasi
Change logs
Detect
Sources
Continuous Queries
Reactions
React
Action

Source

Connection of data

- Azure Cosmos DB Gremlin API
- PostgreSQL
- Event Hubs
- Microsoft SQL Server
- Microsoft Dataverse
- Kubernetes



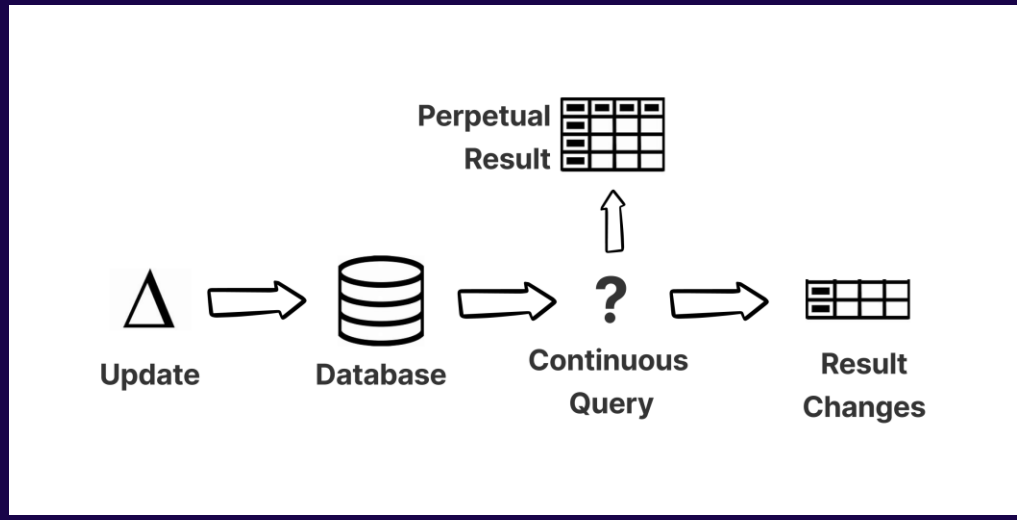
Continuous Queries

Cypher query

- **Real-Time Data Updates:**
- Continuously incorporate changes from the source database.
- Maintain accurate, up-to-date query results.

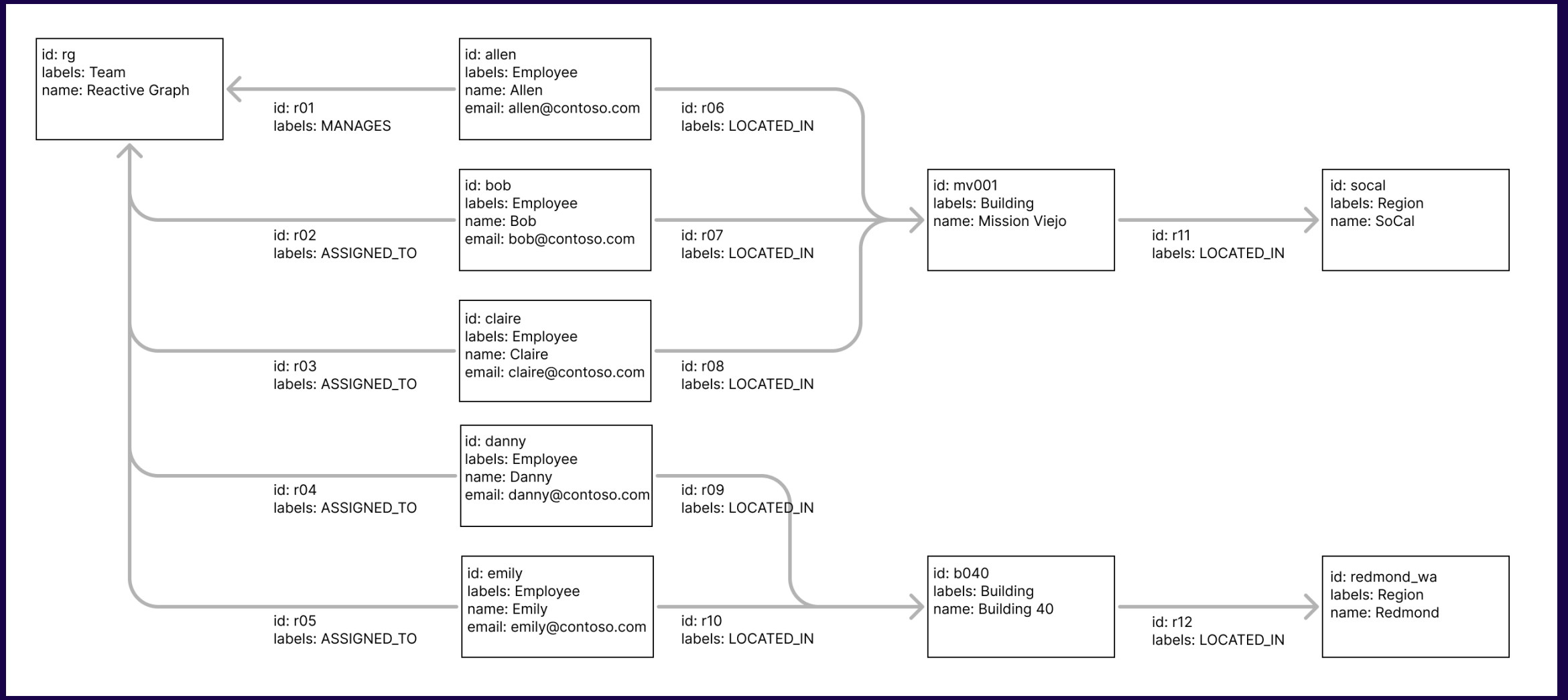
- **Flexible Query Capabilities:**
- Use Cypher Query Language for detailed change detection.
- Rich query logic for properties and relationships.

- **Multi-Source Integration:**
- Queries span multiple data sources without complex joins.
- Integrates relational and graph data seamlessly.

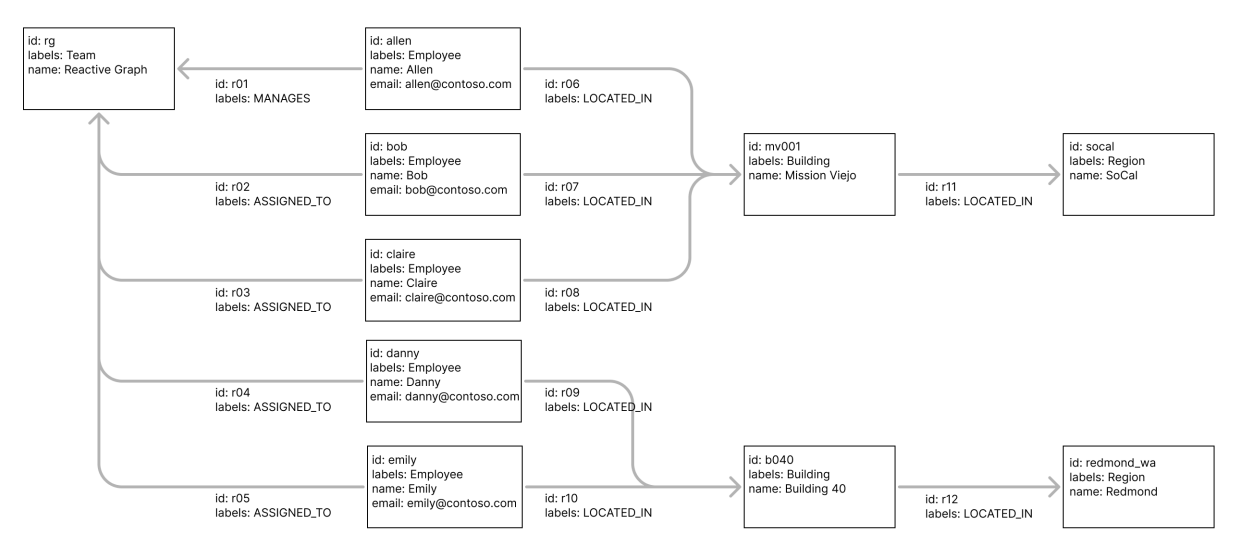


Incident Alert Detection

Cypher query: Node and Relation



Incident Alert Detection

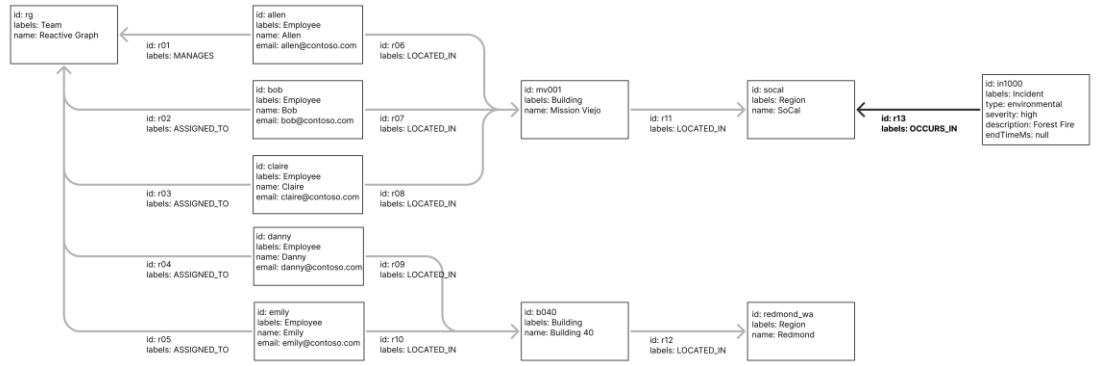


Employees located in **Buildings** within **Regions** where there are active **Incidents** of **type** 'environmental' that have a **severity** level of 'critical' or 'extreme'

The name and email address of the at risk employee and their manage

```
MATCH
  (e:Employee)-[:ASSIGNED_TO]->(t:Team),
  (m:Employee)-[:MANAGES]->(t:Team),
  (e:Employee)-[:LOCATED_IN]->(:Building)-[:LOCATED_IN]->(r:Region),
  (i:Incident {type:'environmental'})-[:OCCURS_IN]->(r:Region)
WHERE
  elementId(e) <> elementId(m) AND i.severity IN [‘critical’, ‘extreme’] AND i.endTimeMs IS NULL
RETURN
  m.name AS ManagerName, m.email AS ManagerEmail,
  e.name AS EmployeeName, e.email AS EmployeeEmail,
  r.name AS RegionName,
  elementId(i) AS IncidentId, i.severity AS IncavidentSeverity, i.description AS IncidentDescription
```

Incident Alert Detection




```
{
 “added”: [
  { “ManagerName”: “Allen”, “ManagerEmail”: “allen@contos
“Southern California”, “IncidentId”: “in1000”, “IncidentSeveri
  { “ManagerName”: “Allen”, “ManagerEmail”: “allen@contos
“Southern California”, “IncidentId”: “in1000”, “IncidentSeveri
 ],
 “updated”: [],
 “deleted”: []
}
```


```
MATCH
  (e:Employee)-[:ASSIGNED_TO]->(t:Team),
  (m:Employee)-[:MANAGES]->(t:Team),
  (e:Employee)-[:LOCATED_IN]->(:Building)-[:LOCATED_IN]->(r:Region),
  (i:Incident {type:'environmental'})-[:OCCURS_IN]->(r:Region)
WHERE
  elementId(e) <> elementId(m) AND i.severity IN [‘critical’, ‘extreme’] AND i.endTimeMs IS NULL
RETURN
  m.name AS ManagerName, m.email AS ManagerEmail,
  e.name AS EmployeeName, e.email AS EmployeeEmail,
  r.name AS RegionName,
  elementId(i) AS IncidentId, i.severity AS IncavidentSeverity, i.description AS IncidentDescription
```

Incident Alert Detection

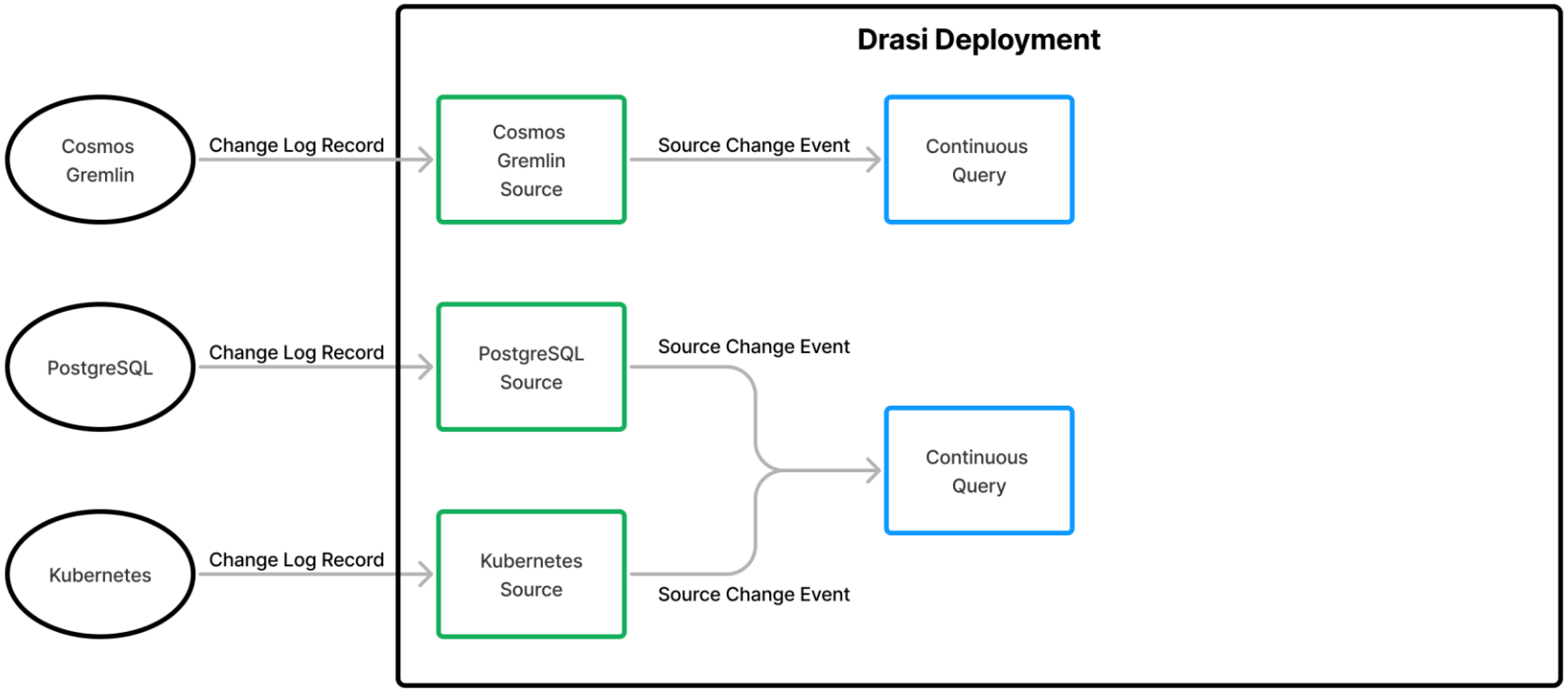
If Bob Change Location

```
{
“added”: [],
“updated”: [],
“deleted”: [ {
“ManagerName”: “Allen”,
“ManagerEmail”:
“allen@contoso.com”,
“EmployeeName”: “Bob…” }
]
}
```

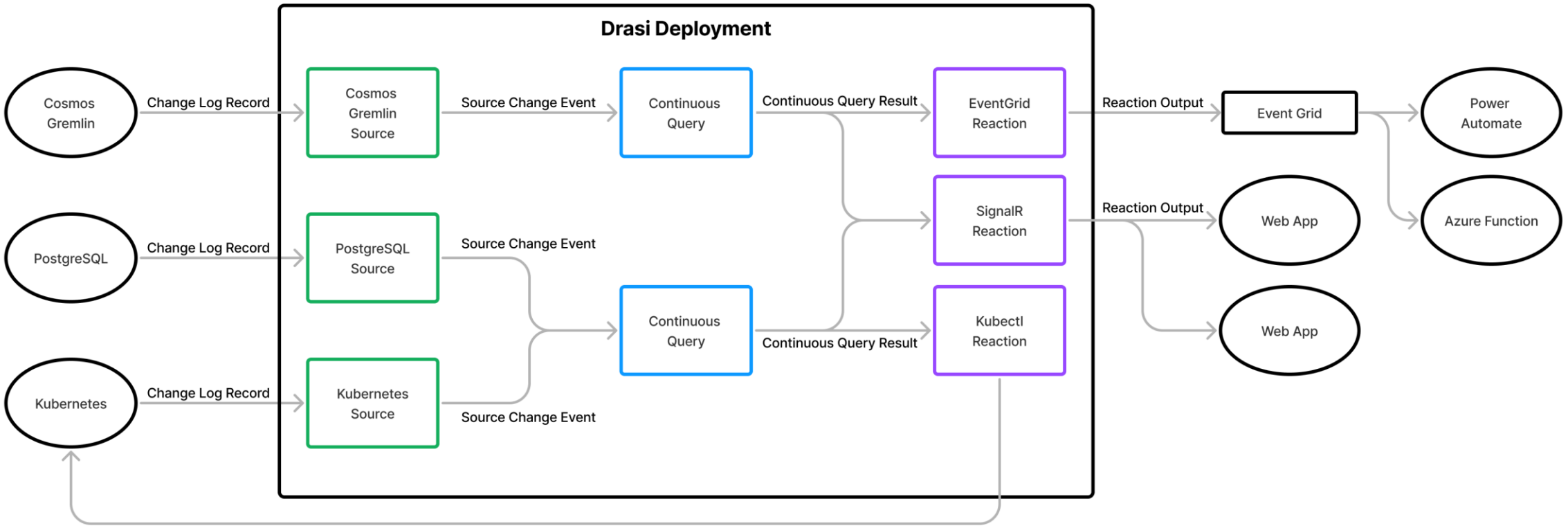
If severity change

```
{
“added”: [],
“updated”:
[ { “before”: { “Manager
“after”: { “ManagerName”
“deleted”: []
}
```

Continuous Query Abstract



Reaction


Drasi Deployment
Cosmos Gremlin
Change Log Record
Cosmos Gremlin Source
Source Change Event
Continuous Query
Continuous Query Result
EventGrid Reaction
Reaction Output
Event Grid
Power Automate
Azure Function
PostgreSQL
Change Log Record
PostgreSQL Source
Source Change Event
SignalR Reaction
Reaction Output
Web App
Kubernetes
Change Log Record
Kubernetes Source
Source Change Event
Continuous Query
Continuous Query Result
Kubectl Reaction
Web App

.NET Conf TAIWAN 2024

How to use 

Propose to CNCF Sandbox Open Source Project

drasi cli

Dependency

1. Kubernetes
2. Dapr

Monitor

1. OpenTelemetry
2. Prometheus

Install on AKS

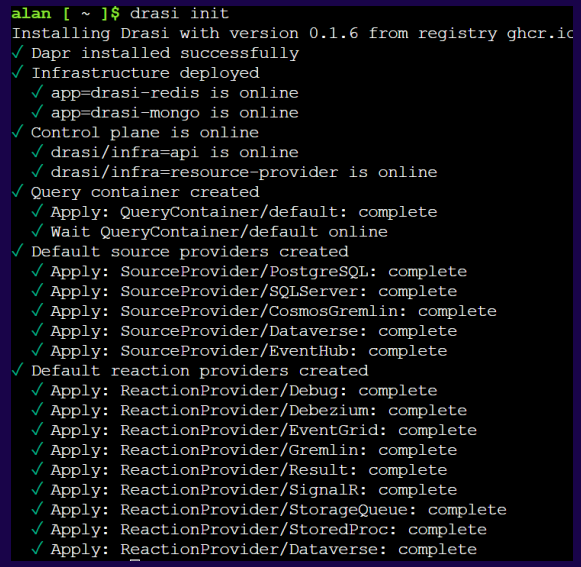
1. Connect to AKS

```
az aks get-credentials
  --resource-group <group>
  --name <name>
```

2. Download drasi cli

```
curl -fsSL
https://raw.githubusercontent.com/drasi-
project/drasi-
platform/main/cli/installers/install-drasi-
cli.sh | /bin/bash
```

3. Call drasi init

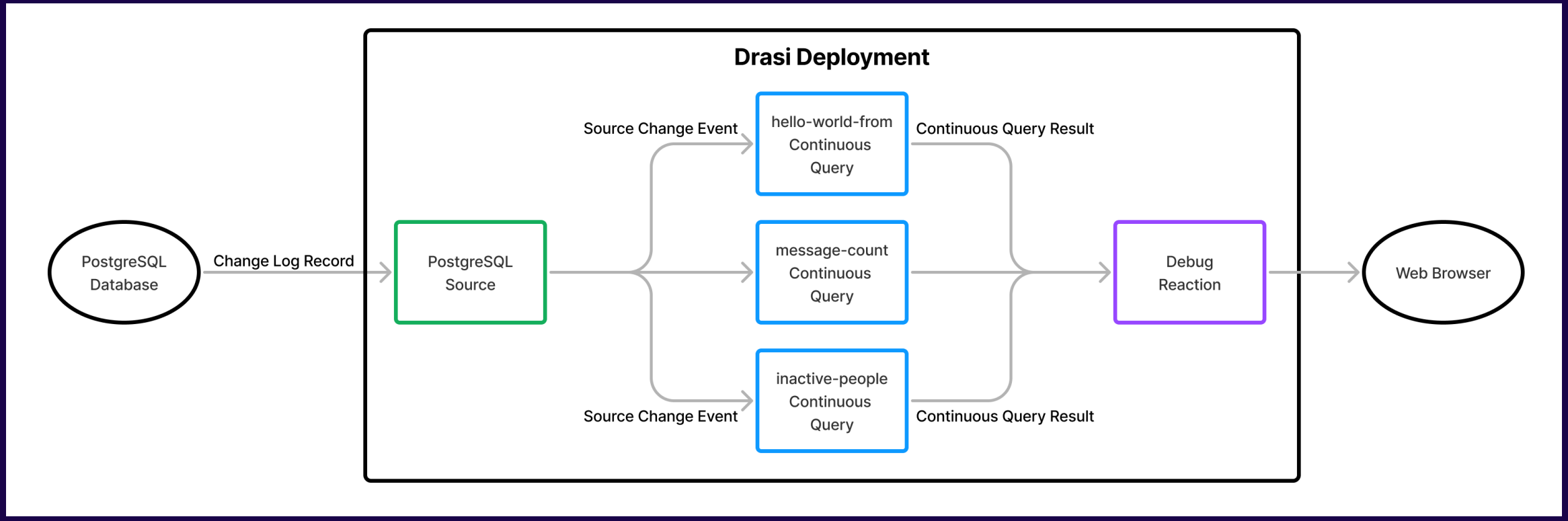



```
alan [ ~ ]$ kubectl get pods -n dapr-system
NAME                                    READY   STATUS    RESTARTS        AGE
dapr-dashboard-5cc65d985f-zkbsc         1/1     Running   0               6m52s
dapr-operator-5d98f57c86-pcjft          1/1     Running   1 (6m28s ago)   6m52s
dapr-placement-server-0                 1/1     Running   0               6m52s
dapr-sentry-697bdc6cc4-g2gdj            1/1     Running   0               6m52s
dapr-sidecar-injector-56c4c4b485-25gvp  1/1     Running   0               6m52s
```

https://drasi.io/how-to-guides/installation/install-on-

Solution Architecture

1. Which people have sent the message “Hello World”?
2. How many times has the same message been sent?
3. Which people haven’t sent a message in the last 20 seconds?



Prepare the data - PostgreSQL

Field Name	Type	Description
MessageId	integer	A unique id for each message.
From	character varying(50)	The name of who the message is from.
Message	character varying(200)	The text of the message.

MessageId	From	Message
1	Buzz Lightyear	To infinity and beyond!
2	Brian Kernighan	Hello World
3	Antoninus	I am Spartacus
4	David	I am Spartacus

Deploy PostrgreSQL

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: test-data-init
data:
 init.sql: >
  CREATE TABLE "Message" (
    "MessageId" integer NOT NULL,
    "From" character varying(50) NOT NULL,
    "Message" character varying(200) NOT NULL
  );

  ALTER TABLE "Message" ADD CONSTRAINT pk_message
   PRIMARY KEY ("MessageId");

  INSERT INTO public."Message" VALUES (1, 'Buzz Lightyear', 'To infinity and beyond!');
  INSERT INTO public."Message" VALUES (2, 'Brian Kernighan', 'Hello World');
  INSERT INTO public."Message" VALUES (3, 'Antoninus', 'I am Spartacus');
  INSERT INTO public."Message" VALUES (4, 'David', 'I am Spartacus');
```

```
kubectl apply
-f drasi-postgres.yaml

kubectl port-forward
svc/postgres
5432:5432
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: postgres
spec:
 replicas: 1
 selector:
  matchLabels:
   app: postgres
 template:
  metadata:
   labels:
    app: postgres
  spec:
   containers:
    - name: postgres
      image: postgres:15-alpine
      args: ["-c", "wal_level=logical"]
      volumeMounts:
      - name: init
       mountPath: "/docker-entrypoint-initdb.d"
      ports:
       - containerPort: 5432
      envFrom:
       - configMapRef:
          name: test-pg-config
   volumes:
    - name: init
      configMap:
       name: test-data-init
```

```
apiVersion: v1
kind: Service
metadata:
 name: postgres
 labels:
  app: postgres
spec:
 type: ClusterIP
 ports:
  - port: 5432
 selector:
  app: postgres
```

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: test-pg-config
 labels:
  app: postgres
data:
 POSTGRES_DB: hello-world
 POSTGRES_USER: test
 POSTGRES_PASSWORD: test
```

Test PostgreSQL

psql -h localhost -U test -d hello-world

```
alan [ ~ ]$ psql -h localhost -U test -d hello-world
psql (14.13, server 15.10)
WARNING: psql major version 14, server major version 15.
         Some psql features might not work.
Type "help" for help.

hello-world=# select * from public.message
hello-world-# SELECT * FROM "Message";
ERROR:  syntax error at or near "SELECT"
LINE 2: SELECT * FROM "Message";
        ^
hello-world=# ^Clect * from public.message
SELECT * FROM "Message";
hello-world=# ^C
hello-world=# ^C
hello-world=# ^C
hello-world=# SELECT * FROM "Message";
 MessageId |      From       |         Message
-----------+-----------------+-------------------------
         1 | Buzz Lightyear  | To infinity and beyond!
         2 | Brian Kernighan | Hello World
         3 | Antoninus       | I am Spartacus
         4 | David           | I am Spartacus
(4 rows)

hello-world=#
```

Source

```yaml
apiVersion: v1
kind: Source
name: hello-world
spec:
  kind: PostgreSQL
  properties:
    host: postgres.default.svc.cluster.local
    port: 5432
    user: test
    password: test
    database: hello-world
    ssl: false
    tables:
      - public.Message
```

```
alan [ ~ ]$ ./drasi apply -f ./hello-world-source.yaml
✓ Apply: Source/hello-world: complete
alan [ ~ ]$ drasi list source
bash: drasi: command not found
alan [ ~ ]$ ./drasi list source
      ID      | AVAILABLE | MESSAGES
--------------+-----------+----------
  hello-world | true      |
alan [ ~ ]$
```

Query


```yaml
apiVersion: v1
kind: ContinuousQuery
name: hello-world-from
spec:
  mode: query
  sources:
    subscriptions:
      - id: hello-world
  query: >
    MATCH
      (m:Message {Message: 'Hello World'})
    RETURN
      m.MessageId AS MessageId,
      m.From AS MessageFrom
```

```yaml
apiVersion: v1
kind: ContinuousQuery
name: message-count
spec:
  mode: query
  sources:
    subscriptions:
      - id: hello-world
  query: >
    MATCH
      (m:Message)
    RETURN
      m.Message AS Message,
      count(m.Message) AS Frequency
```

```yaml
apiVersion: v1
kind: ContinuousQuery
name: inactive-people
spec:
  mode: query
  sources:
    subscriptions:
      - id: hello-world
  query: >
      MATCH
        (m:Message)
      WITH
        m.From AS MessageFrom,
        max(drasi.changeDateTime(m)) AS LastMessageTimestamp
      WHERE
        LastMessageTimestamp <= datetime.realtime() - duration({ seconds: 20 })
      OR
        drasi.trueLater(LastMessageTimestamp <= datetime.realtime() - duration({ seconds:
20 }), LastMessageTimestamp + duration({ seconds: 20 }))
      RETURN
        MessageFrom,
        LastMessageTimestamp
```

```
alan [ ~ ]$ vi hello-world-queries.yaml
alan [ ~ ]$ ./drasi apply -f ./hello-world-queries.yaml
✓ Apply: ContinuousQuery/hello-world-from: complete
✓ Apply: ContinuousQuery/message-count: complete
✓ Apply: ContinuousQuery/inactive-people: complete
alan [ ~ ]$ ./drasi list query
         ID         | CONTAINER | ERRORMESSAGE |              HOSTNAME                | STATUS
--------------------+-----------+--------------+--------------------------------------+---------
  hello-world-from  | default   |              | default-query-host-6959d9446-nnhwf   | Running
  message-count     | default   |              | default-query-host-6959d9446-nnhwf   | Running
  inactive-people   | default   |              | default-query-host-6959d9446-nnhwf   | Running
alan [ ~ ]$
```

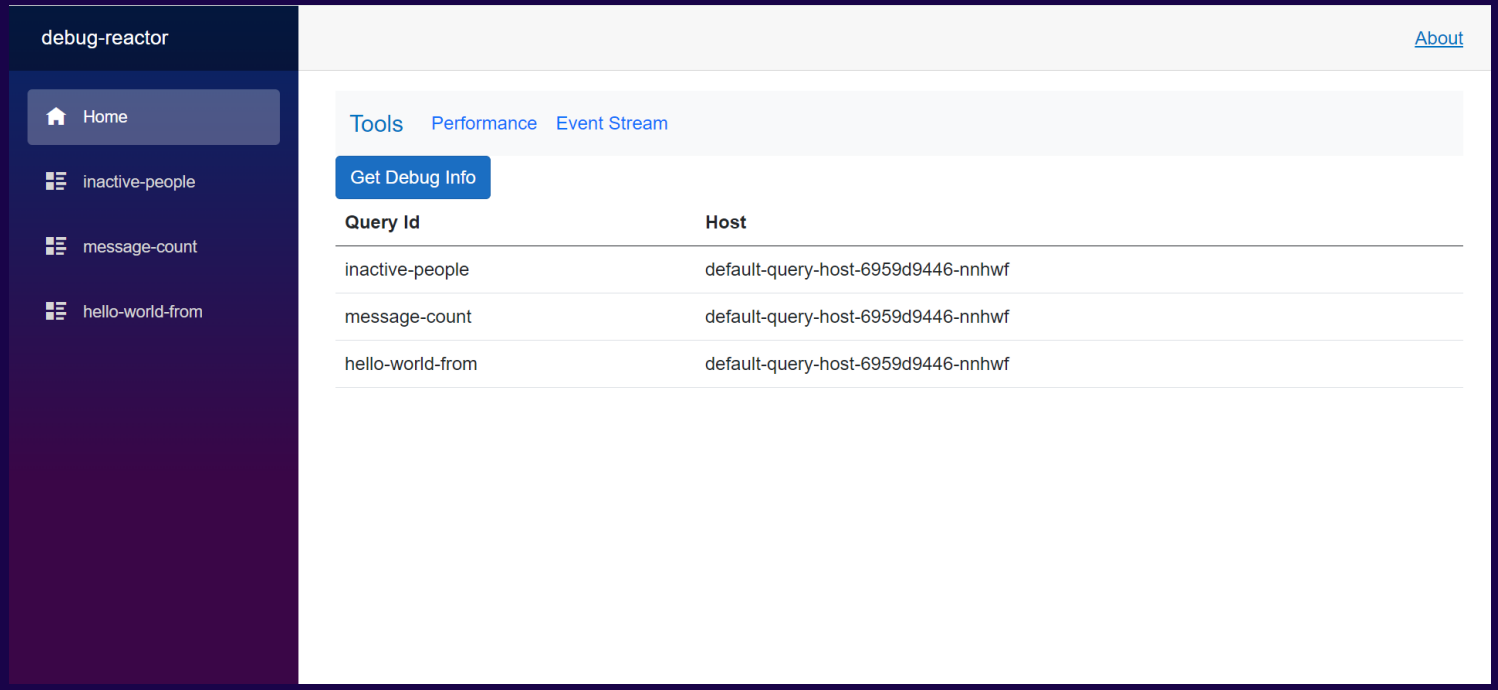
Reaction


```yaml
apiVersion: v1
kind: Reaction
name: hello-world-debug
spec:
  kind: Debug
  queries:
    hello-world-from:
    message-count:
    inactive-people:
```

```
alan [ ~ ]$ ./drasi apply -f ./hello-world-reaction.yaml
✓ Apply: Reaction/hello-world-debug: complete
alan [ ~ ]$ ./drasi list reaction
          ID          | AVAILABLE | MESSAGES
----------------------+-----------+----------
  hello-world-debug | true      |
alan [ ~ ]$
```

Debug

```
kubectl port-forward services/hello-world-debug-gateway 8080:8080 -n drasi-system
```

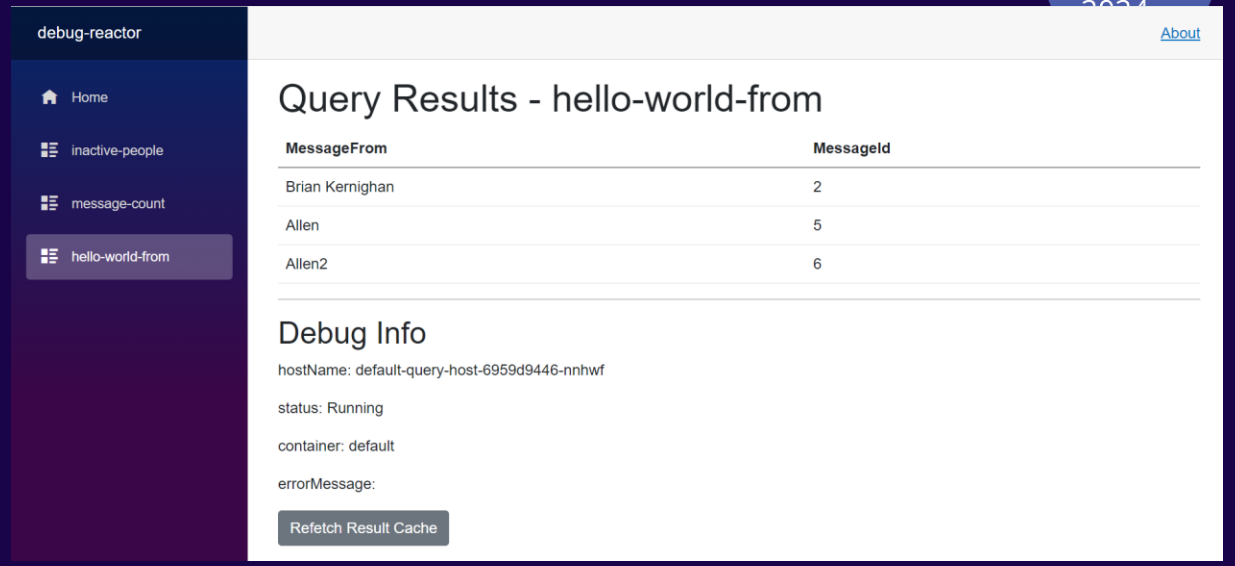


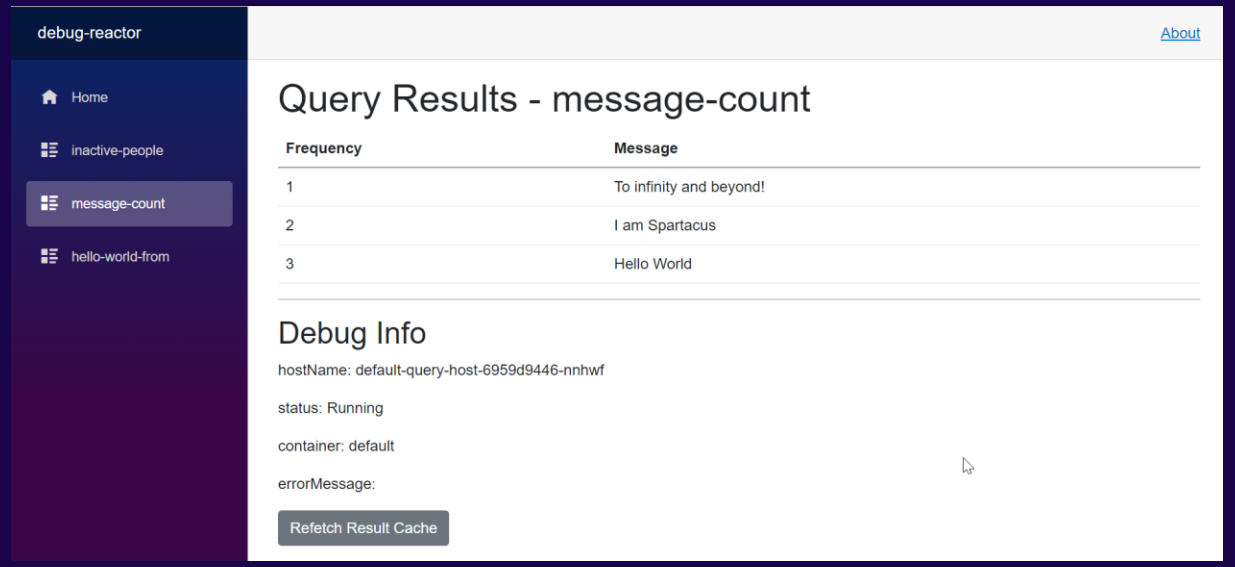
Insert Hello World

Which people have sent the message “Hello World”?
How many times has the same message been sent?

```
INSERT INTO public."Message"
VALUES (5, 'Allen', 'Hello World’);

INSERT INTO public."Message"
VALUES (5, 'Allen', 'Hello World');
```





20 seconds inactive

.NET Conf TAIWAN 2024

debug-reactor

About

- Home
- inactive-people
- message-count
- hello-world-from

Query Results - inactive-people

LastMessageTimestamp	MessageFrom
1970-01-01 00:00:00 +00:00	Buzz Lightyear
1970-01-01 00:00:00 +00:00	Brian Kernighan
1970-01-01 00:00:00 +00:00	Antoninus
1970-01-01 00:00:00 +00:00	David
2024-12-15 03:40:10.867 +00:00	Allen
2024-12-15 03:40:30.672 +00:00	Allen2

Debug Info

hostName: default-query-host-6959d9446-nnhwf

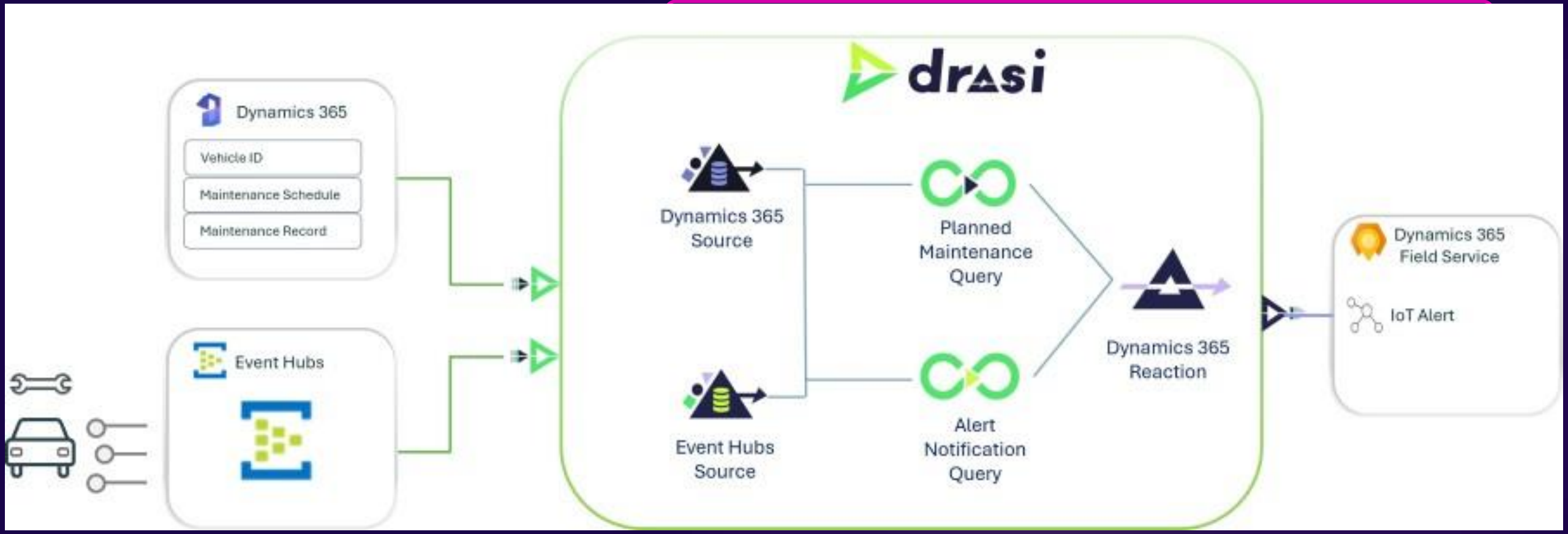
status: Running

container: default

.NET Conf TAIWAN 2024

Conclustion

connected fleet



Solution Design

How to Design Solutions with Drasi

Observing Changes,

Entity CRUD

Observing Conditions,

an Incident becomes critical.

an Order becomes ready for pickup.

a Room’s temperature exceeds 80 degrees.

the occupancy of a Store exceeds 100 people.

Observing Collections

There are new customer orders. These need to be picked from stock.

Once picked, orders need to be packed and prepared for dispatch.

Once prepared, the orders need to be dispatched through various delivery options.

Once dispatched, the orders need to be tracked until delivery.

Once the Order is delivered, it is complete, unless the customer has an issue, in which case a customer support process is initiated...

Solution for Down Stream Application

What is it not good for

01

has a mature change notification capability

02

Continuous Query includes data types for which there is a lot of data

03

stream analytics or streaming data transformation over high volume data streams

Reference

1. Drasi: Microsoft’s newest open-source project simplifies change detection and reaction in complex systems
2. Detect and react intelligently to changes in data with Drasi

3. Drasi.io
4. Learning

.NET Conf
TAIWAN
2024

STUDY4

為 學 習 而 生

.NET Conf TAIWAN 2024

特別感謝

Microsoft® MVP Most Valuable Professional

tsmc

好房網 HouseFun

街口支付 JKOPAY

新加坡商 鈦坦科技 TITANSOFT

STUDY4 為學習而生

以及各位參與活動的各位