



# Building Resilient Data Pipelines

An introduction to Building Data Pipelines in Go on Kubernetes

**Grant Griffiths**  
**Software Engineer**  
**Platform Cloud Engineering**  
**GE Digital**

**PREDIX**

# Who am I?

---



GE Digital - Platform Cloud Engineering  
Sr. Software Engineer



San Francisco, CA



GE Go User Group  
Founder & Organizer



@ggriffiths



@griffithsgrant

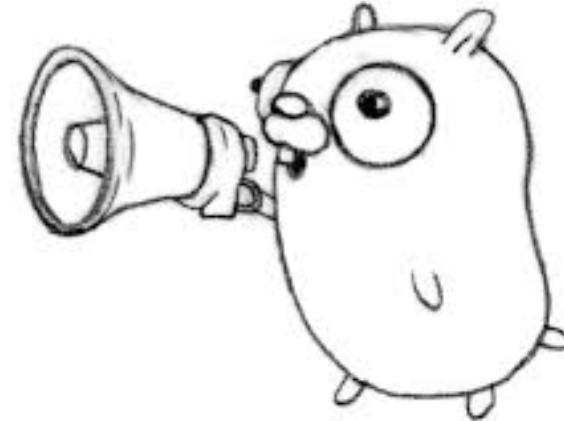
Climbing + Mountaineering



# What we'll cover

---

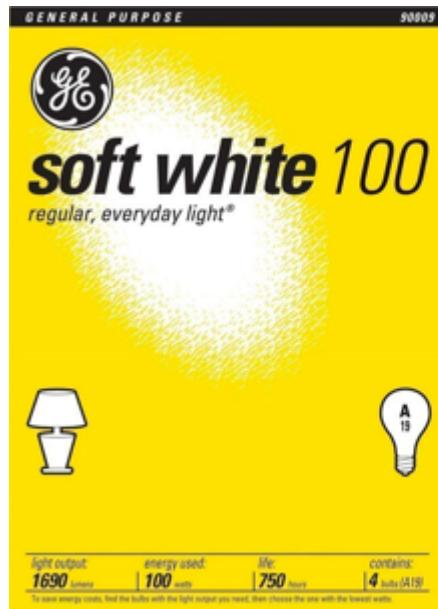
1. Industrial IoT and Data @ GE
2. Introduction to Data Pipelines
3. Sample Data Pipeline in Go
4. Resiliency Testing our Data Pipeline
5. Results and takeaways



1. Industrial IoT and Data @ GE
2. Introduction to Data Pipelines
3. Sample Data Pipeline in Go
4. Reliability Testing our Data Pipeline
5. Results and takeaways



# Who here has owned a GE Fridge, Lightbulb, or Microwave?



Actually, we make  
HUGE Industrial assets...



MAX CRANE STRUCTURE 12.5m





2029

GECX

# Fun facts about GE

---

- GE Power generates roughly 33% of the world's electricity
- Every two seconds, an aircraft powered by GE takes off
- 35,000 wind turbines globally
- 25% of all global hydropower



# Industrial Internet of Things (IIoT)

---

- GE Assets produce petabytes of useful data
- Valuable for gaining insights into these assets
- Optimize w/ Asset Performance Management (APM)
- Small percent increase in efficiency saves billions of dollars



# Predix

---

- Multi-cloud platform for the Industrial Internet of Things
- Many services and applications optimized for industrial data

# PREDIX



# Some Customers

---

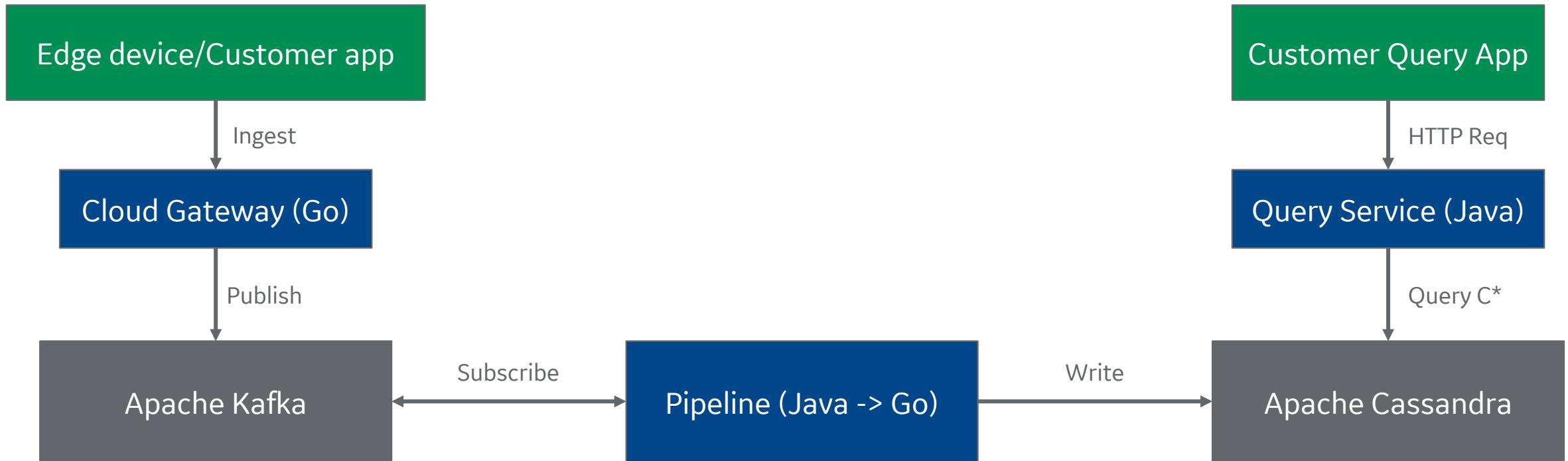
- Schindler
- Exelon
- Rosneft
- BP
- GE Power
- GE Aviation
- GE Renewables



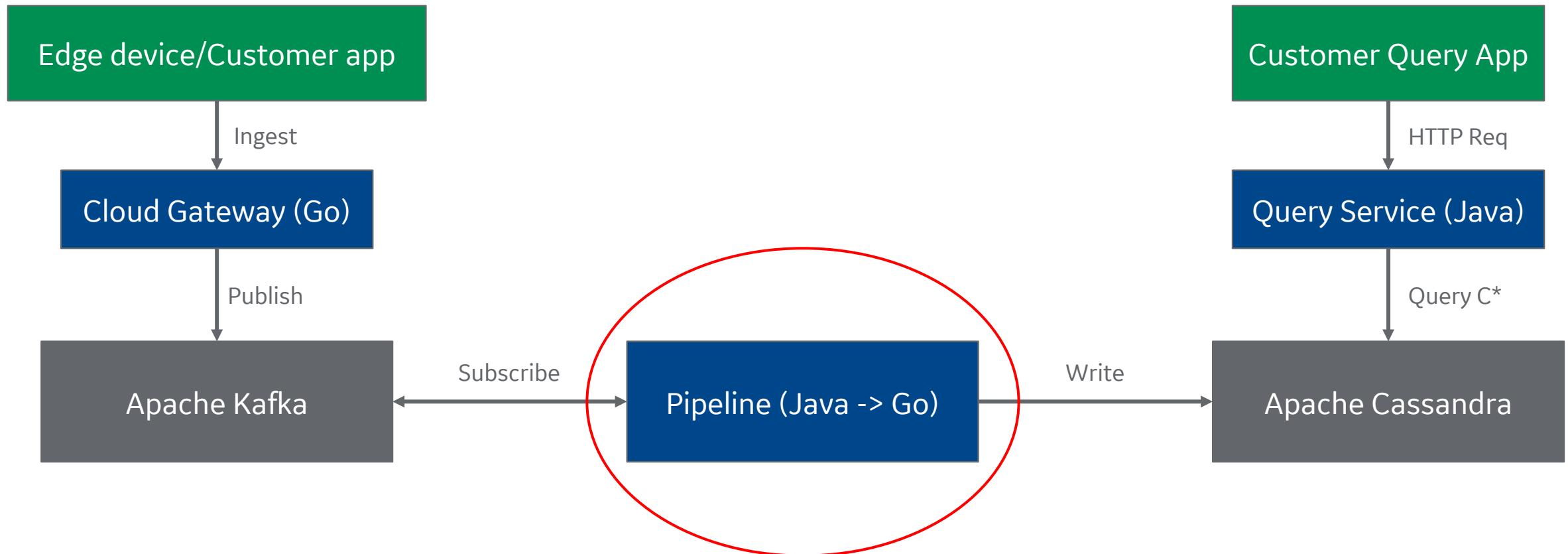
**Schindler**



# Monitoring and Diagnostics Architecture



# Monitoring and Diagnostics Architecture



# Data Pipeline component (Java version)

---

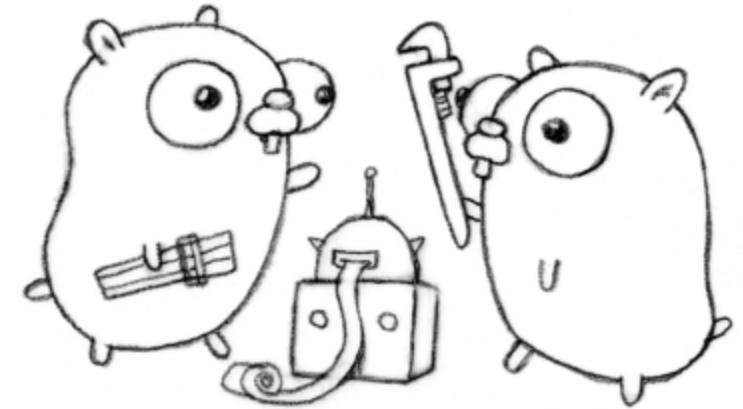
- **Stack:**
  - Stateful Java app
  - Java and Apache Apex
- **Largest deployment (out of several)**
  - 150 Cassandra nodes
  - 30 Kafka nodes
  - 144 Apache Apex containers
- **Results:**
  - 900,000 C\* writes/sec (peak)



# Data Pipeline component (Go version)

---

- **Stack:**
  - Stateless Go app
  - Cloud Foundry application (k8s in some newer envs)
- **Prod Configuration (smaller):**
  - 9 Cassandra nodes
  - 4 Kafka nodes
  - 32+ Go-Pipeline instances
- **Results:**
  - 450,000 C\* writes/sec (peak)



# Comparison (Java vs. Go)

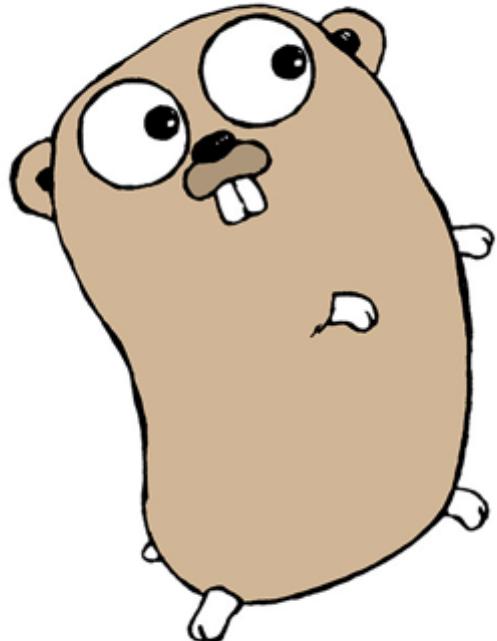
---

Pipeline version	Kafka nodes	C* nodes	Pipeline nodes	Throughput (writes/sec)	AVG Throughput (writes/day)	Environment
Java & Apache	30	<b>150</b>	144	900,000 (peak)	77,760,000,000	Largest production environment
Go (actual numbers)	4	<b>9</b>	32	450,000 (peak)	38,880,000,000	Performance environment
Go (projected)	30	<b>40</b>	128	1,800,000 (peak)	155,520,000,000	Planned production environment



# Rewrite motivations

---



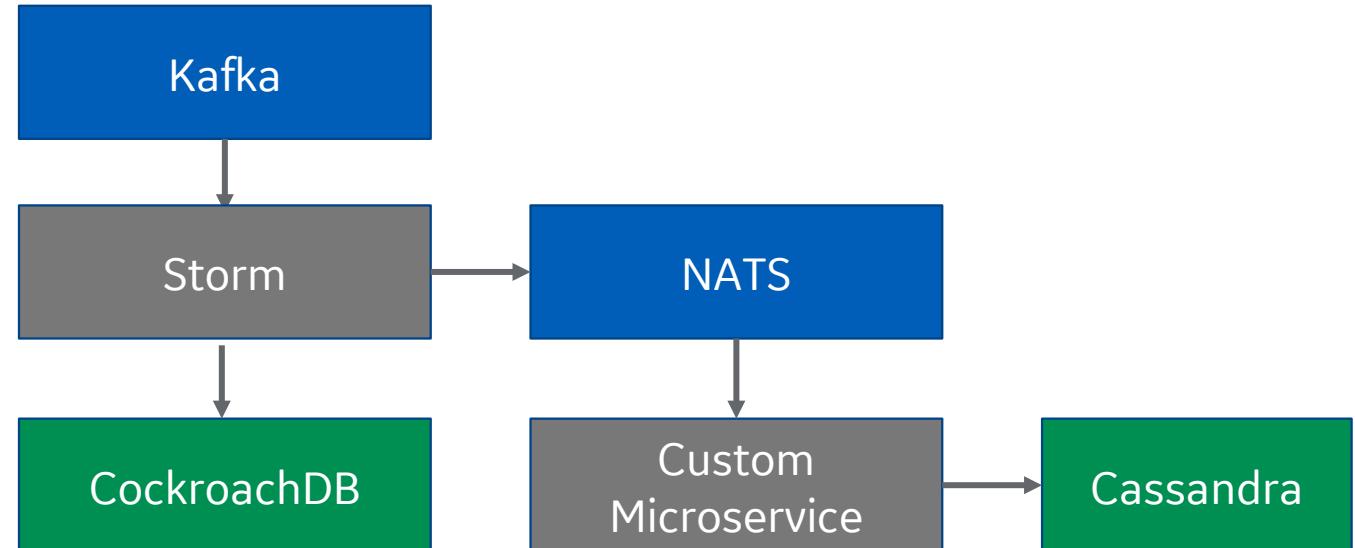
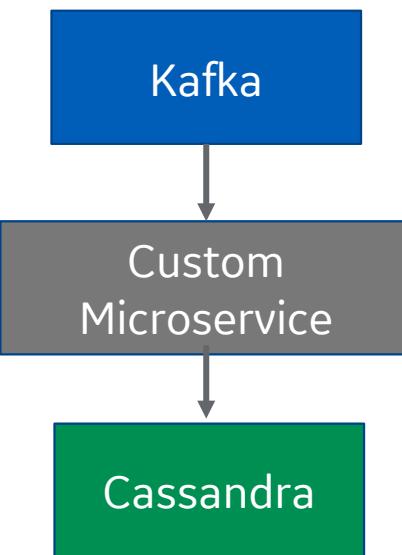
- Change in service vision/purpose
  - Originally – wanted customer specific data models/parsing
  - Now – standard data model, parsing
- Operational cost (\$\$\$)
  - Managing a Hadoop cluster
  - Resources (RAM/CPU/Disk)
- Moving towards Kubernetes
  - Simple Go Microservice
- We love Go!

1. Data @ GE
2. **Introduction to Data Pipelines**
3. Sample Data Pipeline in Go
4. Reliability Testing our Data Pipeline
5. Results and takeaways



# Introduction to Data Pipelines

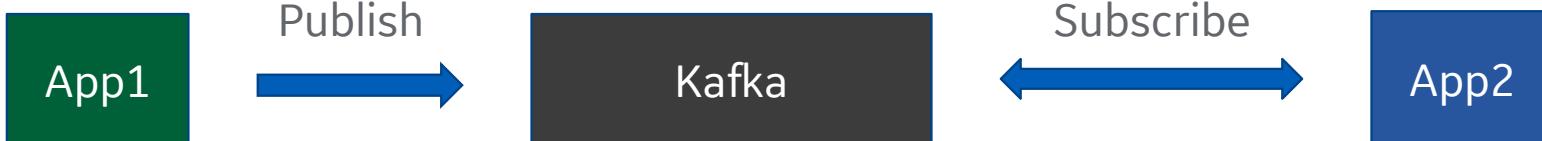
- Move data from one system to another
- Perform transformations and business logic



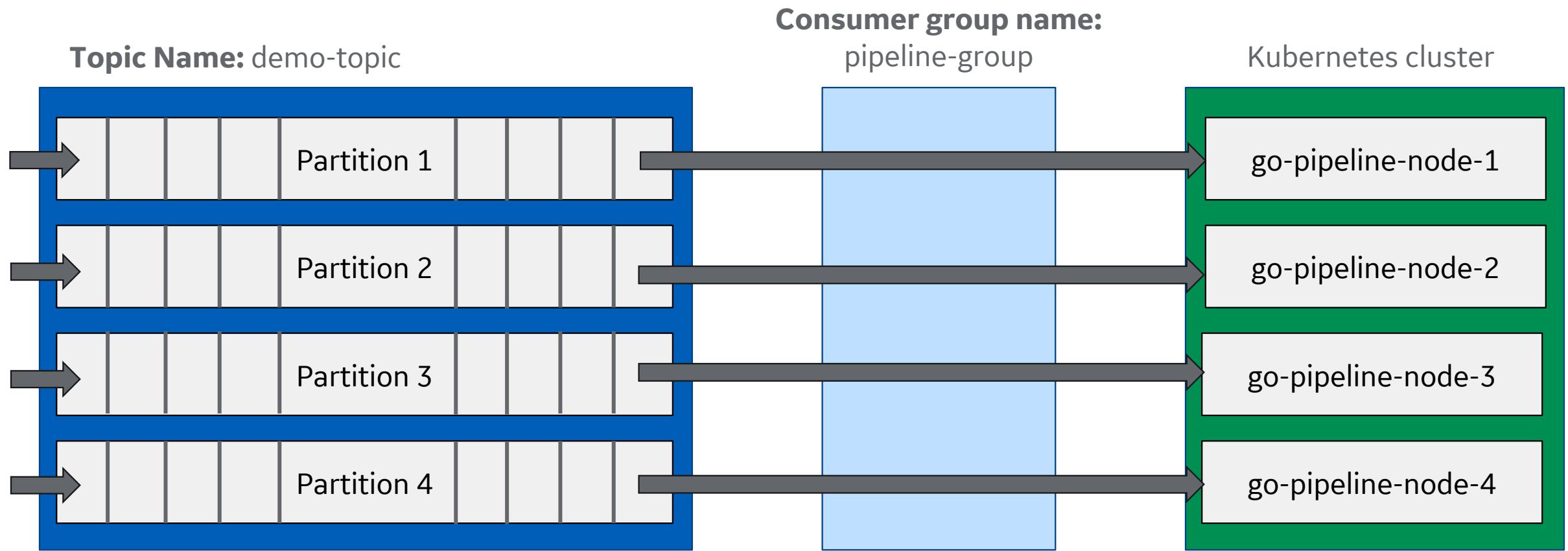
# Data source: Apache Kafka

---

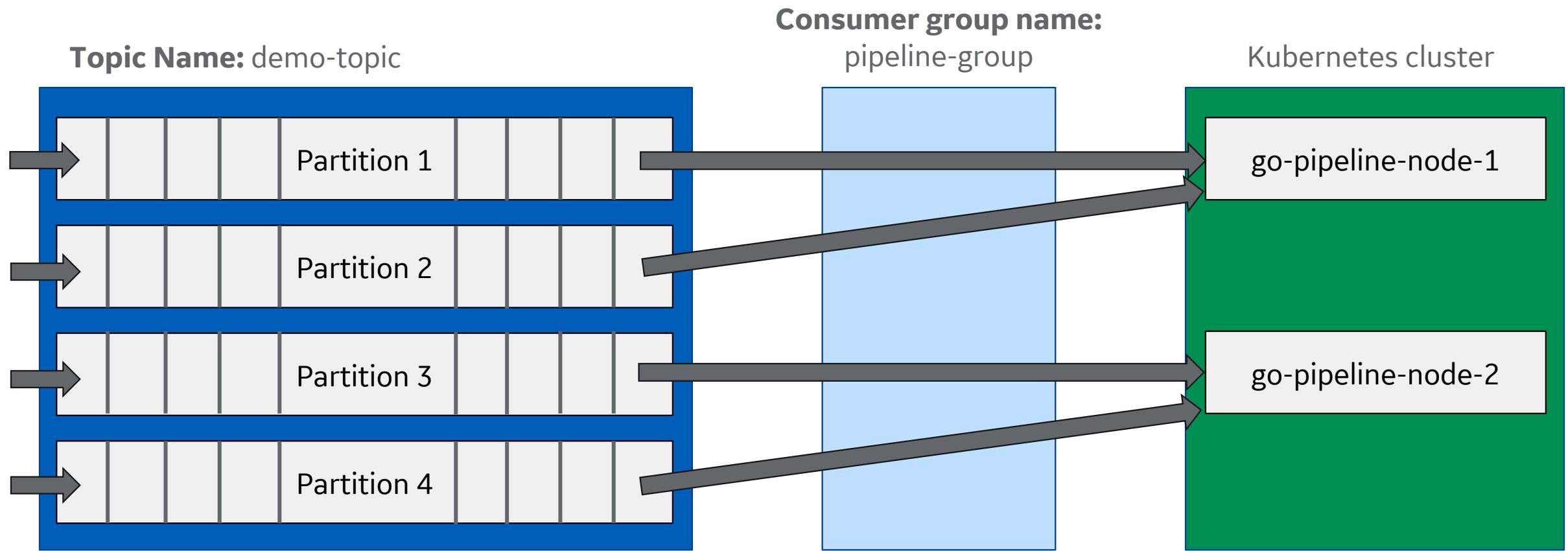
- Publish/Subscribe messaging system
- Parallelized with topic partitions
- High throughput
- Very widely used
- Java, open source - [github.com/apache/kafka](https://github.com/apache/kafka)



# Consumer groups Example: 1:1



# Consumer groups Example: 2:1



# Using Kafka with Go

---

- Many libraries
  - [github.com/Shopify/sarama](https://github.com/Shopify/sarama) + [github.com/bsm/sarama-cluster](https://github.com/bsm/sarama-cluster)
  - [github.com/confluentinc/confluent-kafka-go](https://github.com/confluentinc/confluent-kafka-go)
  - [github.com/segmentio/kafka-go](https://github.com/segmentio/kafka-go) (June 2017)
- Chose Sarama + Sarama Cluster
  1. No CGo dependency
  2. Most mature library (at the time)
  3. Wrote internal tooling + documentation around it for ease of use
- Pick what works for you



# Data store: Apache Cassandra

---

- Column-oriented database
- Fault Tolerant - replicated
- Scalable
  - Apple: over 75,000, over 10 PB of data
  - Netflix: 2,500 nodes, 420 TB, 1 trillion requests
- Java, open source [github.com/apache/cassandra](https://github.com/apache/cassandra)



# Go and Cassandra

---

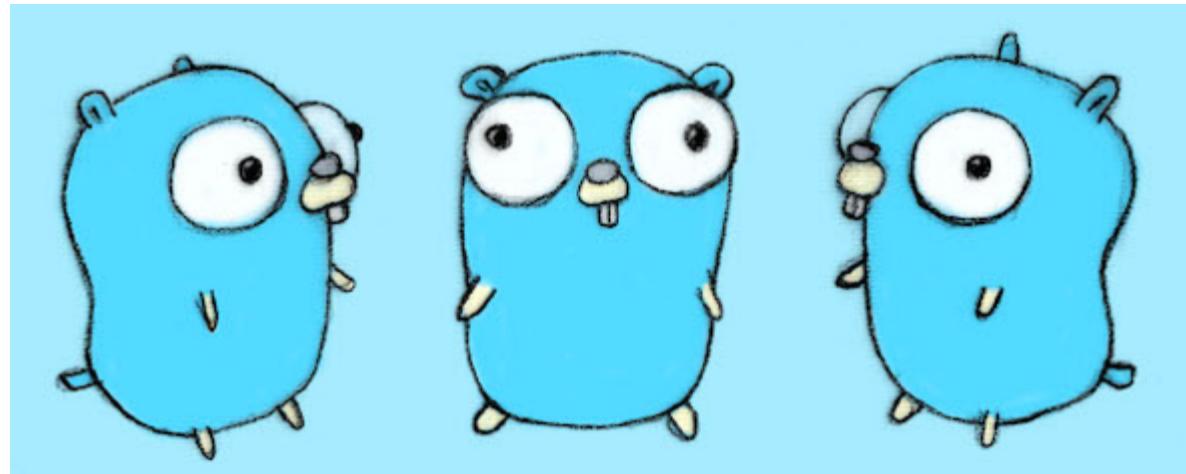
- [github.com/gocql/gocql](https://github.com/gocql/gocql)
- For high performance data bindings:
  - [github.com/scylladb/gocqlx](https://github.com/scylladb/gocqlx)



1. Data @ GE
2. Introduction to Data Pipelines
- 3. Sample Data Pipeline in Go**
4. Reliability Testing our Data Pipeline
5. Results and takeaways



By show of hands...  
Who likes the Go Gopher?



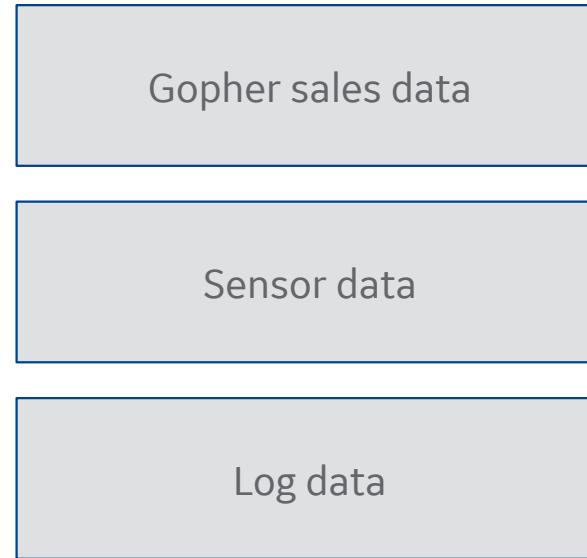
# I bring to you... Gophers “R” Us

---



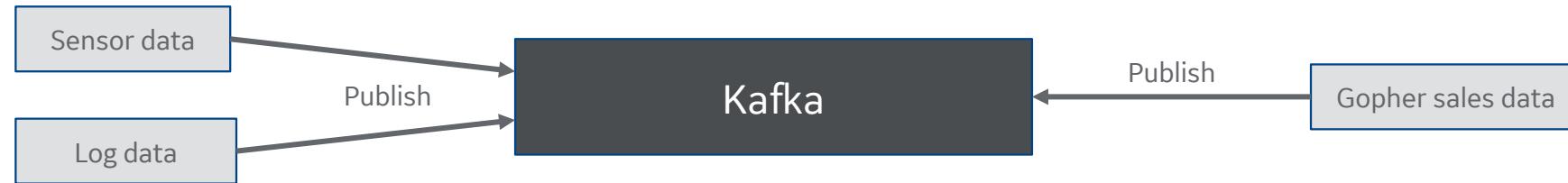
# Example: Data Pipeline (Gophers “R” Us)

---

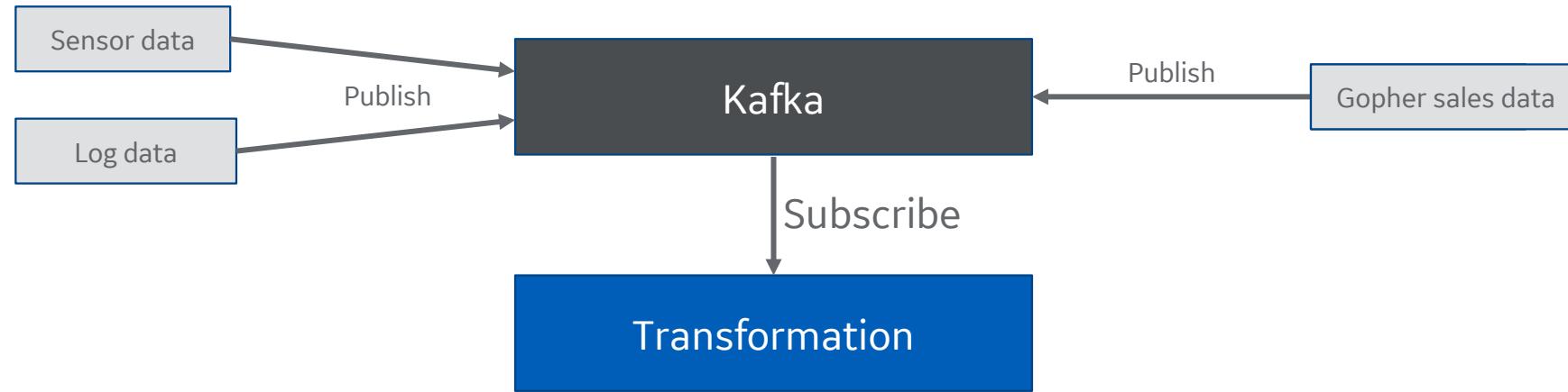


# Data pipeline architecture

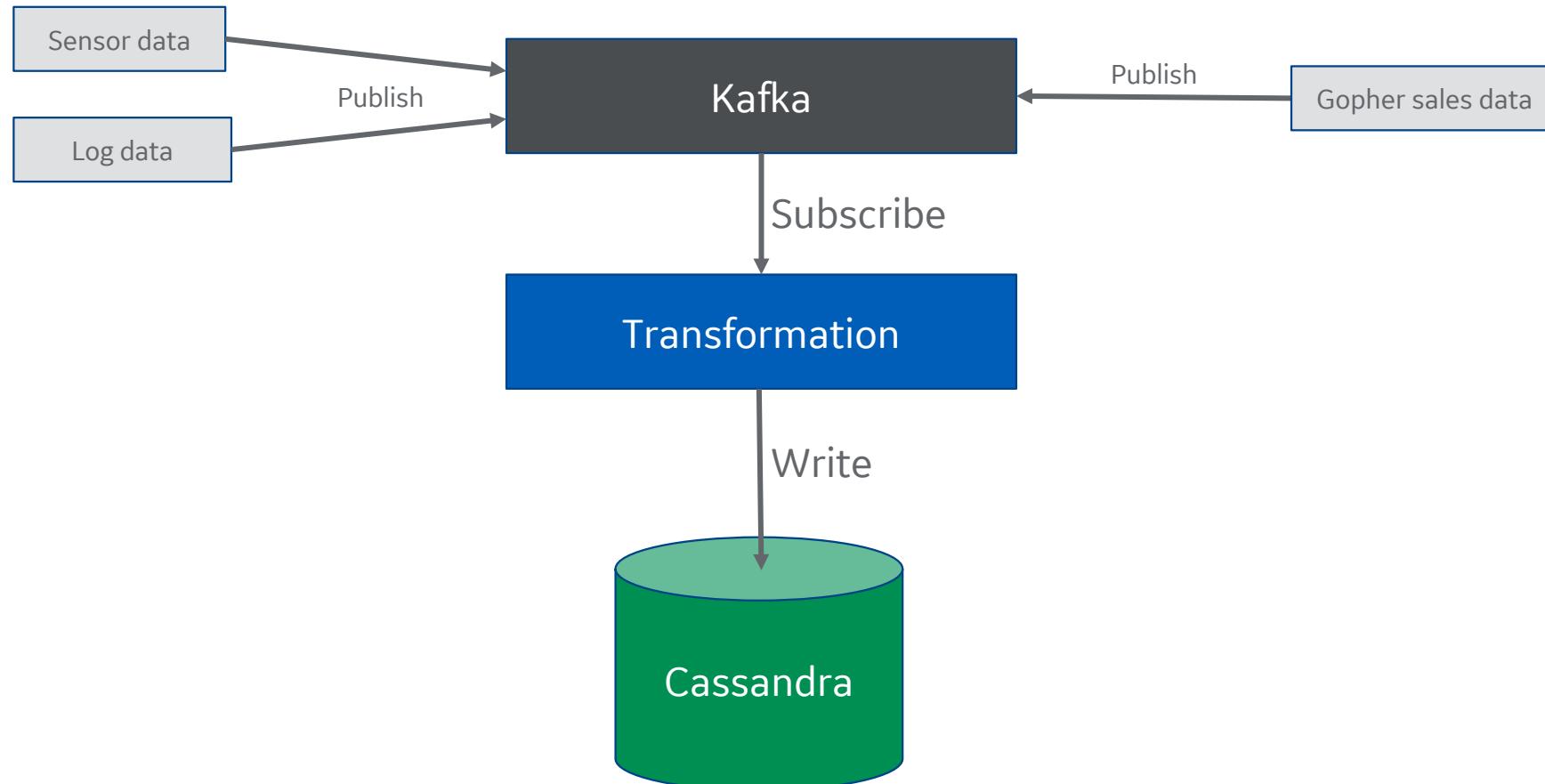
---



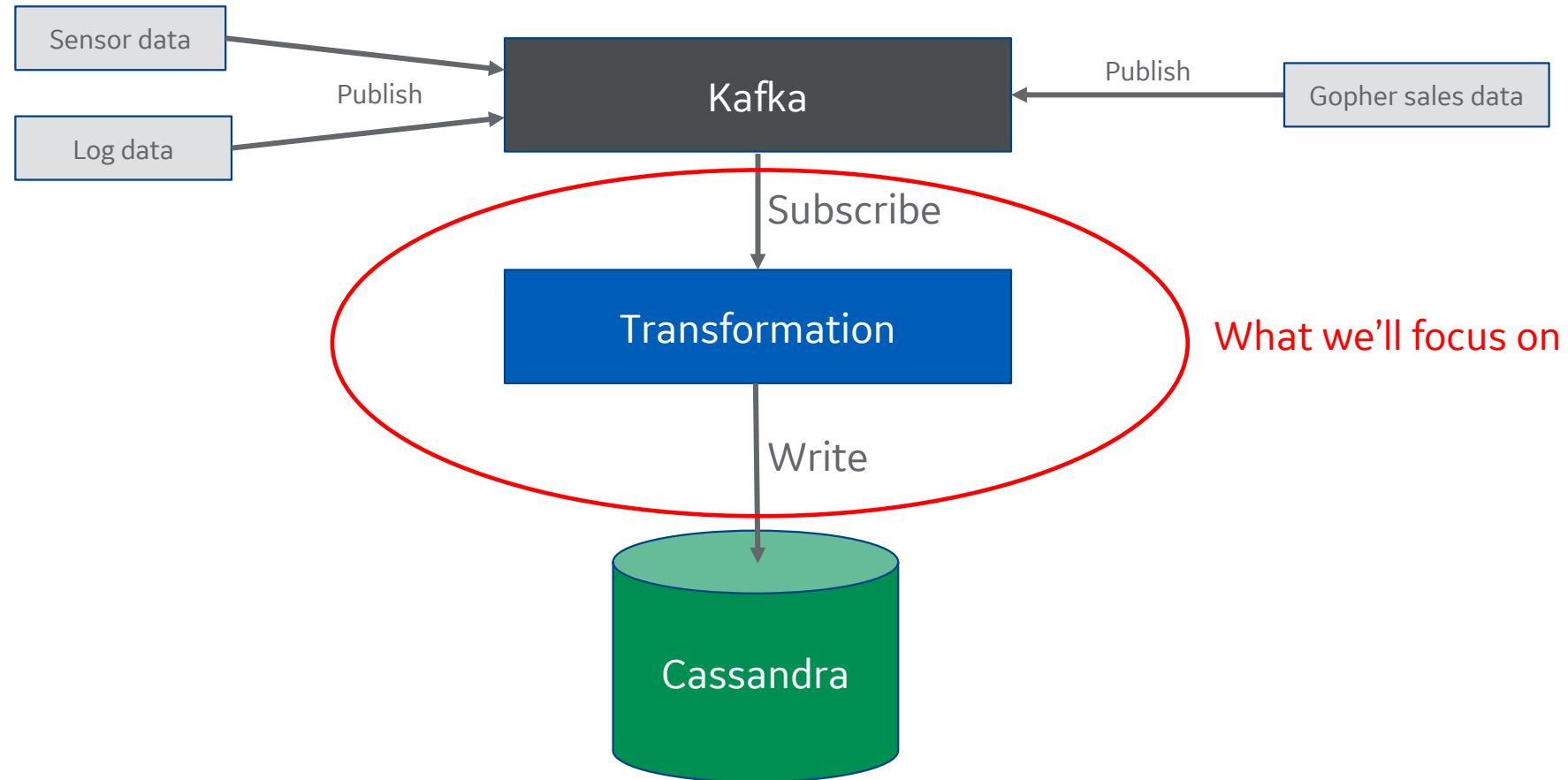
# Data pipeline architecture



# Data pipeline architecture



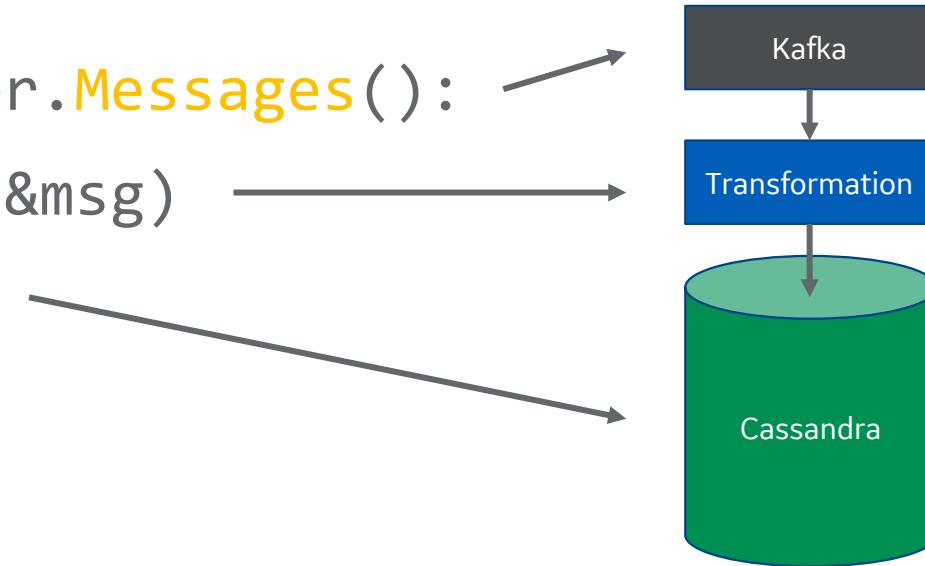
# Data pipeline architecture



# Simplified Application Flow: 3 Easy Steps

---

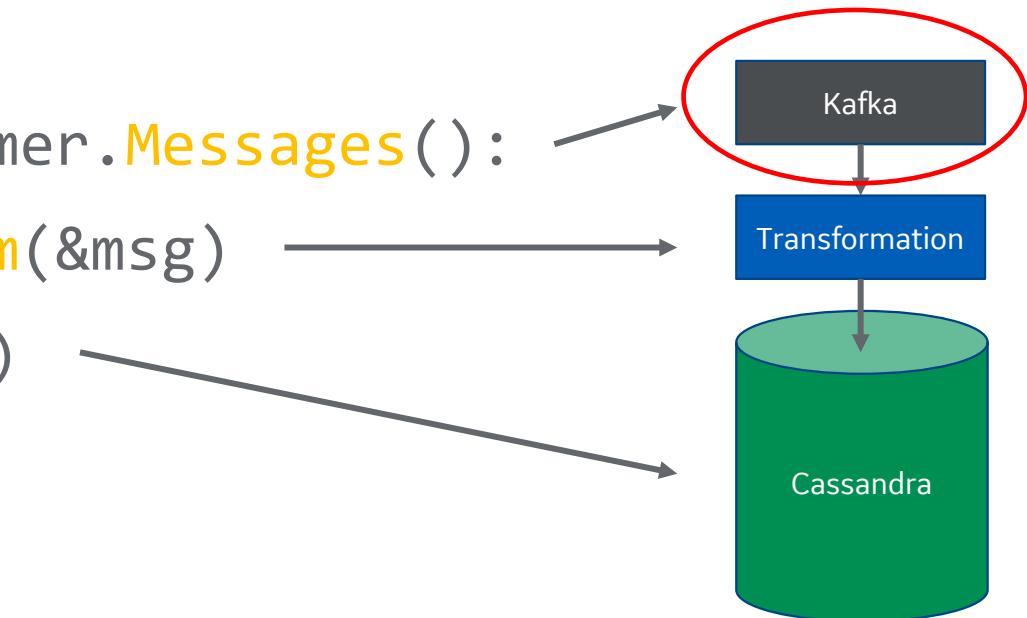
```
for {  
    select {  
        case msg := <-consumer.Messages():  
            event := Transform(&msg)  
            sink.Write(&event)  
    }  
}
```



# Subscribing to Kafka

---

```
for {  
    select {  
        case msg := <-consumer.Messages():  
            event := Transform(&msg)  
            sink.Write(&event)  
    }  
}
```



# Subscribing to Kafka

---

```
for {
    select {
        case msg, ok := <-consumer.Messages():
            log.Printf("Messages received: %v", *msg)

        case notification, ok := <-consumer.Notifications():
            log.Printf("Rebalance received: %v", *notification)

        case err, ok := <-consumer.Errors():
            log.Printf("Error received: %s", err.Error())
    }
}
```

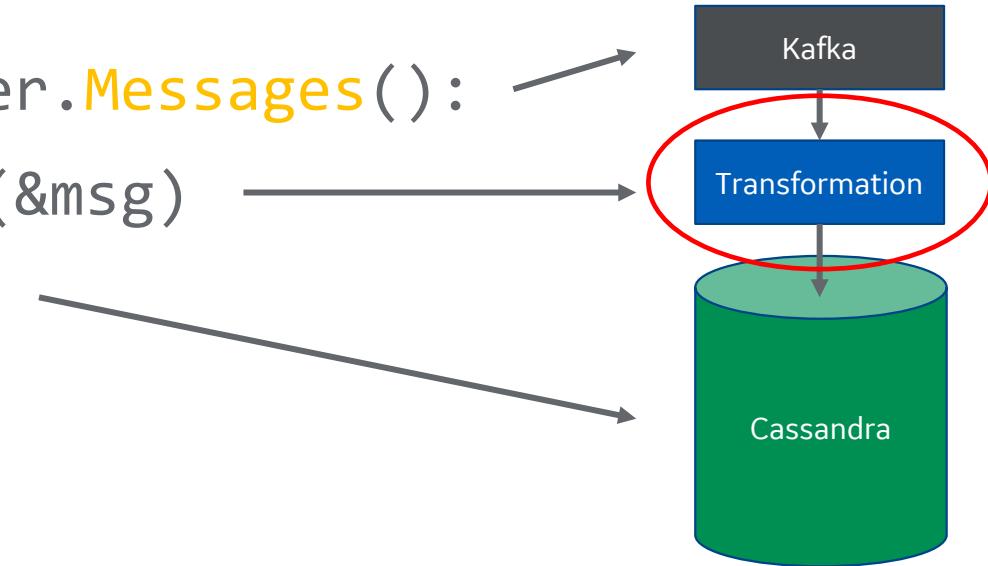
1. **Messages channel:** Data from Kafka
2. **Notifications channel:** Rebalance notifications
3. **Errors channel:** Errors in offset management



# Handling messages

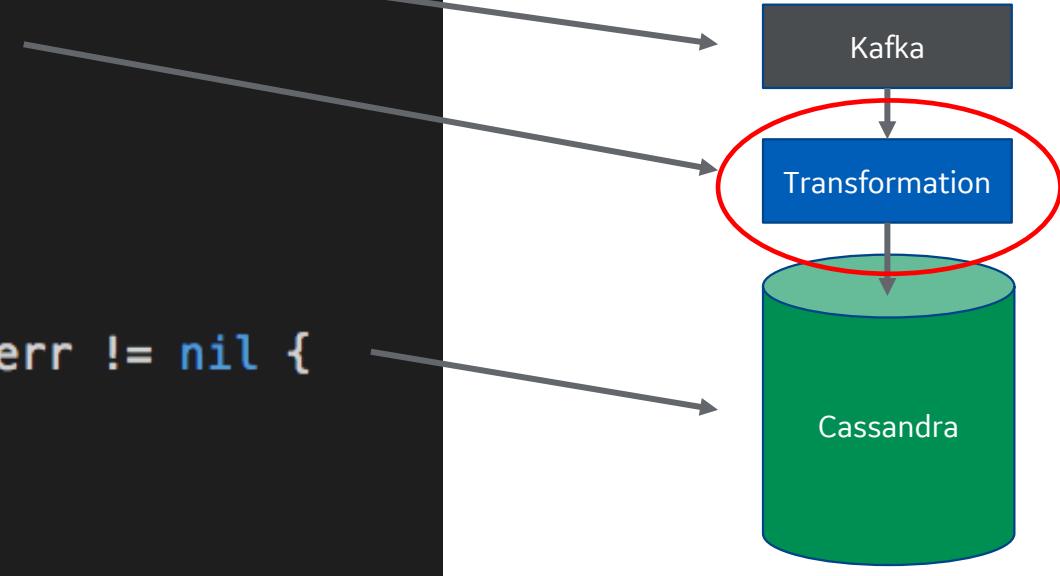
---

```
for {  
    select {  
        case msg := <-consumer.Messages():  
            event := Transform(&msg)  
            sink.Write(&event)  
    }  
}
```



# Handling messages

```
case msg, _ := <-consumer.Messages():
    event, err := Transform(msg)
    if err != nil {
        ...
    }
    if err := sink.Write(event); err != nil {
        ...
    }
    consumer.MarkOffset(msg, "")
```



# Transforming our message

Gophers “R” Us complicated business logic

---

```
// Transform take a sarama Message and transforms it based
// on our business logic
func Transform(msg *sarama.ConsumerMessage) (*StoreEvent, error) {
    var event StoreEvent
    if err := json.Unmarshal(msg.Value, &event); err != nil {
        ...
    }
    event.AssignDepartment()

    return &event, nil
}
```



# Transforming our message

Gophers “R” Us complicated business logic

---

```
// AssignDepartment assigns a department for an item based on it's name
func (se *StoreEvent) AssignDepartment() {
    switch se.ItemName {
        case "blue-gopher", "pink-gopher", "purple-gopher":
            se.ItemDepartment = deptGophers

        case "gophercon-us-sticker", "gosf-sticker", "gophercon-uk-sticker":
            se.ItemDepartment = deptStickers

        case "gophercon-us-shirt", "gophercon-uk-shirt":
            se.ItemDepartment = deptClothing

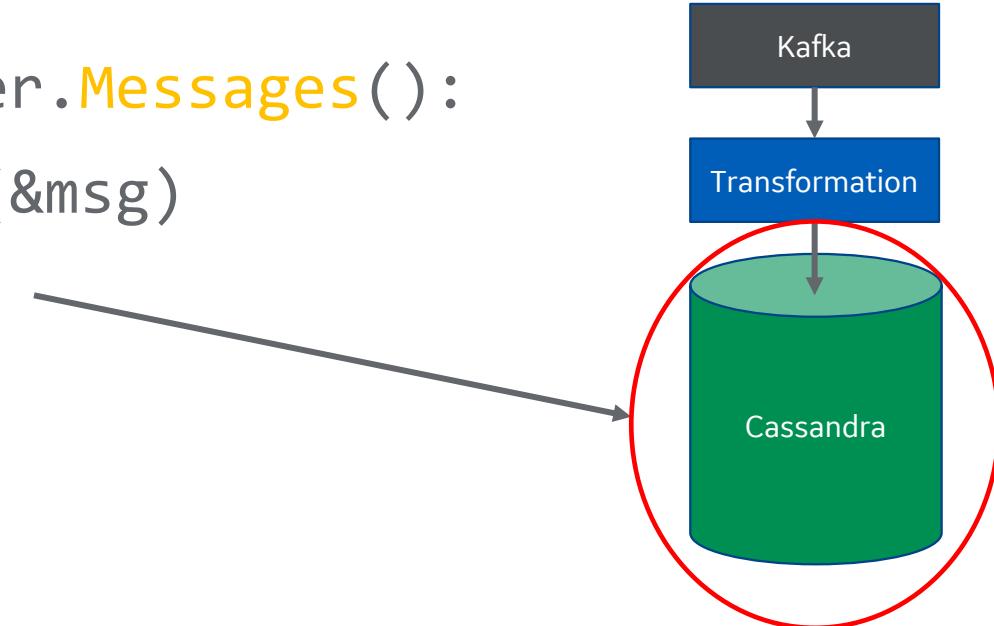
        default:
            se.ItemDepartment = deptMisc
    }
}
```



# Writing to Cassandra

---

```
for {  
    select {  
        case msg := <-consumer.Messages():  
            event := Transform(&msg)  
            sink.Write(&event)  
    }  
}
```



# Writing to Cassandra

---

```
// Write persists an event to Cassandra
func (cs *CassandraSink) Write(e *StoreEvent) error {
    stmt, names := qb.Insert(fmt.Sprintf("%s.%s", cs.Keyspace, cs.StoreEventTable)).
        Columns("name", "price", "dept").
        ToCql()
    q := gocqlx.Query(cs.Session.Query(stmt), names).Bind(e.ItemName, e.Price, e.ItemDepartment)
    return q.Exec()
}
```



# Graceful shutdown

```
signals := make(chan os.Signal, 1)
for {
    select {
    case sig, _ := <-signals:
        switch sig {
        case syscall.SIGTERM:
            log.Printf("Gracefully shutting down. Goodbye!")
            consumer.Close()
            sink.Session.Close()
            os.Exit(0)
        }
    }
}
```

1. Setup channel for listening to OS Signals
2. Listen on os signals channel
3. Handle SIGTERM for graceful shutdown
  - k8s sends this signal when containers are stopped, scaled down, etc
4. Grace period (10-30s)
  - Configurable in k8s:  
*terminationGracePeriodSeconds*
  - Container killed after this period  
(or you can os.Exit(0) manually)



# Deploying to Kubernetes

```
---  
kind: Deployment  
apiVersion: extensions/v1beta1  
metadata:  
  name: gophers-r-us-pipeline  
spec:  
  template:  
    metadata:  
      labels:  
        app: gophers-r-us-pipeline  
        id: "1"  
    spec:  
      containers:  
      - name: gophers-r-us-pipeline  
        image: ggriffiths/gophers-r-us-pipeline:v1  
        ports:  
        - containerPort: 8080  
      terminationGracePeriodSeconds: 15
```



1. Data @ GE
2. Introduction to Data Pipelines
3. Sample Data Pipeline in Go
- 4. Reliability Testing our Data Pipeline**
5. Results and takeaways



# Reliability Testing

---

- Systems fail
  - How does our pipeline behave during times like these?
  - How can we remedy these failures?
  - How can we ensure customer data is not lost?
- Embrace failure scenarios
  - or they will embrace you at 3 AM
- If this interests you: Google SRE Book  
[landing.google.com/sre/book/index.html](https://landing.google.com/sre/book/index.html)



# Reliability Testing our Data Pipeline

---

- What can fail?
  - Data pipeline node(s)
  - Kafka node(s)
  - Cassandra node(s)
  - Kubernetes can fail
- How does our pipeline behave when this happens?
  - We can write a test
- How can we remedy these failures?



# Reliability Test example

```
Describe("Given our Data pipeline is running successfully, reading from Kafka and writing to C*", func() {
    Context("When our Cassandra cluster goes down", func() {
        It("Then no data should be lost", func() {
            initialCount := cassandraRowCount("keyspace", "item_table")

            desiredMessageCount := 50
            for i := 0; i < desiredMessageCount; i++ {
                pushMessages("demo-topic", 1)

                // Take Cassandra down for 20 seconds
                switch i {
                    case 10:
                        pauseDockerImage("cassandra")

                    case 30:
                        unPauseDockerImage("cassandra")
                }

                time.Sleep(1 * time.Second)
            }

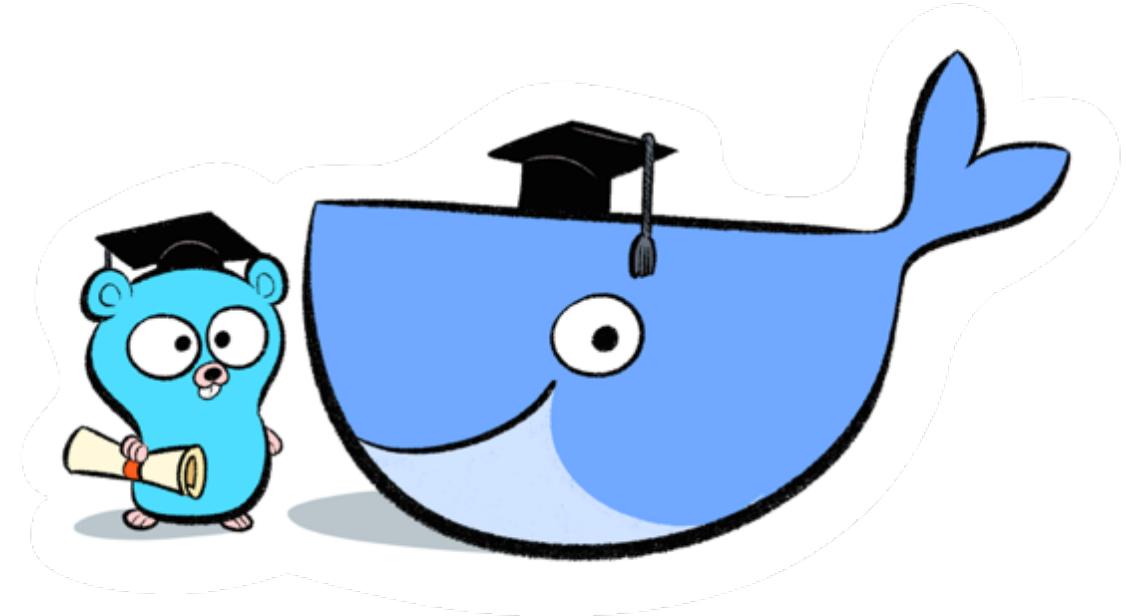
            finalCount := cassandraRowCount("keyspace", "item_table")
            Expect(finalCount - initialCount).To(Equal(desiredMessageCount))
        })
    })
})
```



# Reliability Test example

---

- What else can we test?
  - Partial cluster failures
  - Full cluster failures
  - Pipeline failures
  - High load
  - etc
- Integration Test with Docker!



1. Data @ GE
2. Introduction to Data Pipelines
3. Sample Data Pipeline in Go
4. Reliability Testing our Data Pipeline
5. **Results and takeaways**



# Results & Takeaways

---

- **Building Data Pipelines in Go**
  - Simple to get a small app up and running
  - Good enough community support for Kafka, Cassandra, etc
  - Use chan of os.Signals for graceful shutdown
- **Reliability testing**
  - Use docker to integration/reliability test
  - Understand how your system behaves during failures scenarios
- **Go Pipelines at GE Digital**
  - Replaced our existing Java pipeline for lower operational cost, simplicity, and performance



# Questions?

Or funding for Gophers "R" Us?



PREDIX