# Is Kubernetes a Database?

**Ryanne Dolan**
*(LinkedIn)*

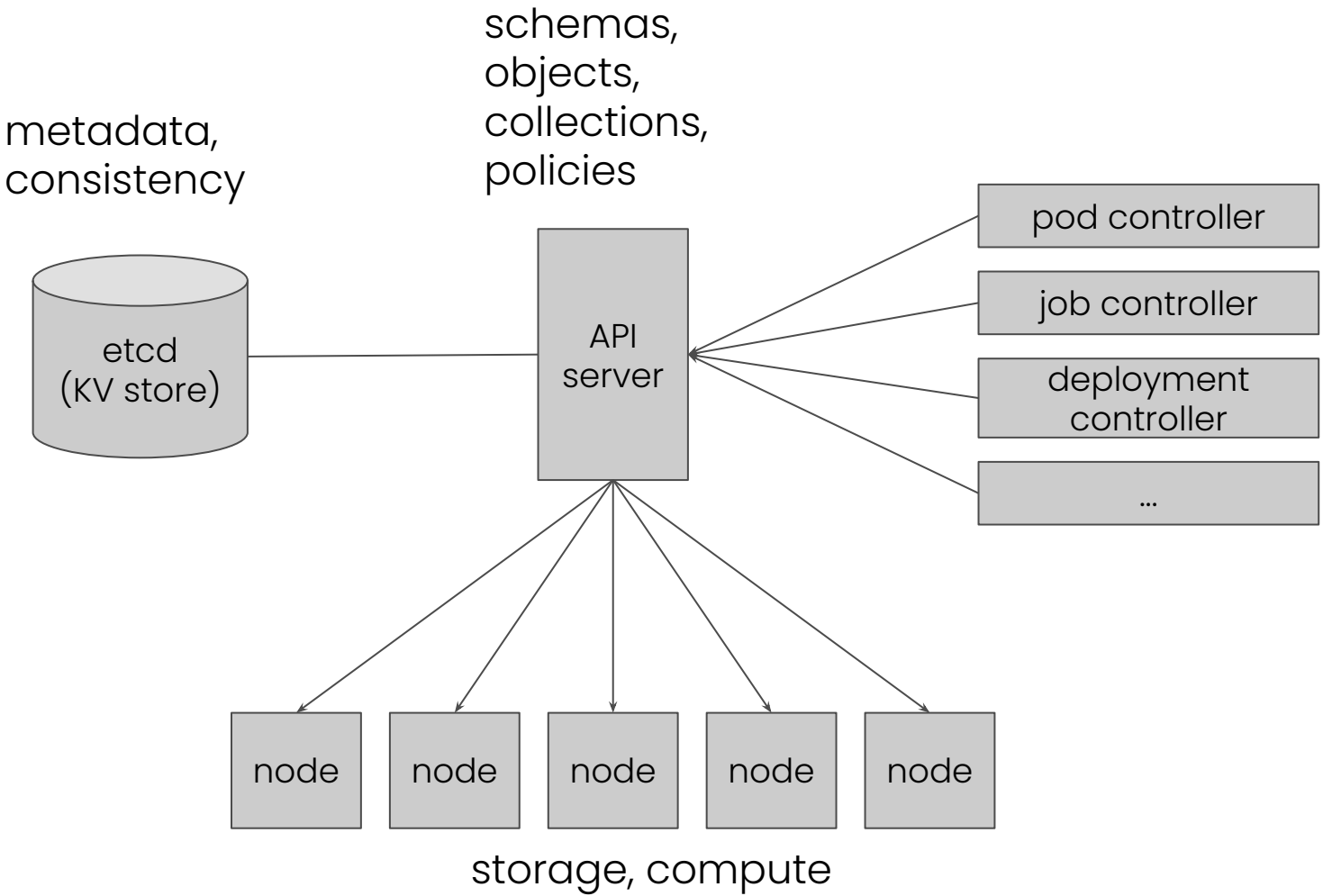
# Kubernetes

"K8s"

Workload orchestrator ("Pods")

Platform for containerized apps

Often: microservices


schemas,
objects,
collections,
policies
metadata,
consistency
etcd
(KV store)
API
server
pod controller
job controller
deployment
controller
...
node
node
node
node
node
storage, compute

# Hadoop

Zookeeper

HDFS

Hive

Yarn

MapReduce

Spark

Ranger

Kerberos

...

(Big Data)

# Kubernetes

scheduler

RBAC

Pods

Deployments

PersistentVolumes

...

(microservices, mostly)

# Data on K8s - the new Hadoop?

# Lots of missing features...

- storage ✅
- compute ✅
- consistency ✅
- schemas ✅
- CRUD API ✅
- tables 👀
- SQL queries 👀
- DML, DDL 👀

# Building a SQL interface to Kubernetes

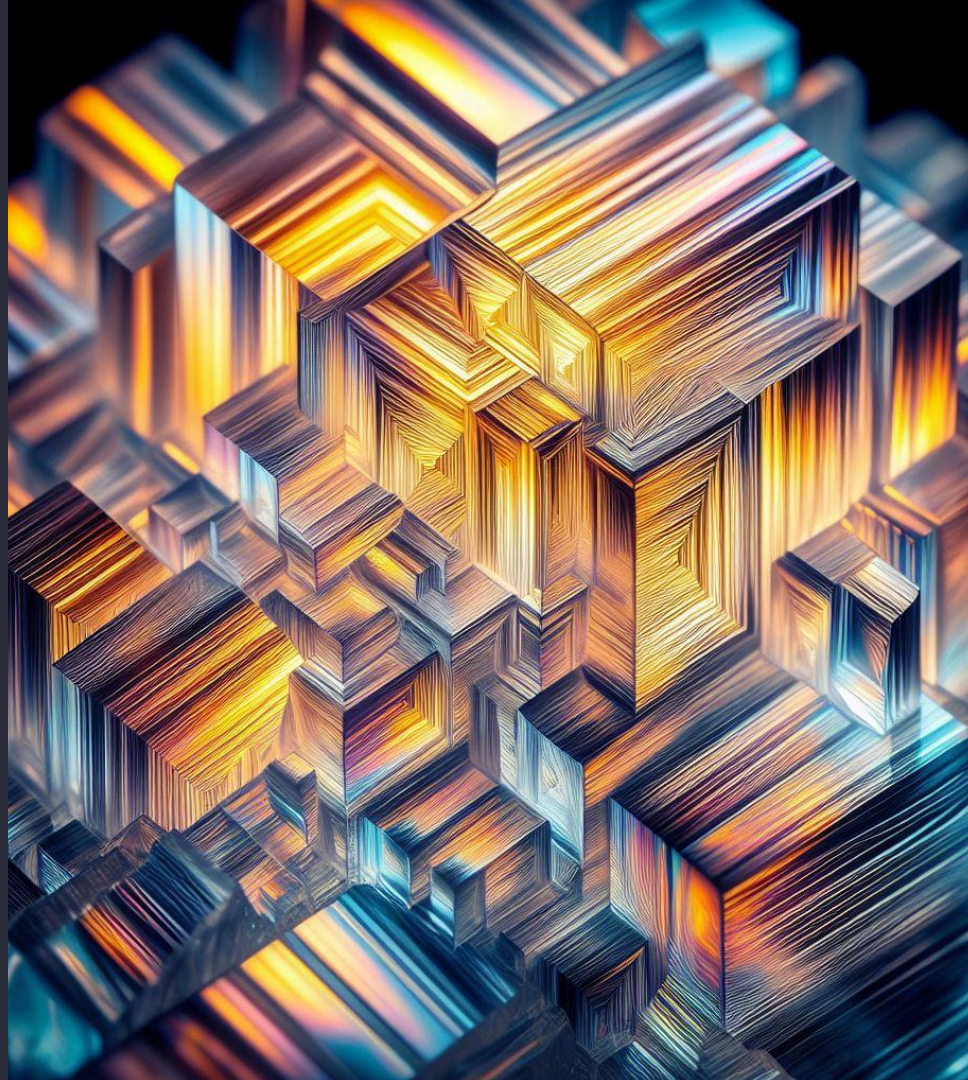
```
> SELECT * FROM KUBERNETES.PODS WHERE STATUS = 'Ready';

+-----------+--------------+
|  NAME     |  STATUS      |
+-----------+--------------+
|  pod-123  |  Ready       |
+-----------+--------------+
|  pod-456  |  Ready       |
+-----------+--------------+
```

# Apache Calcite

Framework for building databases

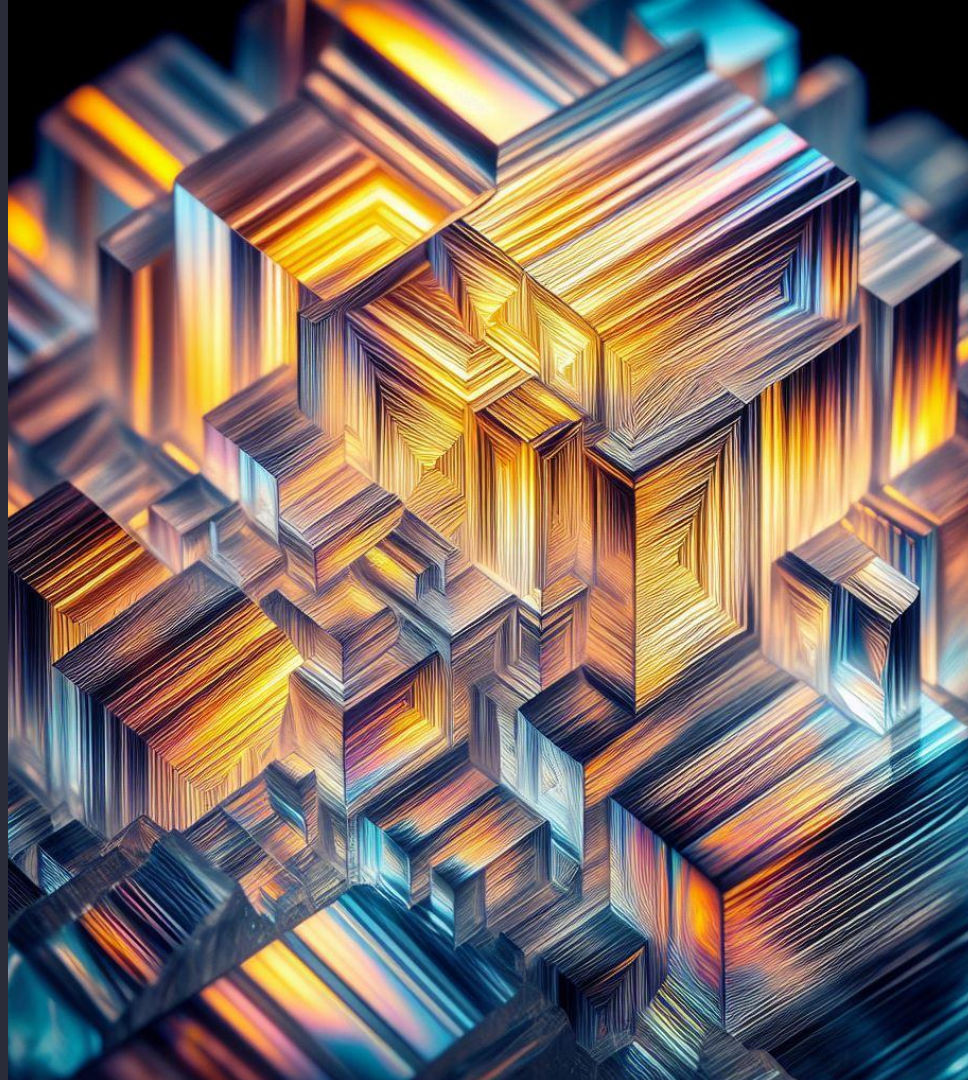
"Database without a storage layer"

What if K8s is the storage layer?



# Apache Calcite

- SQL parser, planner, optimizer
- APIs for tables, databases, catalogs, data types, etc
- Support for Java types, Java objects, and Java collections
- JDBC driver, e.g. `jdbc:calcite:model=my-catalog.yaml`



```
$ cat my-model.yaml
schemas:
- name: FOO
  type: custom
  factory: com.example.MySchemaFactory

$ sqlline -u jdbc:calcite:model=my-model.yaml
sqlline> SELECT * FROM FOO.BAR;
```

```
public class MySchemaFactory implements SchemaFactory {
  @Override
  public Schema create(SchemaPlus parentSchema, String name, Map<String, Object> operands) {
    return new MySchema(operands);
  }
}


public class MySchema extends AbstractSchema {
  private final Map<String, Table> tableMap;
  @Override
  public Map<String, Table> getTableMap() {
    return tableMap;
  }
  --->%---
}
```

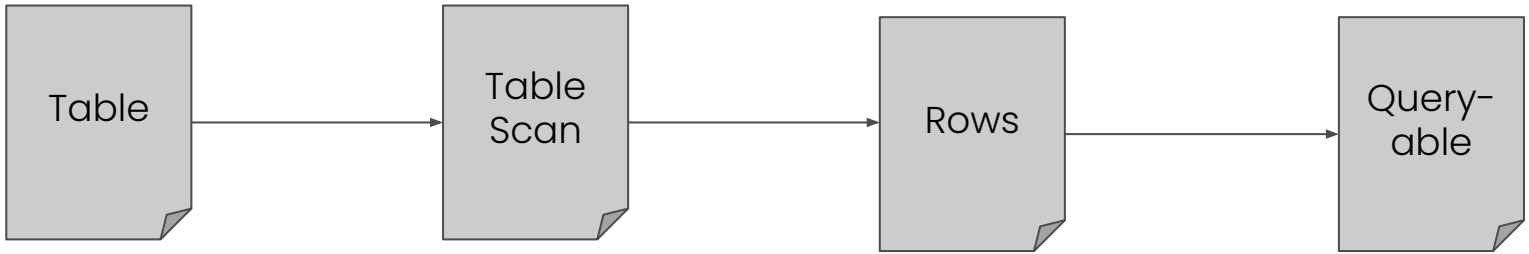


Table
Table Scan
Rows
Query-able

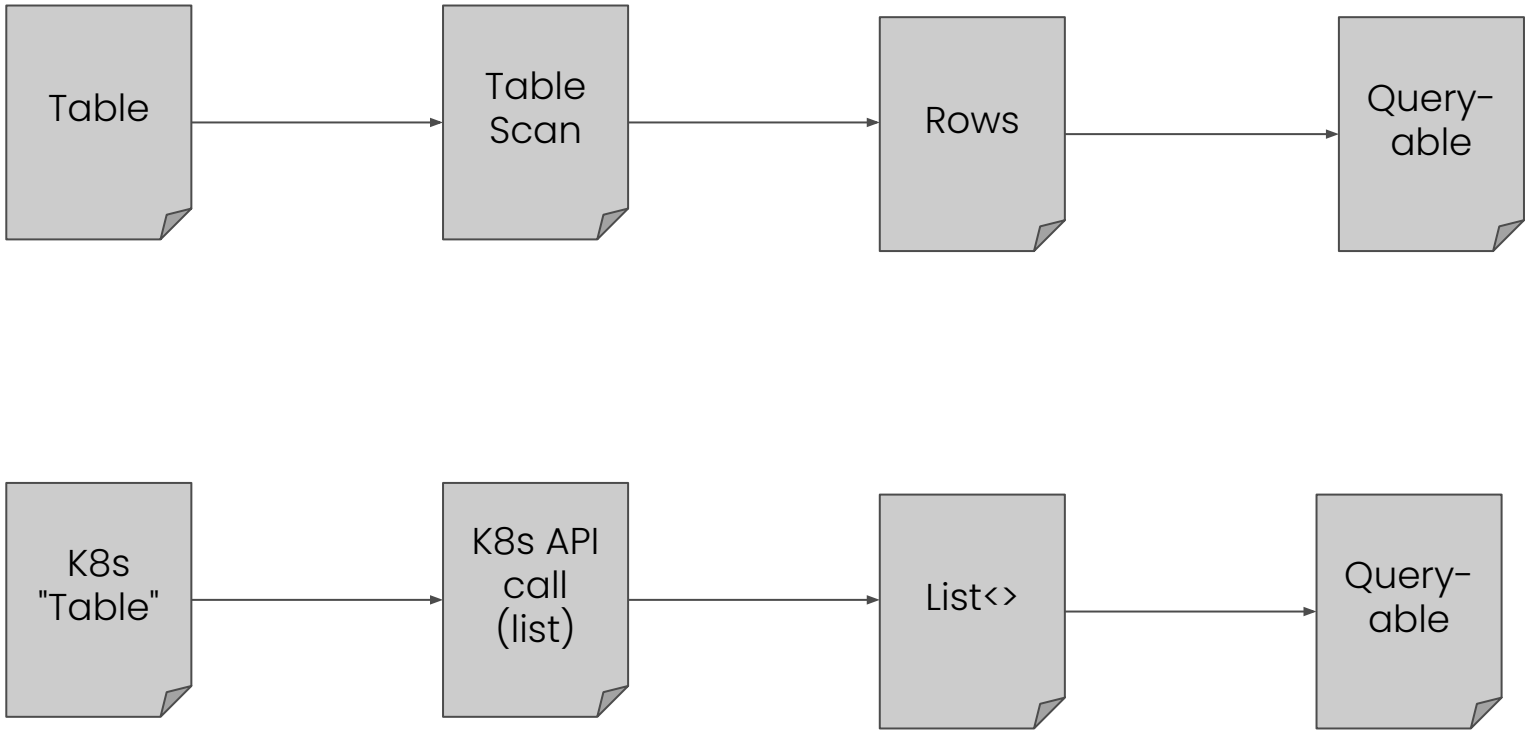


Table
Table Scan
Rows
Query-able
K8s "Table"
K8s API call (list)
List<>
Query-able

```
public class PodTable extends KubernetesTable<V1Pod, V1PodList, PodTable.Row> {

  public static class Row {
    public String NAME;
    public String STATUS;

    public Row(String name, String status) {
      this.NAME = name;
      this.STATUS = status;
    }
  }

  @Override
  public Row toRow(V1Pod pod) {
    return new Row(pod.getMetadata().getName(), ...);
  }
  --->%---
}
```

```
// get a List of Pods from Kubernetes:

GenericKubernetesApi api = new GenericKubernetesApi(V1Pod.class, V1PodList.class, ... "pods");
KubernetesApiResponse<V1PodList> resp = api.list();
List<V1Pod> pods = resp.getObject().getList();

// turn them into Tables and Rows:

PodTable podTable = new PodTable(pods);
List<PodTable.Row> rows = podTable.scan();

// Query the table:

PodTable.Row res = podTable.asQueryable().select(...).first();
```

```
public class KubernetesSchema extends AbstractSchema {
  private final Map<String, Table> tableMap = new HashMap<>();
  public KubernetesSchema() {
    tableMap.put("PODS", new PodTable(listPods()));
    tableMap.put("DEPLOYMENTS", new DeploymentTable(listDeployments());
    ...
  }
  --->%---
}

public class KubernetesSchemaFactory implements SchemaFactory {
  @Override
  public Schema create(SchemaPlus parentSchema, String name, Map<String, Object> operands) {
    return new KubernetesSchema();
  }
}
```

```
$ cat my-model.yaml
schemas:
- name: KUBERNETES
  type: custom
  factory: com.example.KubernetesSchemaFactory

$ sqlline -u jdbc:calcite:model=my-model.yaml
sqlline> SELECT * FROM KUBERNETES.PODS WHERE STATUS = 'Ready';

+-----------+--------------+
|  NAME     |  STATUS      |
+-----------+--------------+
|  pod-123  |  Ready       |
+-----------+--------------+
|  pod-456  |  Ready       |
+-----------+--------------+
```

# Building a Kubernetes JDBC Driver

```
$ sqlline -u jdbc:kubernetes:
sqlline> SELECT * FROM KUBERNETES.PODS WHERE STATUS = 'Ready';

+-----------+--------------+
|  NAME     |  STATUS      |
+-----------+--------------+
|  pod-123  |  Ready       |
+-----------+--------------+
|  pod-456  |  Ready       |
+-----------+--------------+
```

```
public class KubernetesDriver extends org.apache.calcite.jdbc.Driver {

  @Override
  protected String getConnectStringPrefix() {
    return "jdbc:kubernetes:";
  }

  @Override
  public Connection connect(String url, Properties props) throws SQLException {
    CalciteConnection connection = (CalciteConnection) super.connect(url, props);
    SchemaPlus rootSchema = connection.getRootSchema();
    rootSchema.add("KUBERNETES", new KubernetesSchema());
    return connection;
  }
}
```

```
$ sqlline -u jdbc:kubernetes:
sqlline> SELECT * FROM KUBERNETES.PODS WHERE STATUS = 'Ready';

+-----------+--------------+
|  NAME     |  STATUS      |
+-----------+--------------+
|  pod-123  |  Ready       |
+-----------+--------------+
|  pod-456  |  Ready       |
+-----------+--------------+

// bonus points:
$ kubectl sql -q "SELECT * FROM KUBERNETES.PODS WHERE STATUS = 'Ready'"
```

Pods, Deployments (cool)

Customers, Sales (cooler)

# Extending Kubernetes with Custom Resources

```
$ kubectl apply -f my-pods.yaml
pod-1 created
pod-2 created

$ kubectl apply -f my-deployments.yaml
deployment-1 created
deployment-2 created

$ helm install kafka-operator some-kafka-operator
$ kubectl apply -f my-kafka-cluster.yaml  # a "custom resource"
kafka-1 created

$ kubectl apply -f my-customers.yaml      # ?!
customer-1 created
customer-2 created
```

```
$ cat customer.crd.yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: customers.example.com
spec:
  group: example.com
  names:
    kind: Customer
    listKind: CustomerList
    plural: customers
    singular: customer
  scope: Namespaced
--->%---
```

```
--->%---
      schema:
        openAPIV3Schema:
          type: object
          properties:
            customerId:
              type: integer
              minimum: 1
            firstName:
              type: string
            lastName:
              type: string
          required:
          - customerId
--->%---
```

```
$ cat my-customers.yaml
apiVersion: example.com/v1alpha1
kind: Customer
metadata:
  name: customer-123
customerId: 123
firstName: Rick
lastName: Wakeman
---
apiVersion: example.com/v1alpha1
kind: Customer
metadata:
  name: customer-124
customerId: 124
lastName: Vangelis
```

```
$ kubectl apply -f customers.crd.yaml
customers.example.com created
$ kubectl apply -f my-customers.yaml
customer-123 created
customer-124 created

$ kubectl get customers
NAME            CUSTOMER_ID     FIRST_NAME      LAST_NAME
customer-123            123     Rick            Wakeman
customer-124            124                     Vangelis
```

# Is this a database? 🤦🦋

```
public class CustomerTable extends KubernetesTable<Customer, CustomerList, PodTable.Row> {

  public static class Row {
    public int CUSTOMER_ID;
    public String FIRST_NAME;
    public String LAST_NAME;

    --->%---
  }
  --->%---
}
```

```
$ sqlline -u jdbc:kubernetes:
sqlline> SELECT LAST_NAME FROM KUBERNETES.CUSTOMERS;

+-------------+
|  LAST_NAME  |
+-------------+
|  Wakeman    |
+-------------+
|  Vangelis   |
+-------------+
```

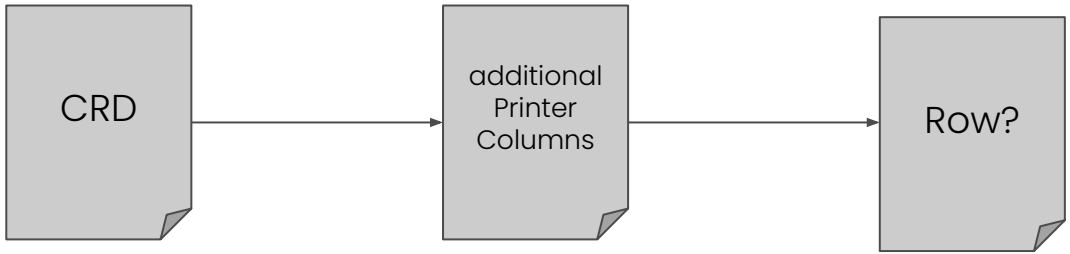
# Automatically Loading CRDs as Tables

```
$ kubectl apply -f customers.crd.yaml
customers.example.com created
$ kubectl apply -f my-customers.yaml
customer-123 created
customer-124 created

$ kubectl get customers
NAME            CUSTOMER_ID     FIRST_NAME      LAST_NAME
customer-123    123             Rick            Wakeman
customer-124    124                             Vangelis
```

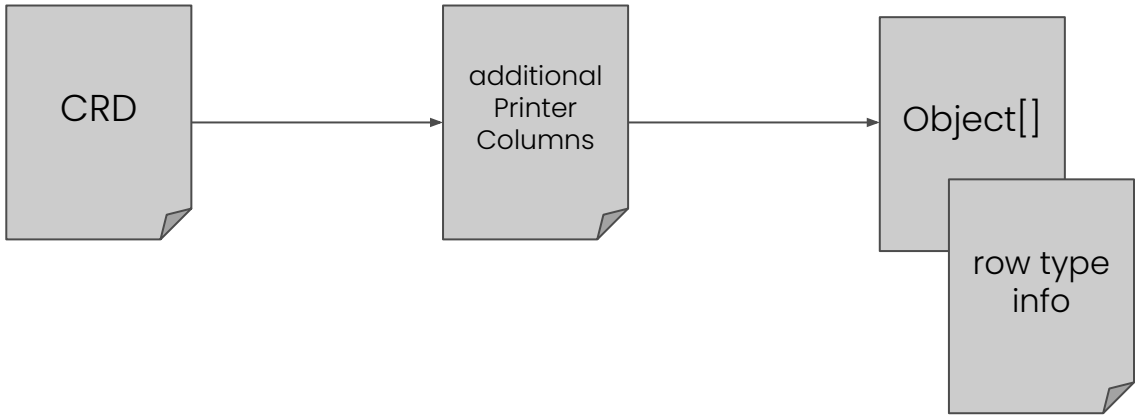
# How does kubectl do it?

```
$ cat customer.crd.yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: customers.example.com
spec:
  --->%---
  additionalPrinterColumns:
  - name: CUSTOMER_ID
    type: integer
    jsonPath: .customerId
  - name: FIRST_NAME
    type: string
    jsonPath: .firstName
  - name: LAST_NAME
    type: string
    jsonPath: .lastName
```

```
$ kubectl get customers
NAME            CUSTOMER_ID     FIRST_NAME      LAST_NAME
customer-123    123             Rick            Wakeman
customer-124    124                             Vangelis
```




```
public static class Row {
    public String CUSTOMER_ID;
    public String FIRST_NAME;
    public String LAST_NAME;
    public Row(String id, String first, String last) {
      CUSTOMER_ID = id;
      FIRST_NAME = first;
      LAST_NAME = last
    }
  }
```


CRD
additional
Printer
Columns
Object[]
row type
info

```
public class CustomResourceTable implements Table {

  @Override
  public RelDataType getRowType(RelDataTypeFactory typeFactory) {
    RelDataTypeFactory.FieldInfoBuilder builder = typeFactory.builder();
    additionalPrinterCols().forEach(x -> builder.add(x.name(), toSqlType(x.type())));
    return builder.build();
  }

  private SqlTypeName toSqlType(String colType) {
    switch(colType) {
    case "string":
      return SqlTypeName.VARCHAR;
    ...
```

```
// get a List of CRDs from Kubernetes:

GenericKubernetesApi api = new GenericKubernetesApi(V1CustomResourceDefinition.class,
    V1CustomResourceDefinitionList.class, ... "customresourcedefinitions");
KubernetesApiResponse<V1CustomResourceDefinition> resp = api.list();
List<V1CustomResourceDefinition> crds = resp.getObject().getList();

// turn them into Tables:

crds.forEach(x -> schema.add(x.spec().name(), new CustomResourceTable(x)));
```

```
$ kubectl apply -f sales.crd.yaml
sales.example.com created
$ kubectl apply -f my-sales.yaml
sale-1000 created
sale-1001 created

$ sqlline -u jdbc:kubernetes:
sqlline> SELECT DISTINCT c.LAST_NAME FROM KUBERNETES.CUSTOMERS c, KUBERNETES.SALES s WHERE
s.CUSTOMER_ID = c.CUSTOMER_ID;

+-------------+
|  LAST_NAME  |
+-------------+
|  Wakeman    |
+-------------+
```

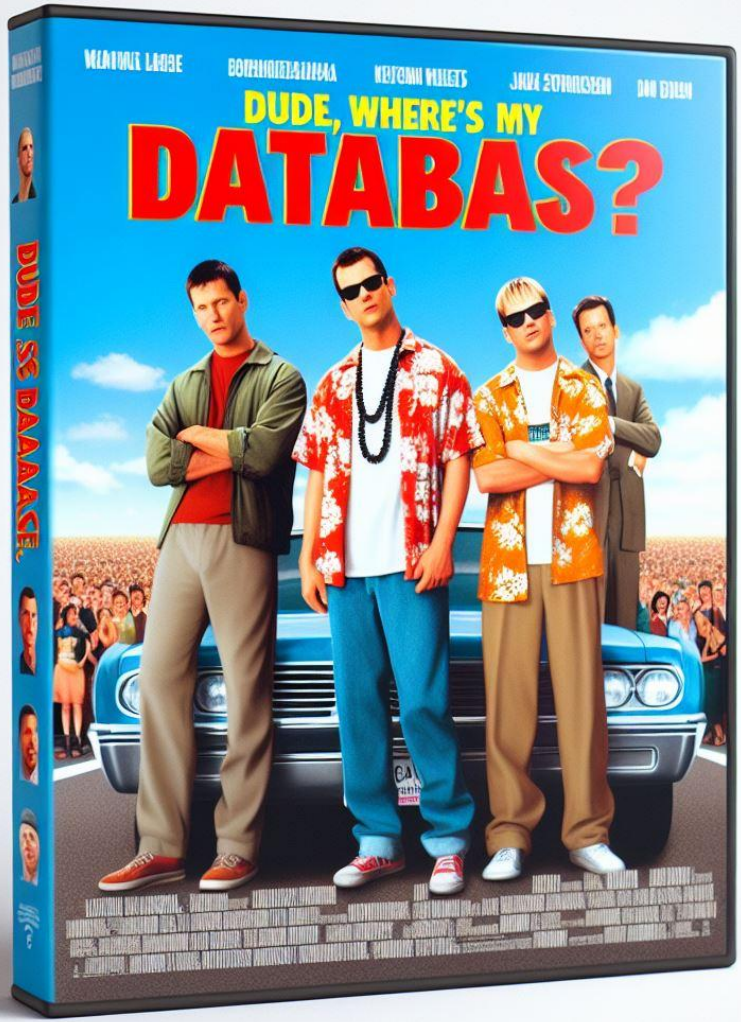
# The view so far

1. hand-write a CRD
2. install it
3. hand-write objects
4. create them
5. start querying!

# But why?

- zero infra
- CRDs as schemas
- in-schema validation (CEL)
- cluster-local data e.g. API keys
- "data-as-code"
- lightweight dev environments

```
$ kind create cluster
$ kubectl apply -f crds.yaml
$ kubectl apply -f dev-env.yaml
$ kind delete cluster
```



YAML
(cool)

DML
(cooler)

# Manipulating K8s with DML+DDL

```
public class KubernetesArrayList<T extends KubernetesObject, U extends KubernetesListObject, V>
    extends ArrayList<V> {
--->%---
  @Override
  public boolean add(V v) {
    api.create(mapper.fromRow(v));
    return super.add(v);
  }
  ...
}

// alternative: keep track of changes and make batch API calls during commit()
// variation: add hooks to the Rows (instead of List<Row>)
```

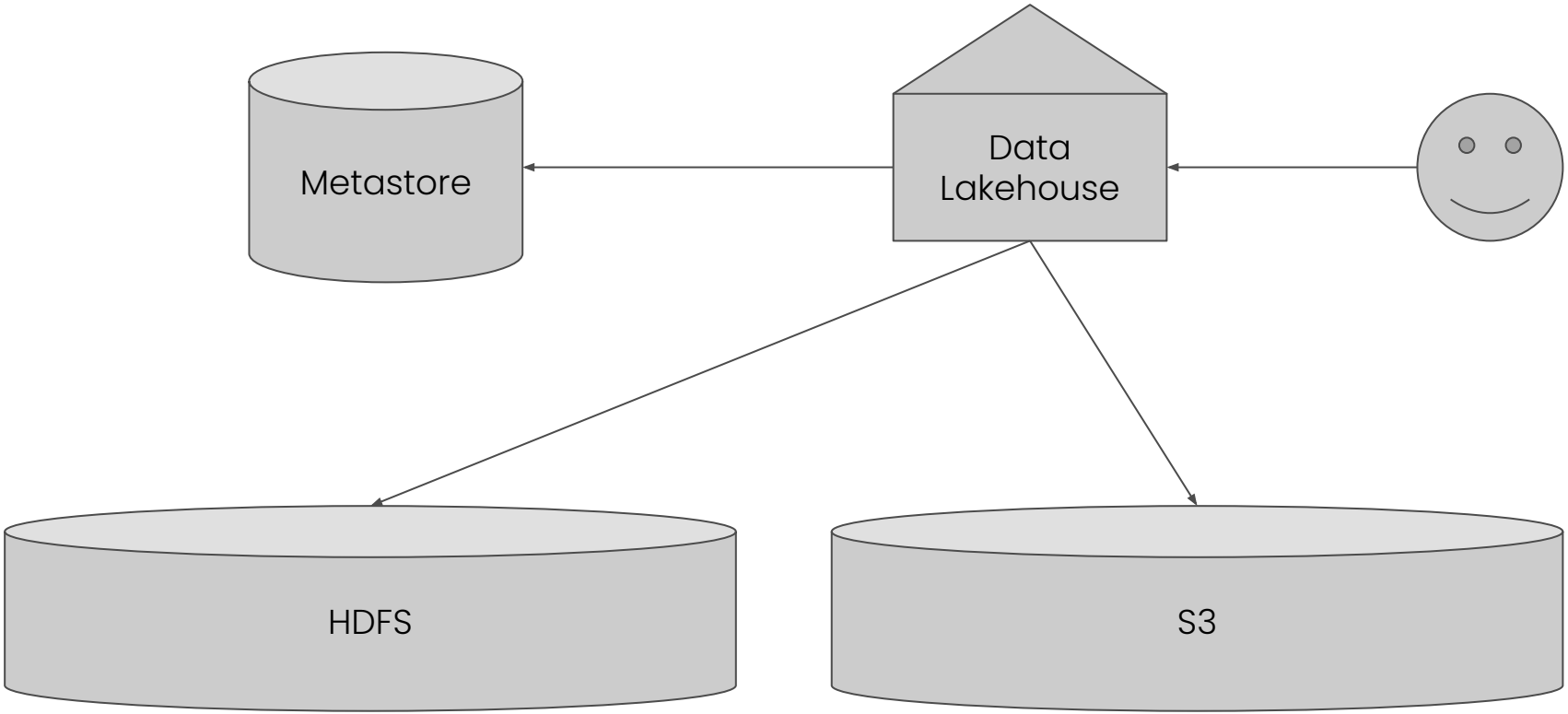
```
$ kubectl apply -f sales.crd.yaml
sales.example.com created

-- same as --

$ sqlline -u jdbc:kubernetes:
sqlline> INSERT INTO KUBERNETES.SALES VALUES('sale-1003', 'customer-123');

$ kubectl get sales
NAME            CUSTOMER_ID
sale-1003               123
```

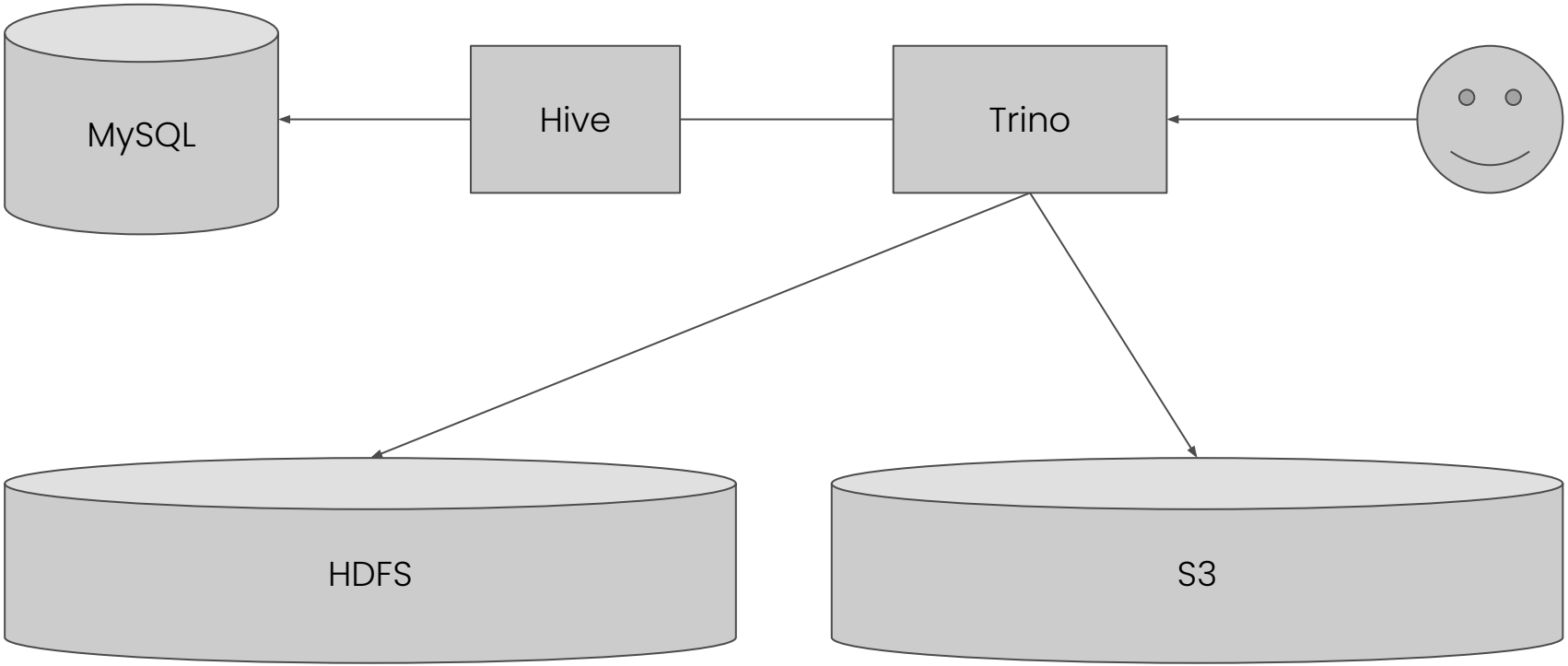
CRDs+DML+SQL
(cool)

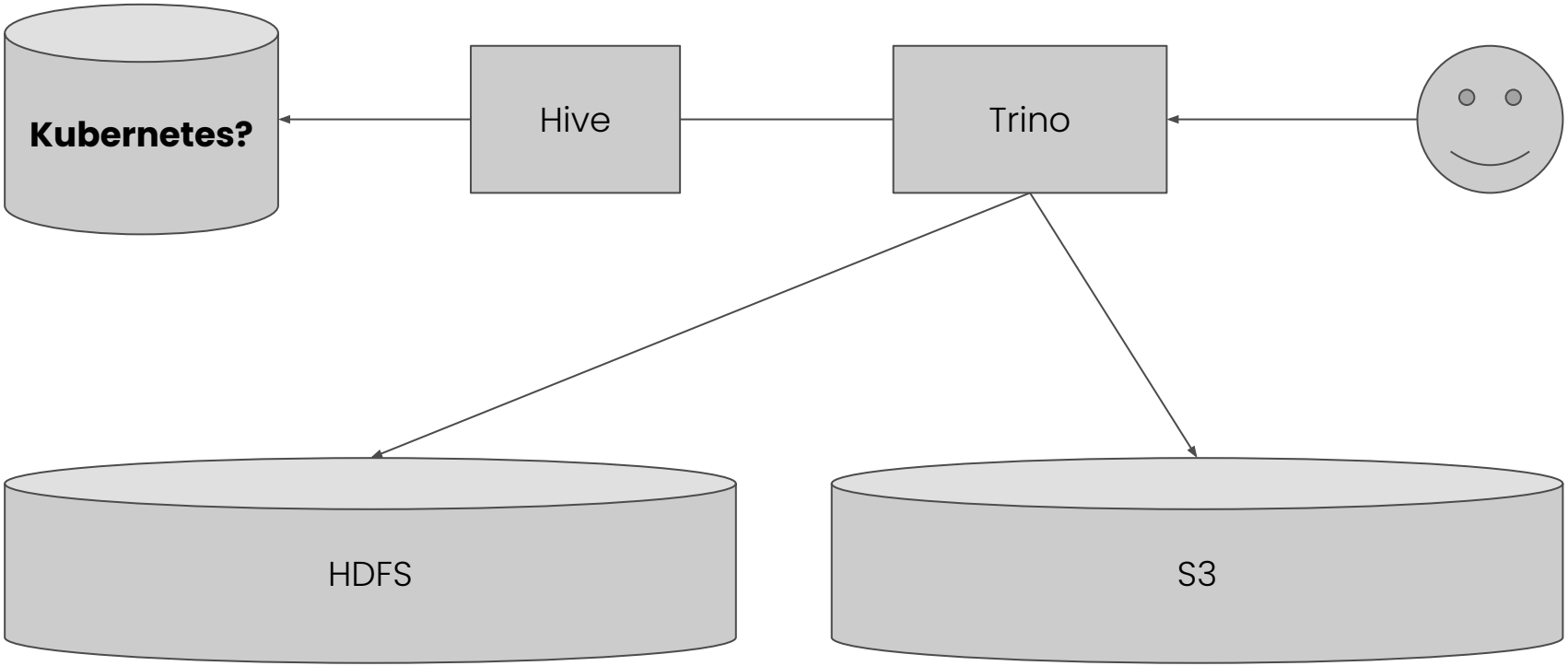
DDL
(ok I guess?)

```
$ sqlline -u jdbc:kubernetes:
sqlline> CREATE TABLE SALES (SALE_ID VARCHAR, CUSTOMER_ID VARCHAR);
sqlline> INSERT INTO SALES VALUES('sale-1003', 'customer-123');

$ kubectl get crds
NAME
sales.example.com

$ kubectl get sales
NAME            CUSTOMER_ID
sale-1003               123

# Watch out for Kubernetes' strict naming requirements.
# DDL might get hairy for complex CRDs.
```

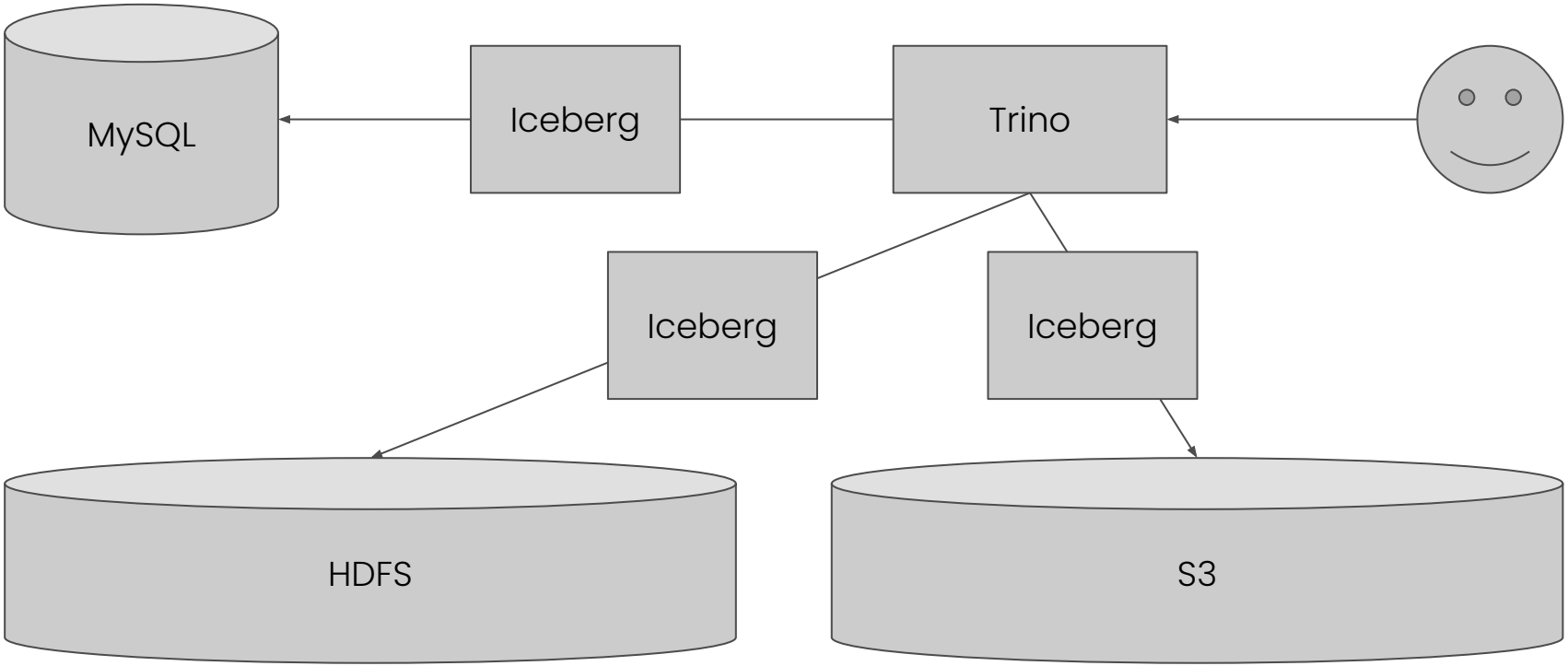
# Using Kubernetes as a Metastore

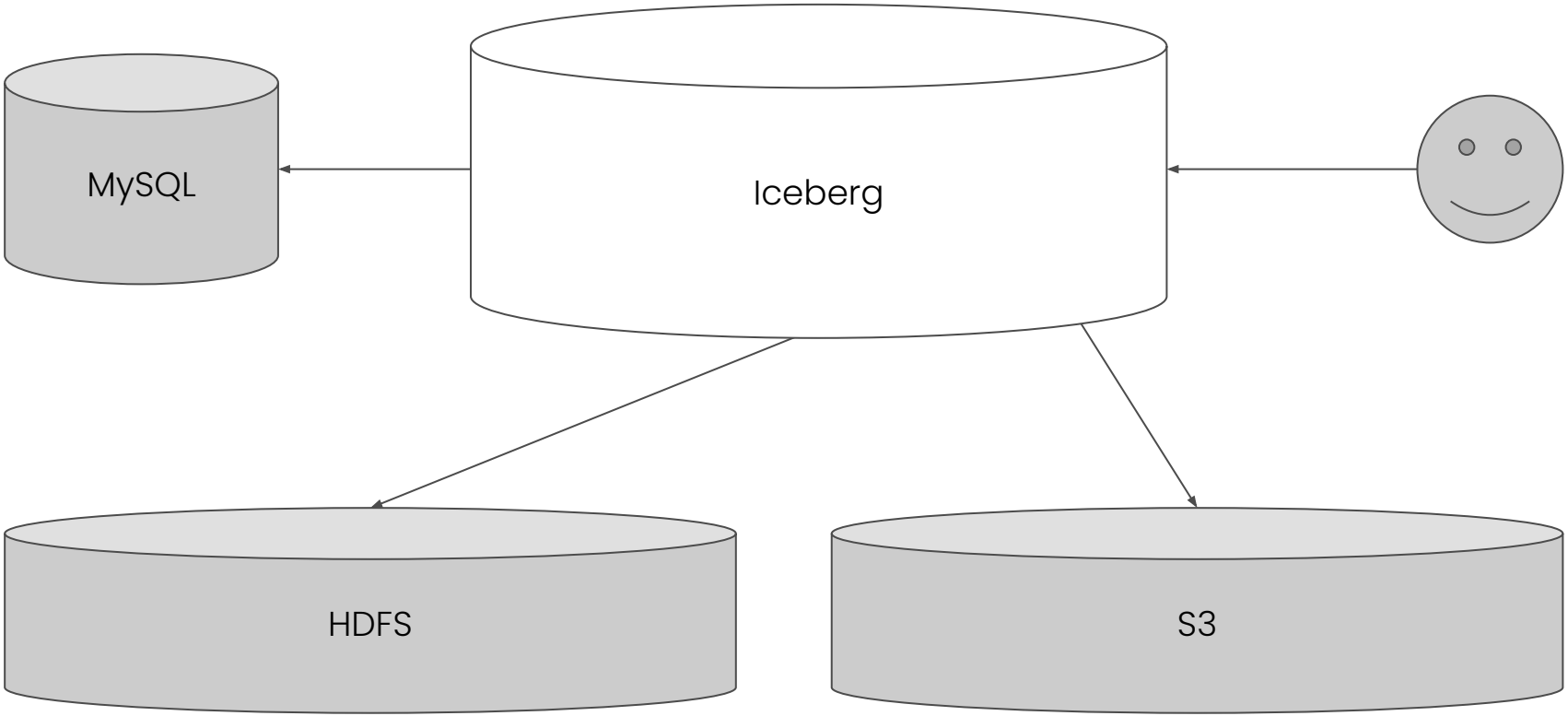

Metastore
Data
Lakehouse
HDFS
S3

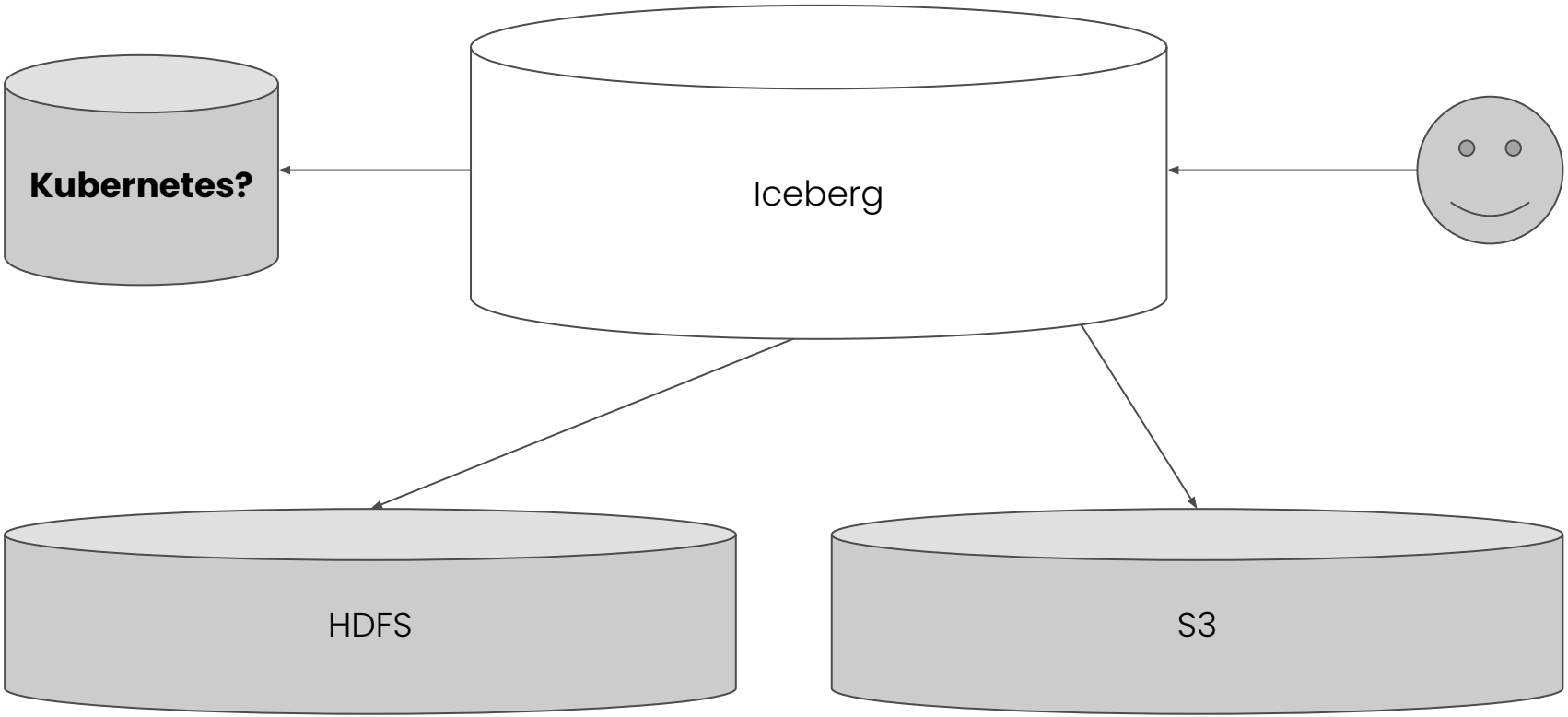

MySQL
Hive
Trino
HDFS
S3


Kubernetes?
Hive
Trino
HDFS
S3

```
$ cat hive-site.xml

--->%---
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:kubernetes:</value>
--->%---
```


MySQL
Iceberg
Trino
Iceberg
Iceberg
HDFS
S3


MySQL
Iceberg
HDFS
S3

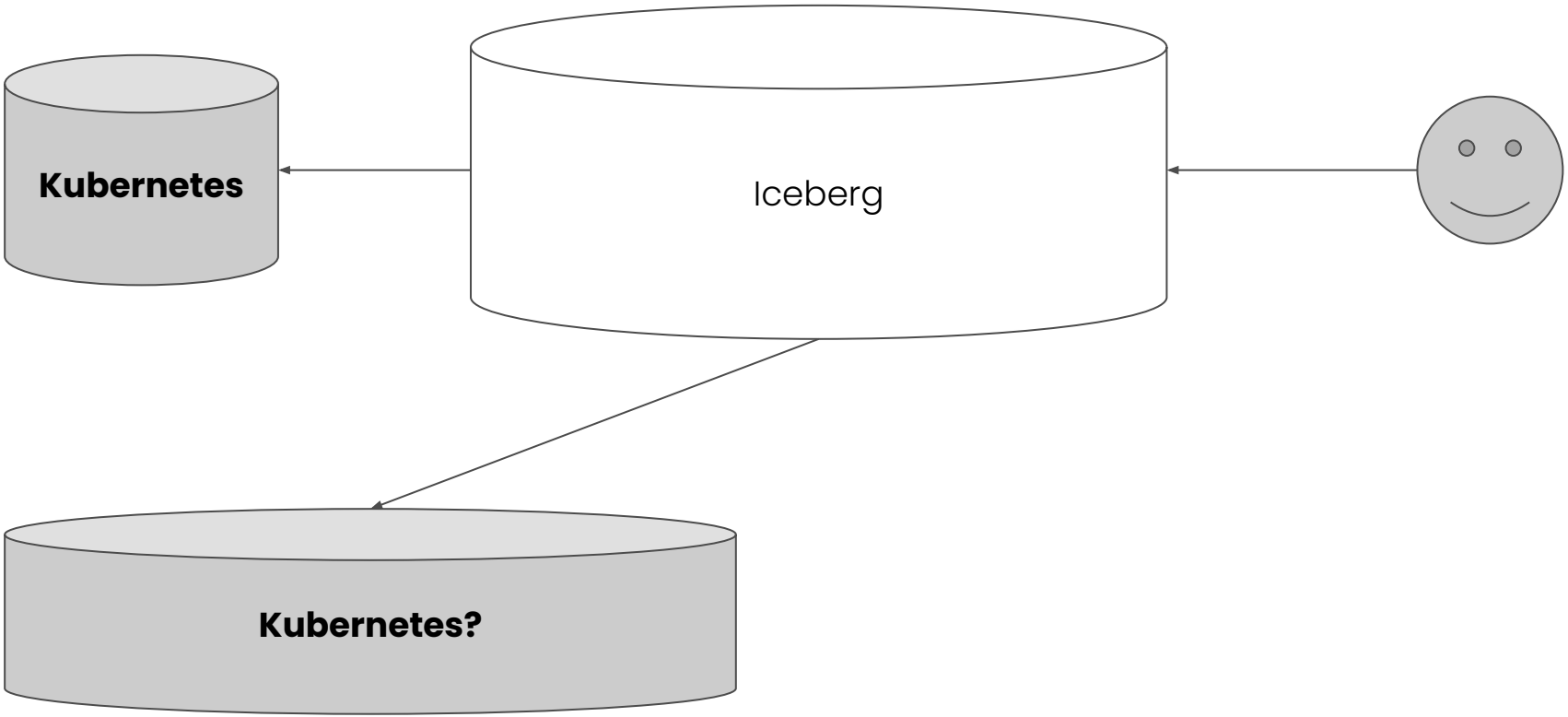

Kubernetes?
Iceberg
HDFS
S3

```
$ spark-sql --jars iceberg-spark-runtime.jar,kubernetes-jdbc-driver.jar \
    --conf spark.sql.catalog.kubernetes=org.apache.iceberg.spark.SparkCatalog \
    --conf spark.sql.catalog.kubernetes.warehouse=s3://my-s3-bucket \
    --conf spark.sql.catalog.kubernetes.type=jdbc \
    --conf spark.sql.catalog.kubernetes.uri=jdbc:kubernetes:
```

# Paradise?

- practically zero infra
- no external metastore
- data only in S3 and K8s

# Using Kubernetes as a File System?


Kubernetes
Iceberg
Kubernetes?

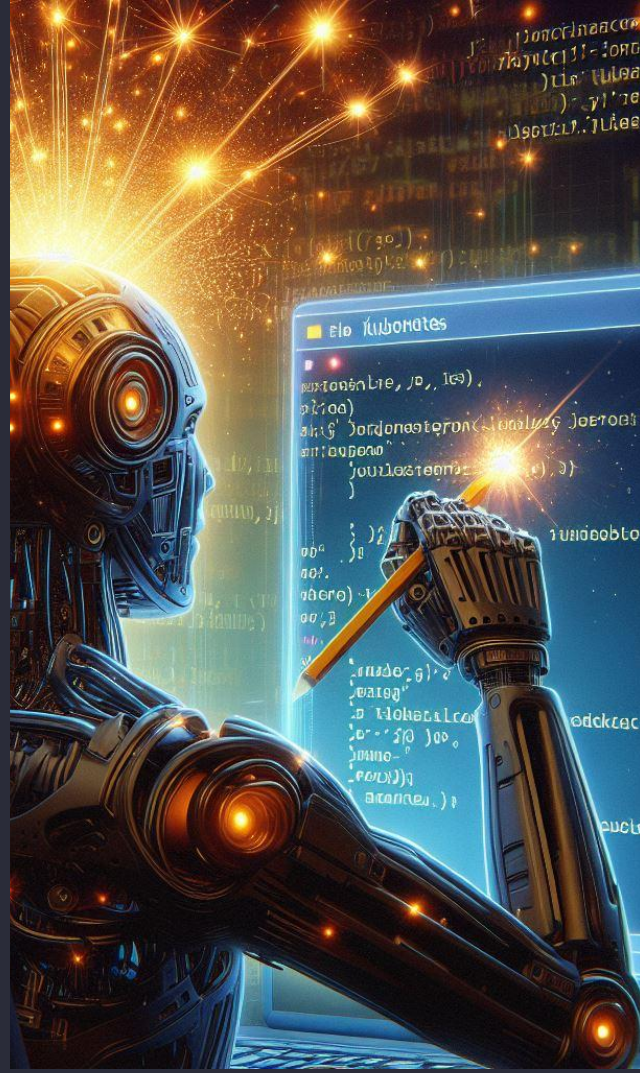
# Some ideas...

- Blob CRD: K8s-as-blob-store?
- Files, Segments, Partitions, Buckets?
- typical limit is **1MB** per object, **2GB** total

# But Why?

- Data lake for integration tests!
- Start small with *just* a hosted K8s cluster.
- Scale-to-zero without losing data.
- Zero pods, zero PVCs.
- Never wait for a database to spin up.
- Managed backups.
- Upgrade to a "real" DB later! (just switch JDBC drivers)

# Check out the Hoptimator project!



## Thanks!

Ryanne Dolan
in/RyanneDolan
@dolanRyanne

All images generated by Microsoft Copilot designer.