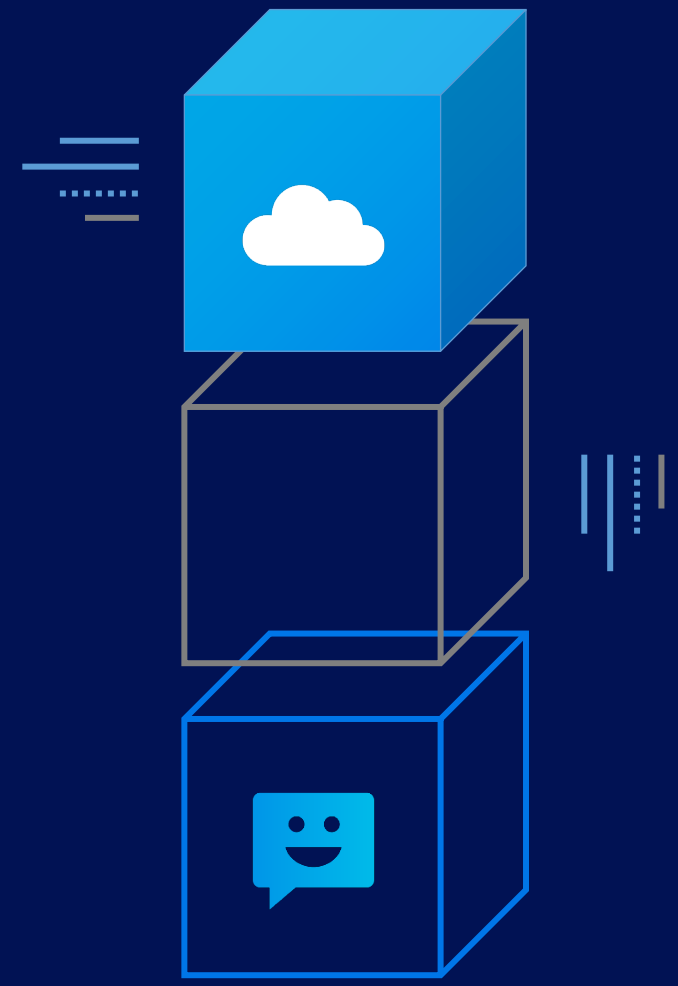
instaclustr

Building and Scaling a Robust Zero-code Streaming Data Pipeline with Open Source Technologies

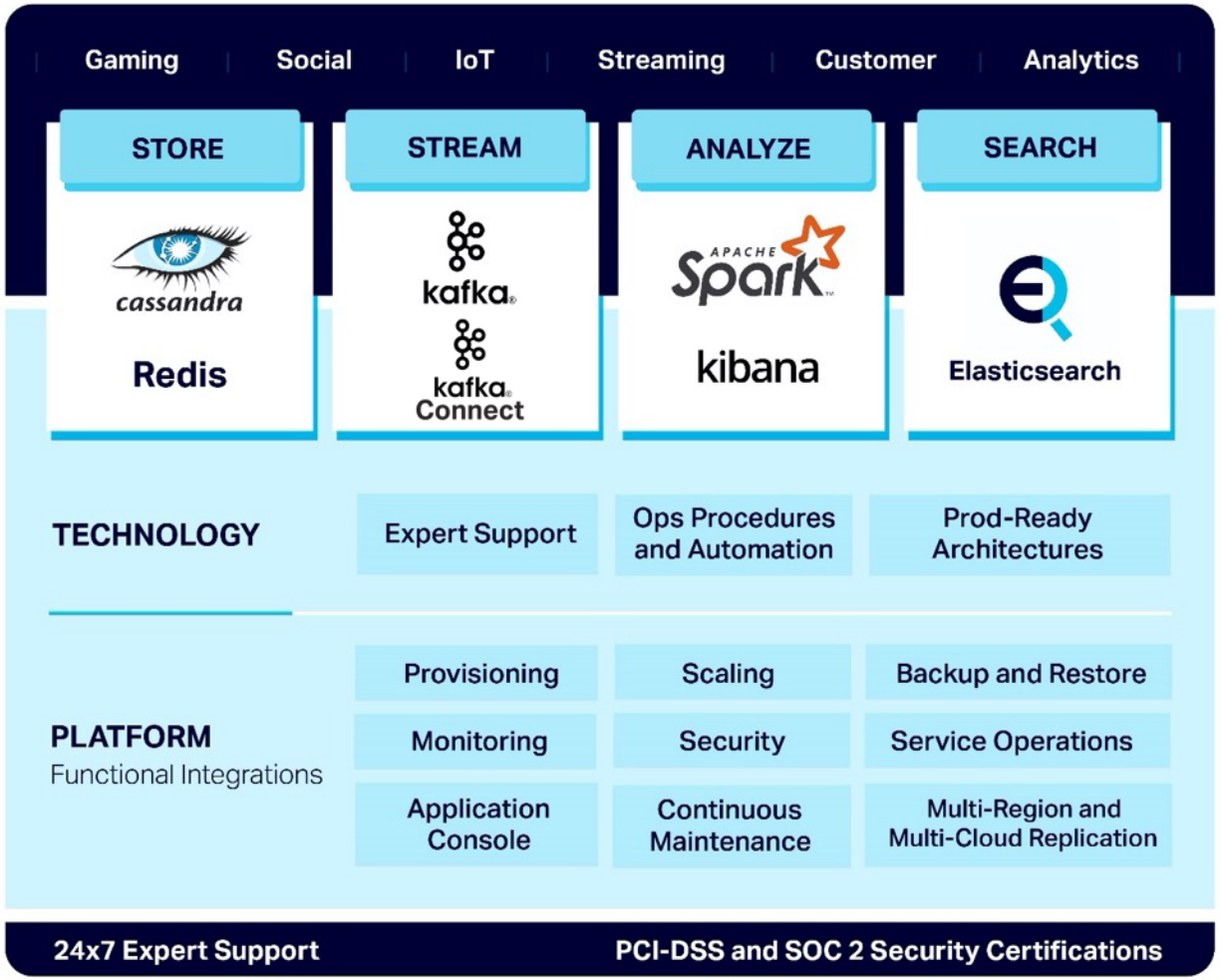


- Apache Kafka, Kafka Connect, Camel Kafka Connectors
- Open Distro for Elasticsearch and Kibana
- Prometheus and Grafana

Paul Brebner
Instaclustr—Technology Evangelist

Instaclustr Managed Platform

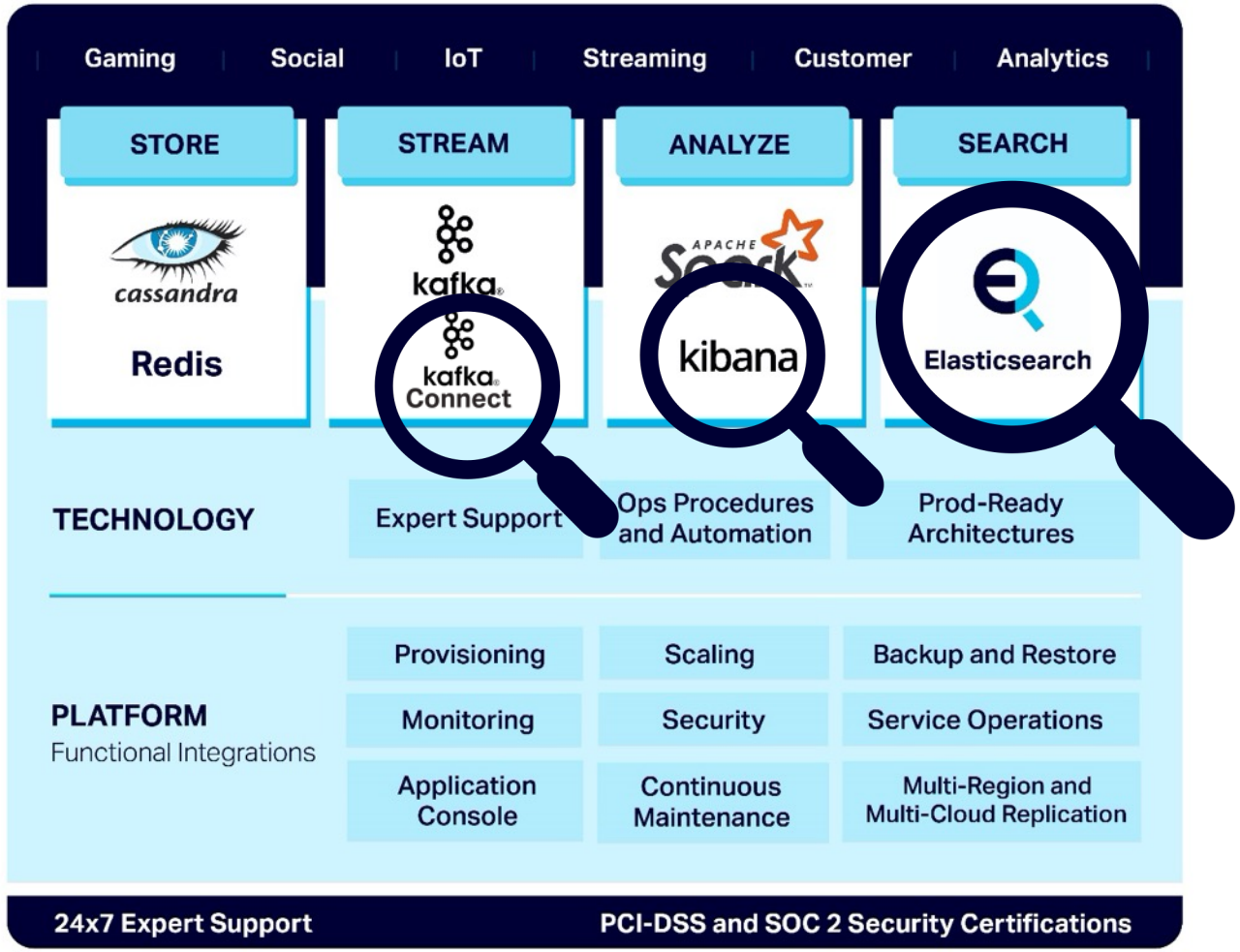
A complete ecosystem to support mission critical open source big data applications.



This talk...

Focuses on three recent additions to our managed platform:

- Kafka Connect
- Elasticsearch
- Kibana



And Pipes – puzzling pipes? Berlin = Beer?

(Source: Paul Brebner)

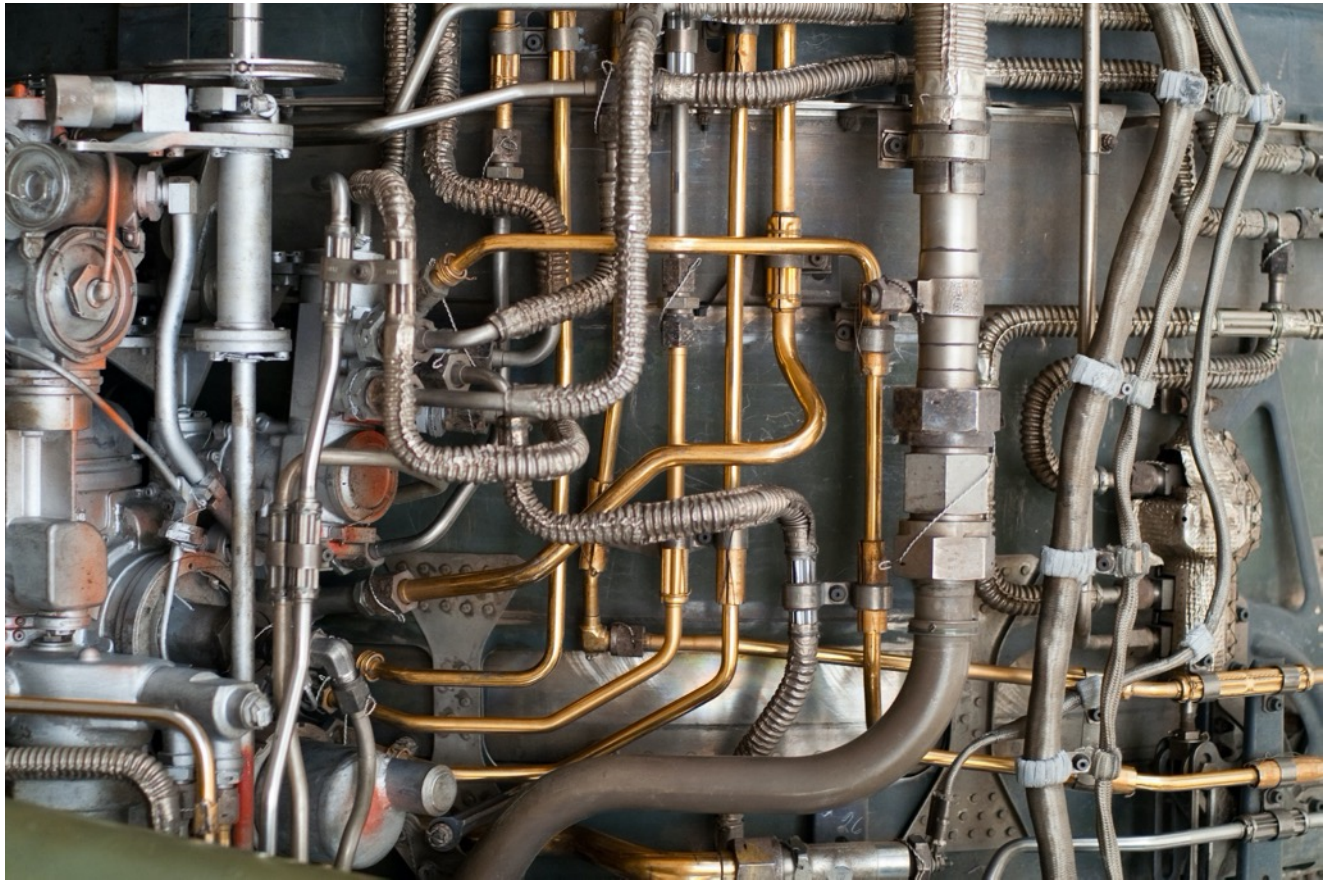
Pipes Can Be Boring

(Source: Shutterstock)

Pipes Can Be Exhilarating!

(Source: Shutterstock)

Pipes (and Integration) Can Be Complicated



(Source: Shutterstock)

Is Building a Robust Scalable Zero-Code Data Pipeline in Open Source a Pipe-Dream?!

instaclustr

(Source: Shutterstock)

Let’s Find Out

(Source: Shutterstock)

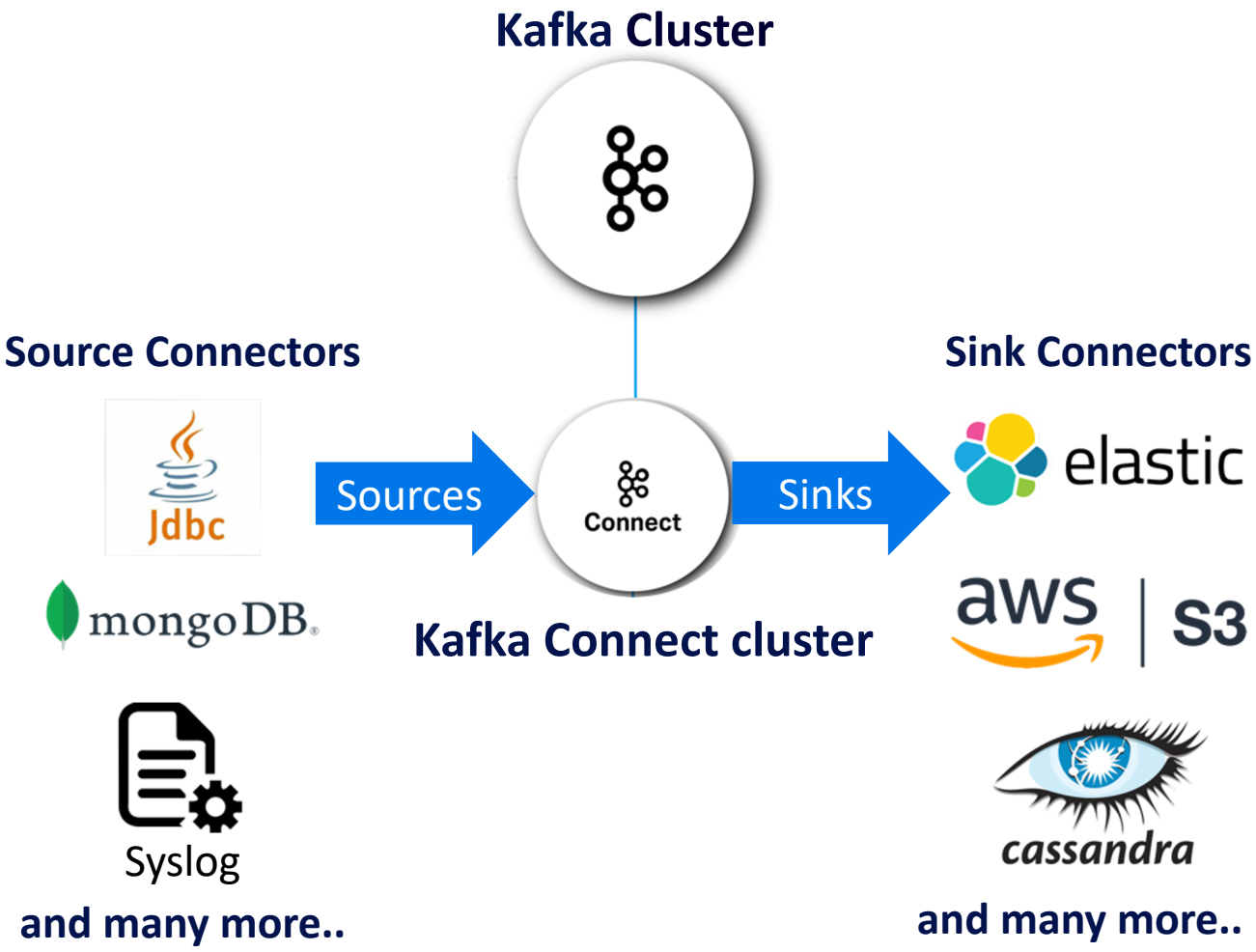
Easy Pipelines With Kafka Connect

What?

- Distributed solution to integrate Kafka with other heterogeneous data sources/stores.
- Connectors (source or sink) handle specifics of particular integrations
  - Source → Kafka
  - Kafka → Sink

Why?

- Zero-code integration
- High availability
- Elastic scaling independent of Kafka

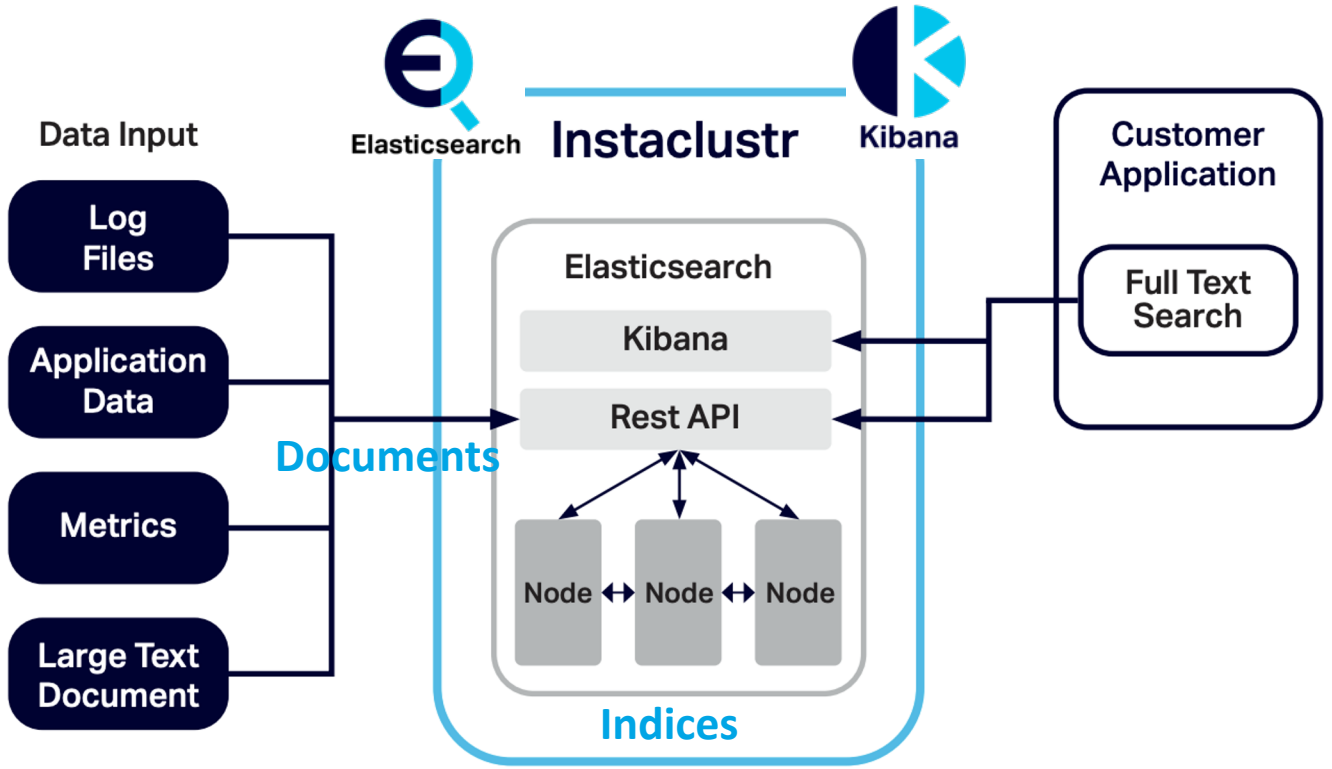


Managed Elasticsearch + Kibana

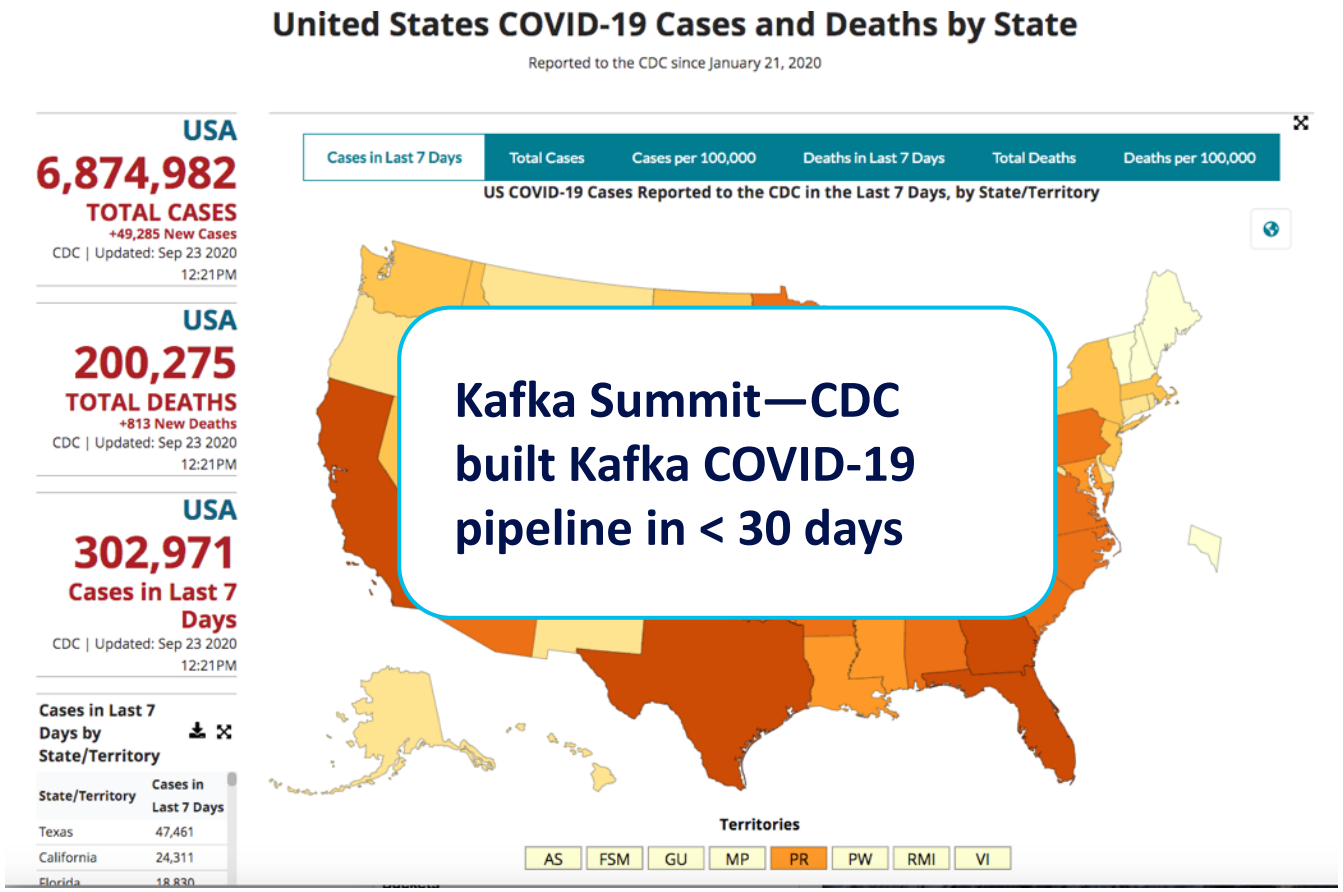
Elasticsearch—scalable search of **indexed documents**

Kibana—visualization

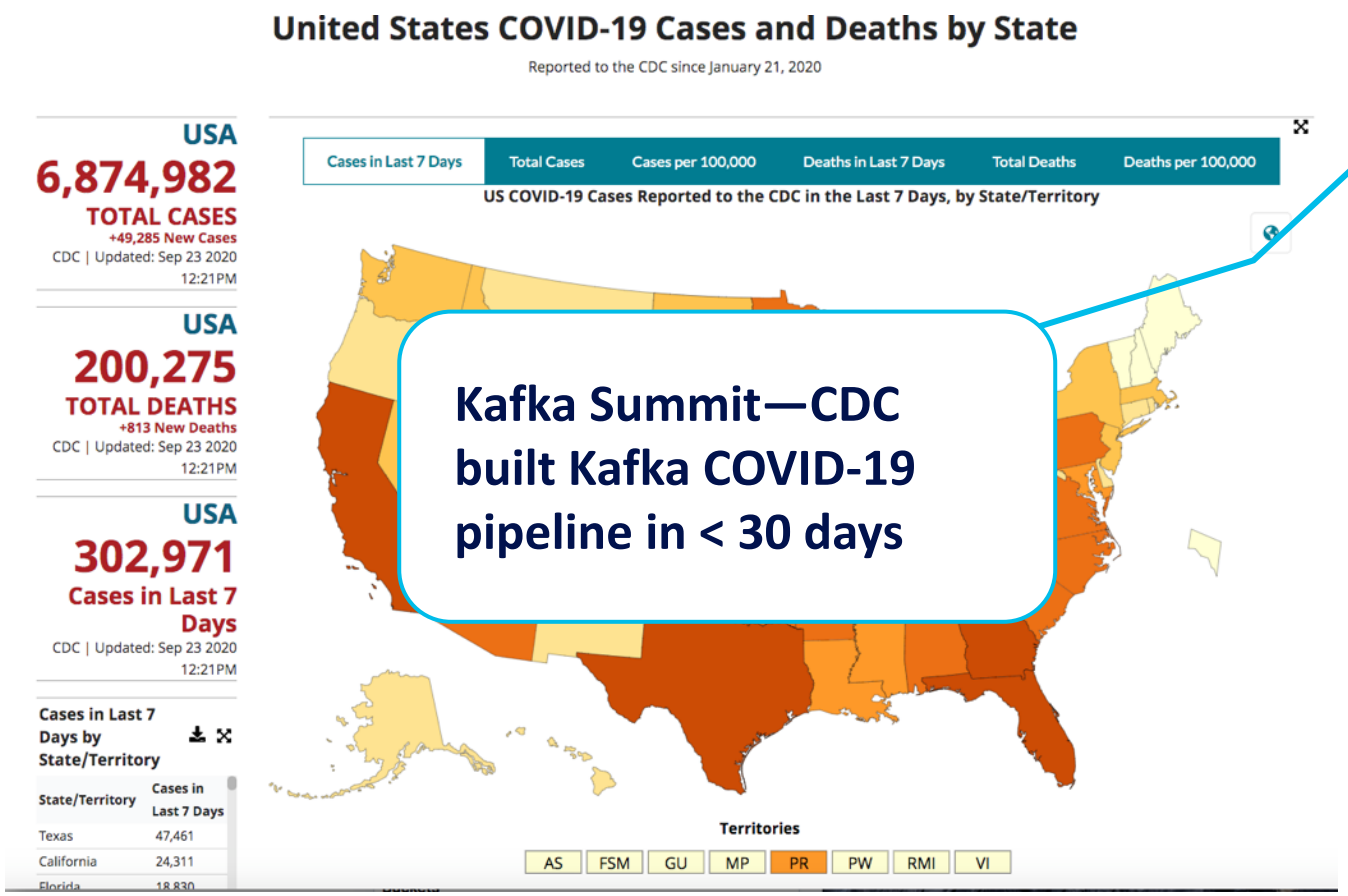
Open Distro for Elasticsearch—100% Apache 2.0 licensed



What's The Story?



What's The Story?

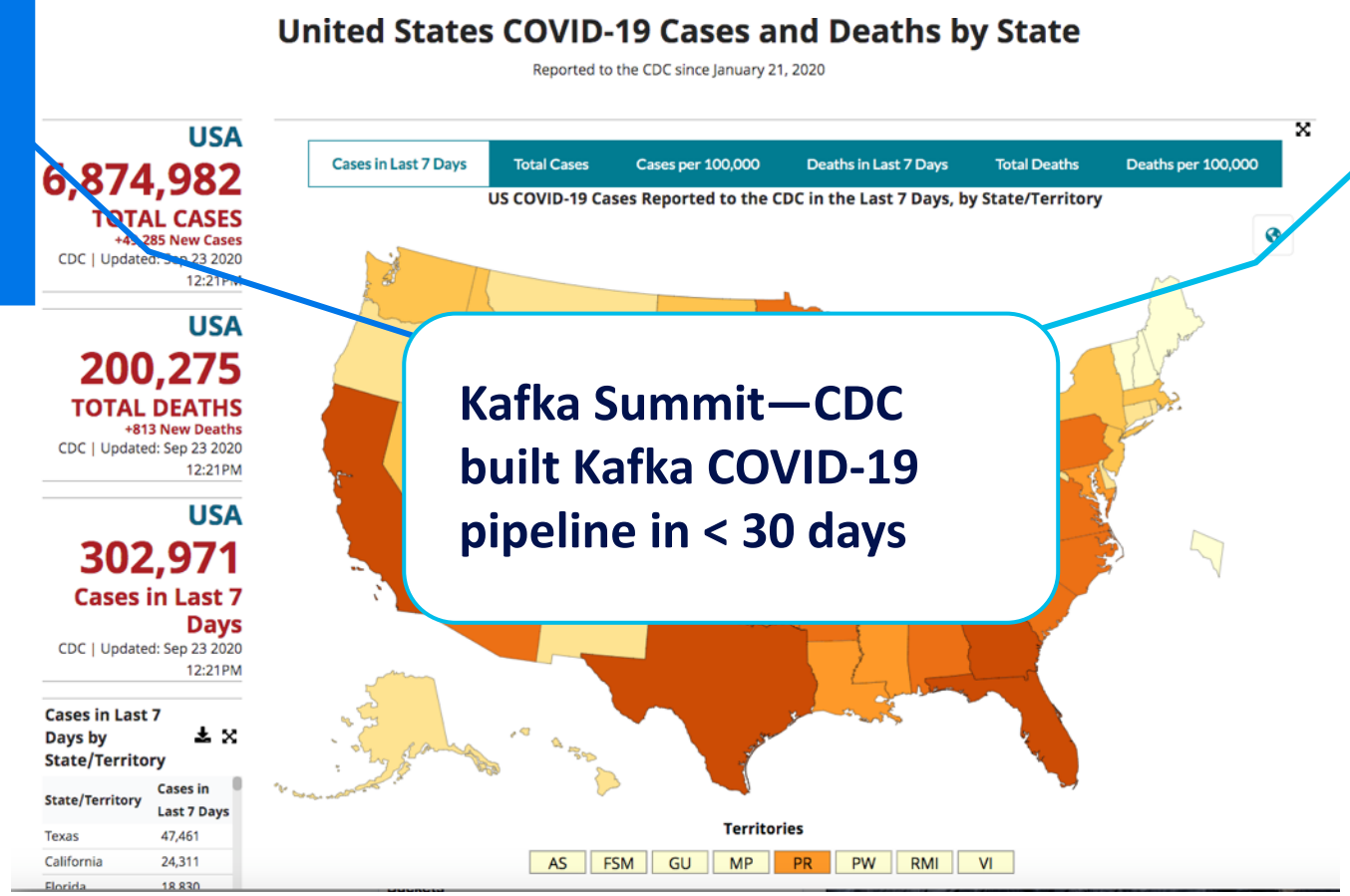


Kafka Summit—CDC built Kafka COVID-19 pipeline in < 30 days

Instaclustr consultants built an integration demo using public climate change data via REST connectors running on docker

What's The Story?

Idea: Use streaming REST public data sources
AND deploy on Instaclustr managed platform



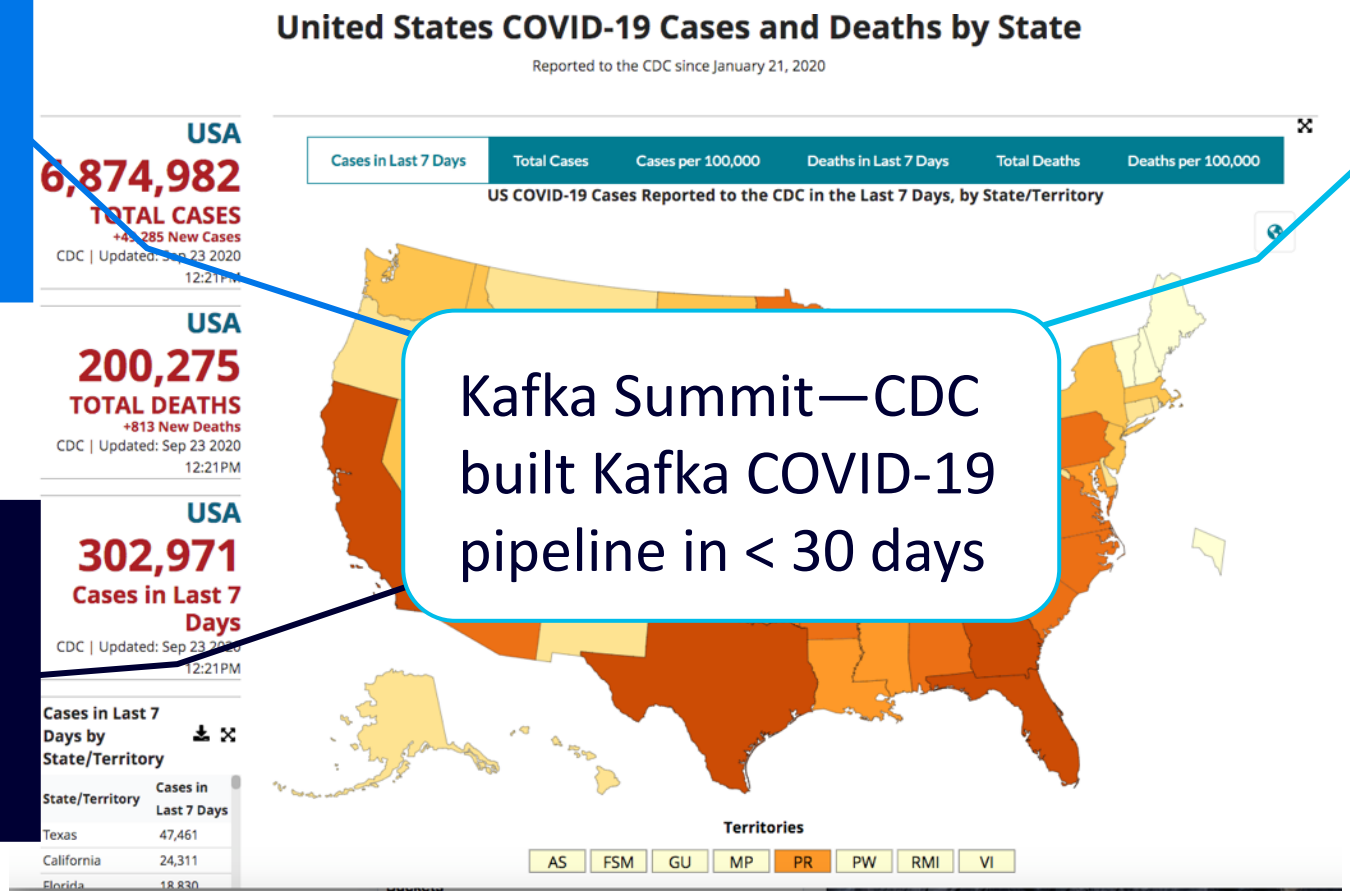
Kafka Summit—CDC built Kafka COVID-19 pipeline in < 30 days

Instaclustr consultants built an integration demo using public climate change data via REST connectors running on docker

What's The Story?

Idea: Use streaming REST public data sources
AND deploy on Instaclustr managed platform

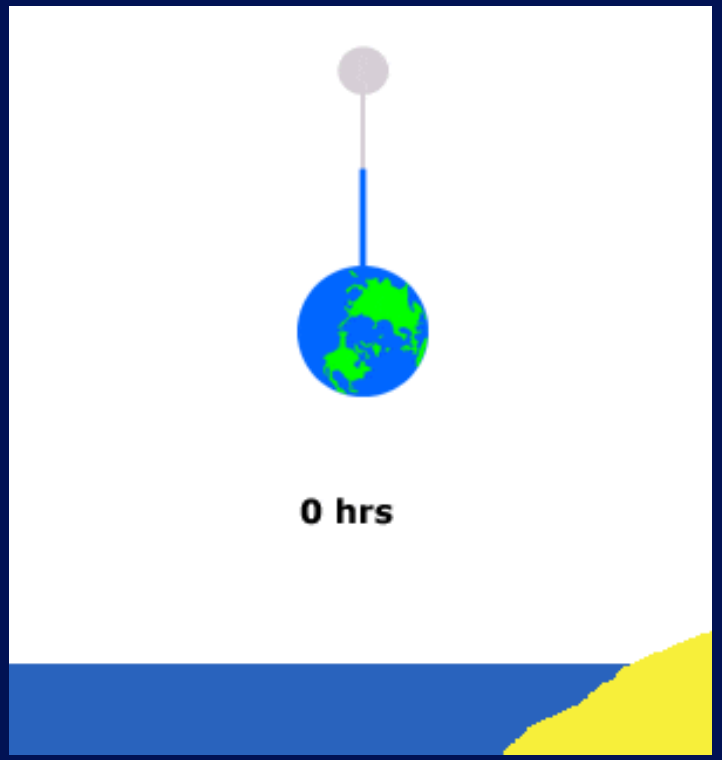
Look for public streaming REST APIs with easy to use JSON data format, complete data, interesting domain, not political or apocalyptic… Impossible?



Kafka Summit—CDC built Kafka COVID-19 pipeline in < 30 days

Instaclustr consultants built an integration demo using public climate change data via REST connectors running on docker

Success! Tides Follow Lunar Day

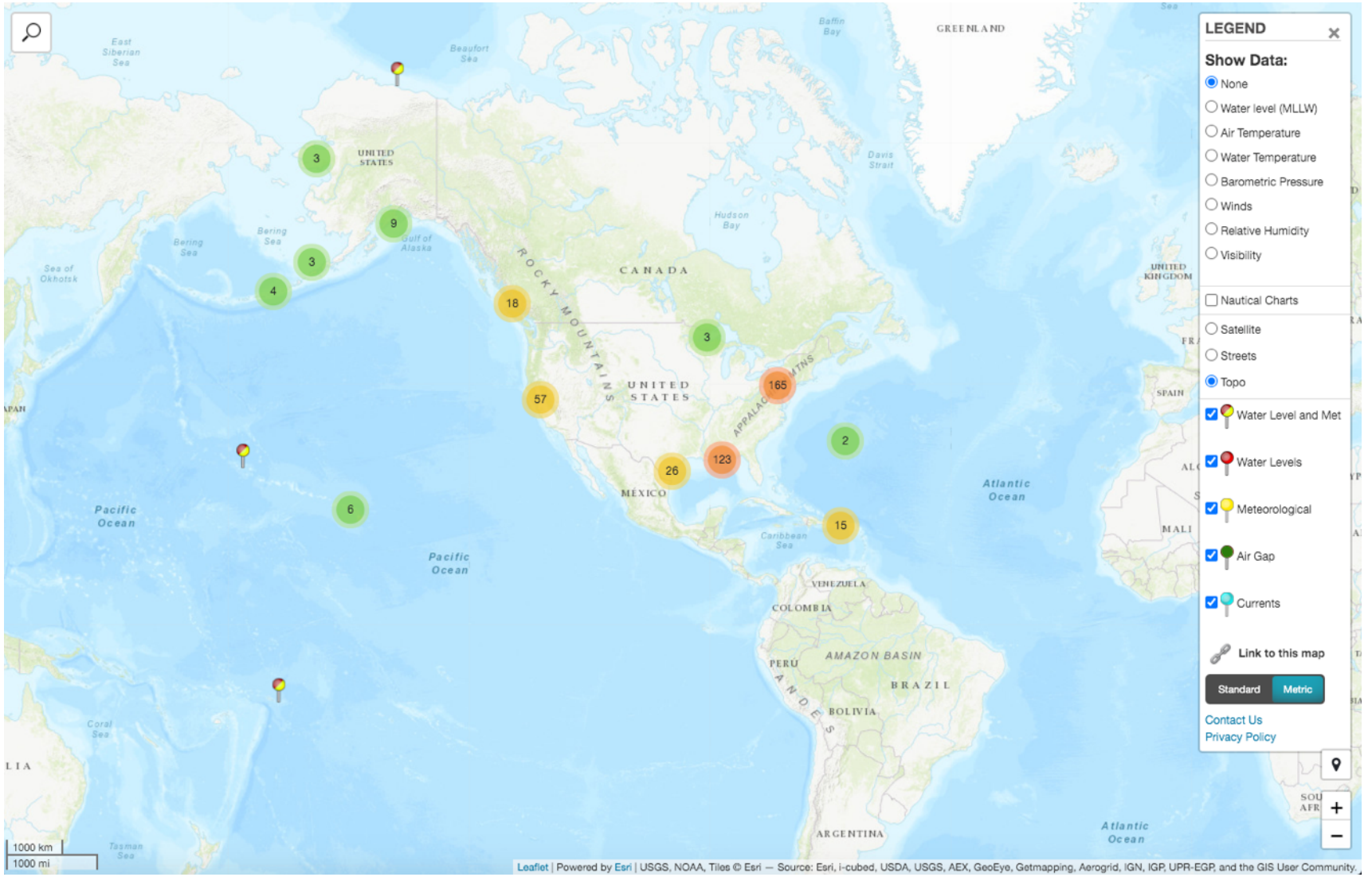


USA Tidal Data

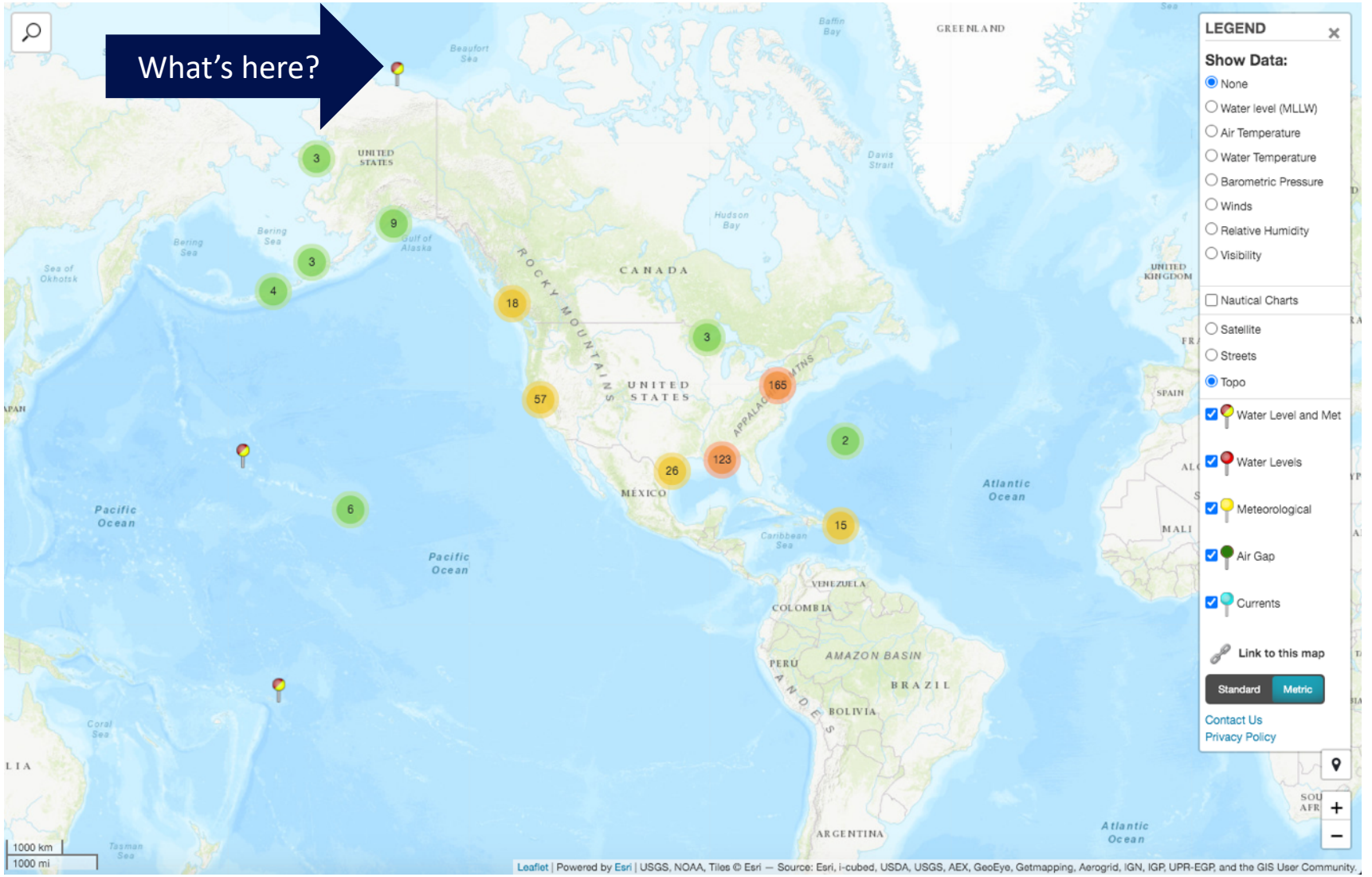
National Oceanic and Atmospheric Administration

https://oceanservice.noaa.gov/

instaclustr



Bonus, NOAA tidal map https://tidesandcurrents.noaa.gov/map/

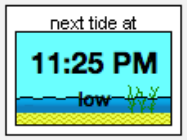


Bonus, NOAA tidal map https://tidesandcurrents.noaa.gov/map/

Prudhoe Bay, AK - Station ID: 9497645

Established:	Jul 17, 1990
Time Meridian:	0° E
Present Installation:	Jul 17, 1990
Date Removed:	N/A
Water Level Max (ref MHHW):	4.10 ft. Aug 11, 2000
Water Level Min (ref MLLW):	-3.43 ft. Mar 04, 2013
Mean Range:	0.51 ft.
Diurnal Range:	0.7 ft.
Latitude	70° 24.7' N
Longitude	148° 31.9' W
NOAA Chart#:	16061
Met Site Elevation:	N/A

Today's Tides (LST/LDT)



5:56 AM	high	0.9 ft.
12:31 PM	low	0.5 ft.
4:40 PM	high	0.6 ft.
11:25 PM	**low**	**0.3** ft.

Prudhoe Bay, AK

5 more station photos available, click to view.

Sensor Information

Sensor	Sensor ID	DCP#	Sensor Height	Status
Wind	C1	1	112.5 ft. above site Elevation	✔ Enabled
Air Temperature	D1	1	13.1 ft. above site Elevation	✔ Enabled
Water Temperature	E1	1	5 ft. below MLLW	✔ Enabled
Barometric Pressure	F1	1	N/A	✔ Enabled
Tsunami WL	U1	1	N/A	✔ Enabled
Microwave WL	Y1	1	N/A	✔ Enabled

For questions about disabled sensors, please contact CO-OPS User Services.

Observations

Turn off refresh:

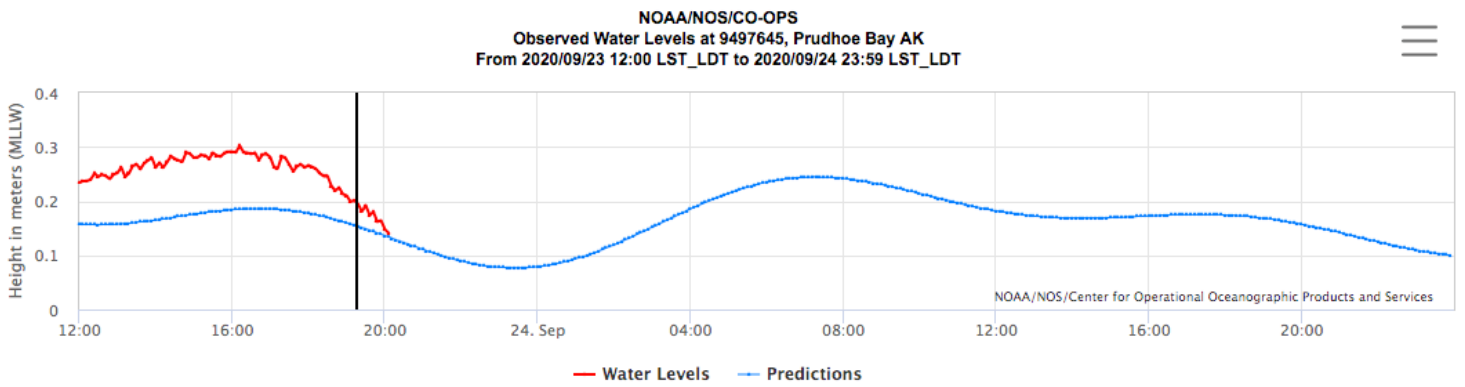
Standard | Metric

Water Levels

0.141 m

MLLW

as of 09/23/2020 20:06 LST/LDT

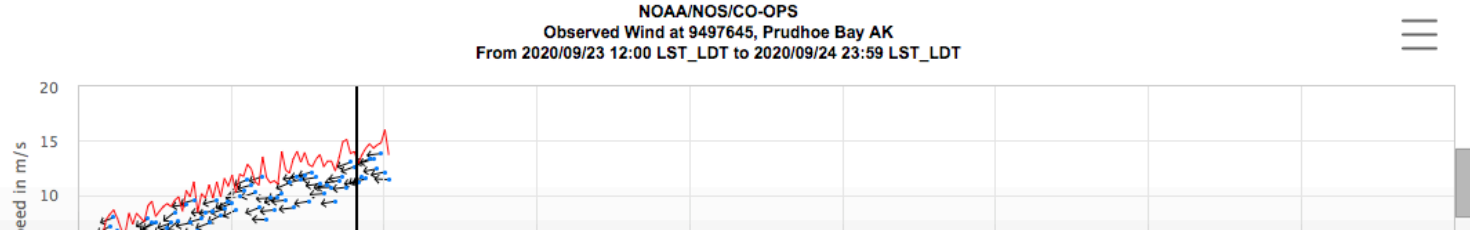


Winds

11.4m/s

from E

as of 09/23/2020 20:06 LST/LDT



API description https://api.tidesandcurrents.noaa.gov/api/prod/

JUMP TO:

- Station ID
- Date & Time
- Data Products
- Datum
- Units
- Time Zone
- Format
- Interval
- Bin
- Application
- Retrieval Time
- Sample URL
- Error Message
- Contact Us

HELP LINKS:

- Response Help Page

CO-OPS API For Data Retrieval

The CO-OPS API for data retrieval can be used to retrieve observations and predictions from CO-OPS stations.

Station ID

A 7 character station ID, or a currents station ID. Specify the station ID with the "station=" parameter.
Example: station=9414290
Station listings for various products can be viewed at https://tidesandcurrents.noaa.gov or viewed on a map at Tides & Currents Station Map

Date & Time

The API understands several parameters related to date ranges.
All dates can be formatted as follows:
yyyyMMdd, yyyyMMdd HH:mm, MM/dd/yyyy, or MM/dd/yyyy HH:mm

One the 4 following sets of parameters can be specified in a request:

Parameter Name (s)	Description
begin_date and end_date	Specify the date/time range of retrieval
date	Valid options for the date parameters are: latest (last data point available within the last 18 min), today, or recent (last 72 hours)
begin_date and a range	Specify a begin date and a number of hours to retrieve data starting from that date
end_date and a range	Specify an end date and a number of hours to retrieve data ending at that date
range	Specify a number of hours to go back from now and retrieve data for that date range

January 1st, 2012 through January 2nd, 2012
begin_date=20120101&end_date=20120102
48 hours beginning on April 15, 2012
begin_date=20120415&range=48
48 hours ending on March 17, 2012
end_date=20120307&range=48
Today's data
date=today
The last 3 days of data
date=recent
The last data point available within the last 18 min
date=latest
The last 24 hours from now
range=24

REST Example

Specify station ID, data type and datum (I used water level, mean sea level), latest data point, JSON

Call

https://api.tidesandcurrents.noaa.gov/api/prod/datagetter?date=latest&station=8724580&product=water_level&datum=msl&units=metric&time_zone=gmt&application=instaclustr&format=json

Returns


```
{"metadata": {
   "id":"8724580",
   "name":"Key West",
   "lat":"24.5508”,
   "lon":"-81.8081"},
 "data":[{
   "t":"2020-09-24 04:18",
   "v":"0.597",
   "s":"0.005", "f":"1,0,0,0", "q":"p"}]}
```


instaclustr

Let’s Start the Pipeline Using This Rest API for Data Sources…

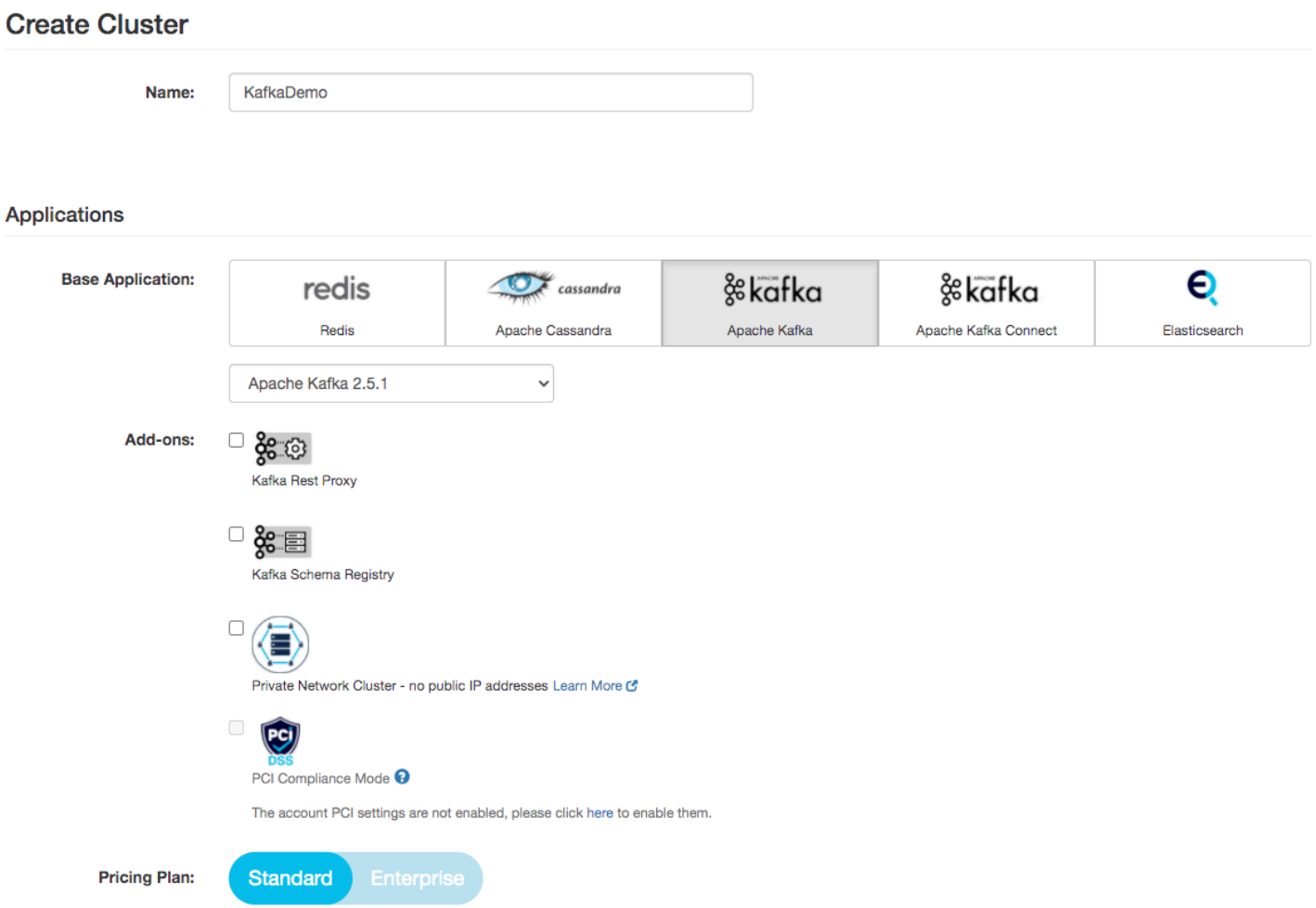
REST call

JSON result

What Else Do We Need?

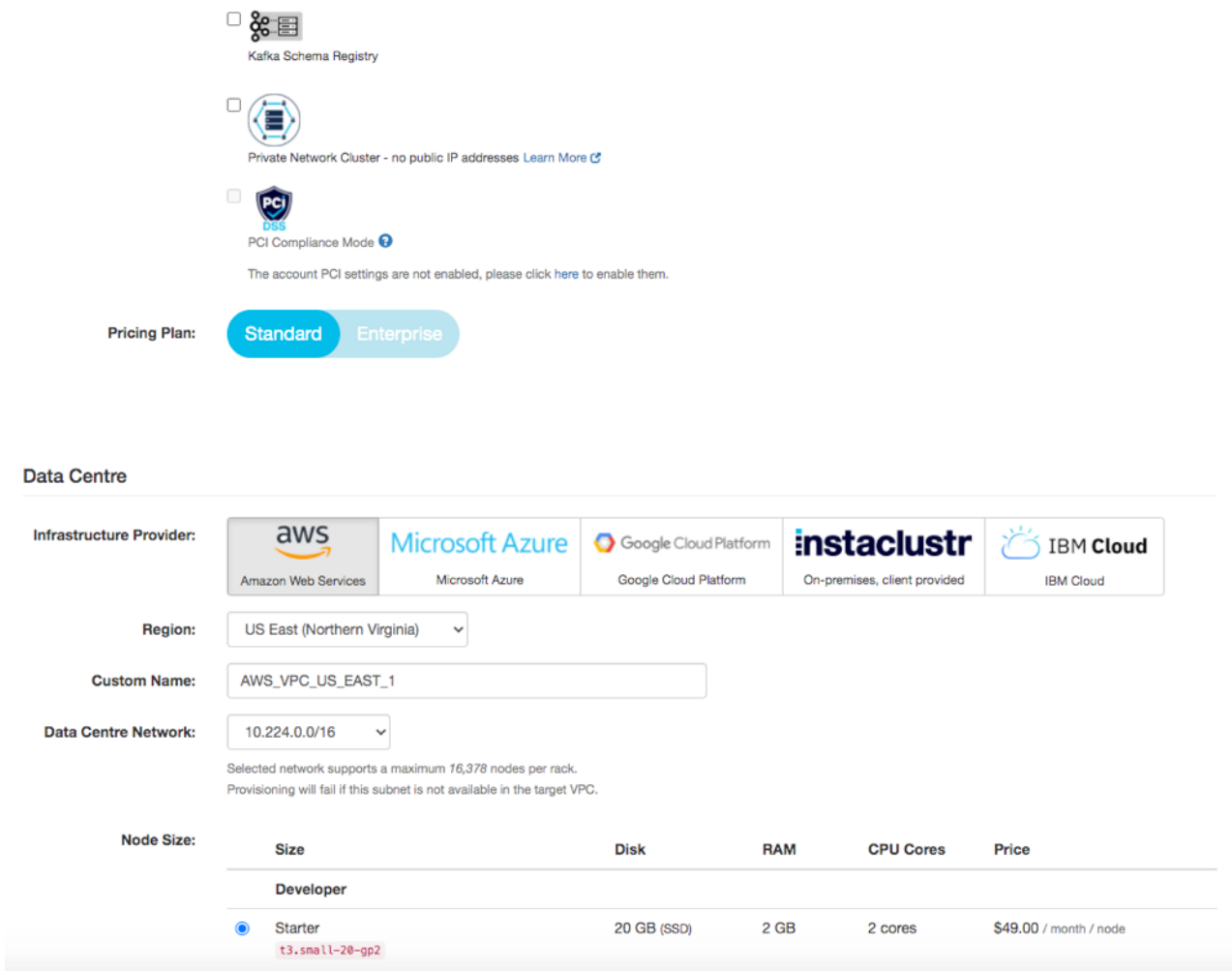
The Instaclustr Console

Provision Kafka and Kafka Connect clusters



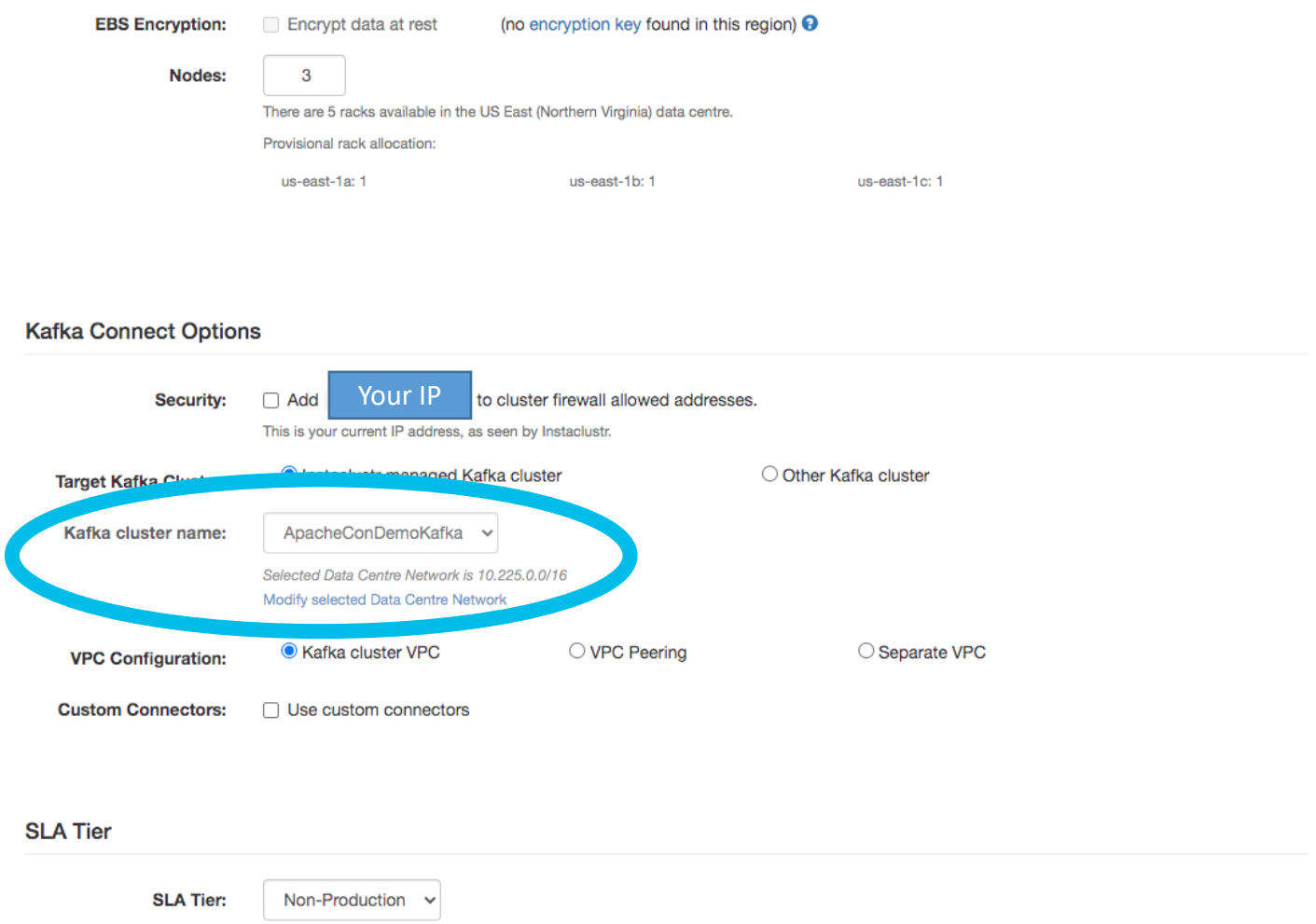
What Else Do We Need?

Select cloud provider, region, instance size and number, security etc.

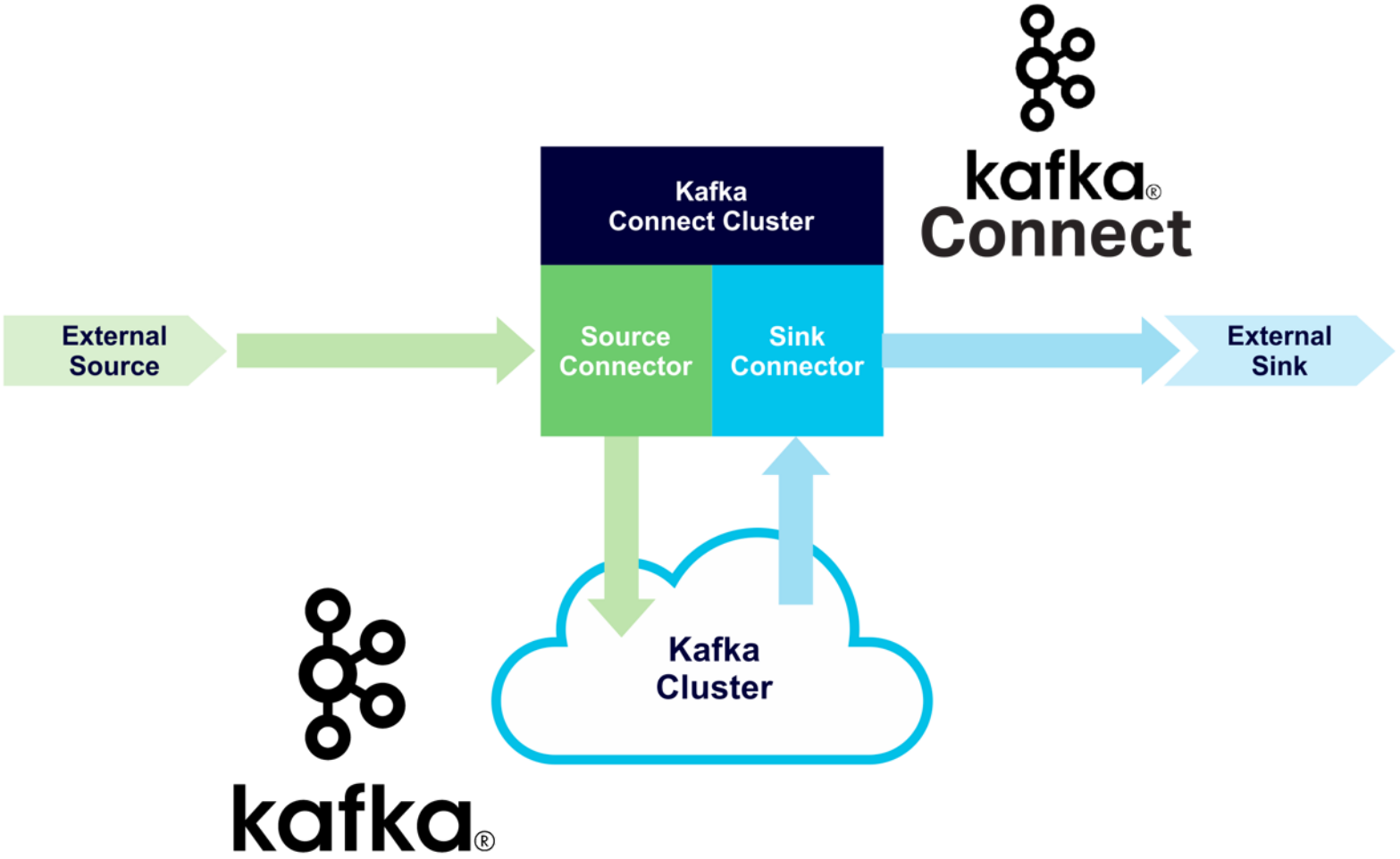


What Else Do We Need?

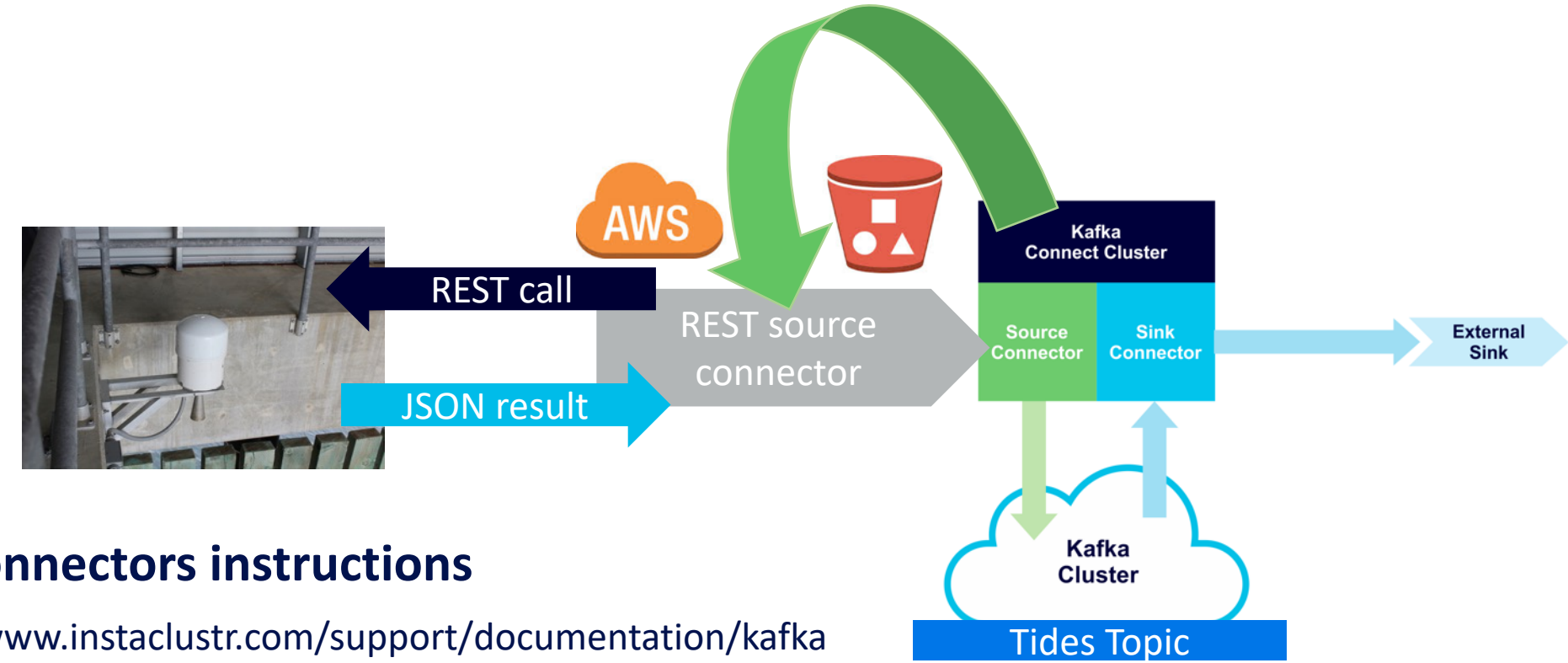
Tell Kafka connect cluster which Kafka cluster to use, then provision



Now we have a Kafka and Kafka Connect clusters



Next, find a REST connector, deploy to S3 bucket, tell connect cluster which bucket, configure connector and run



BYO connectors instructions

https://www.instaclustr.com/support/documentation/kafka-connect/accessing-and-using-kafka-connect/updating-custom-connectors/

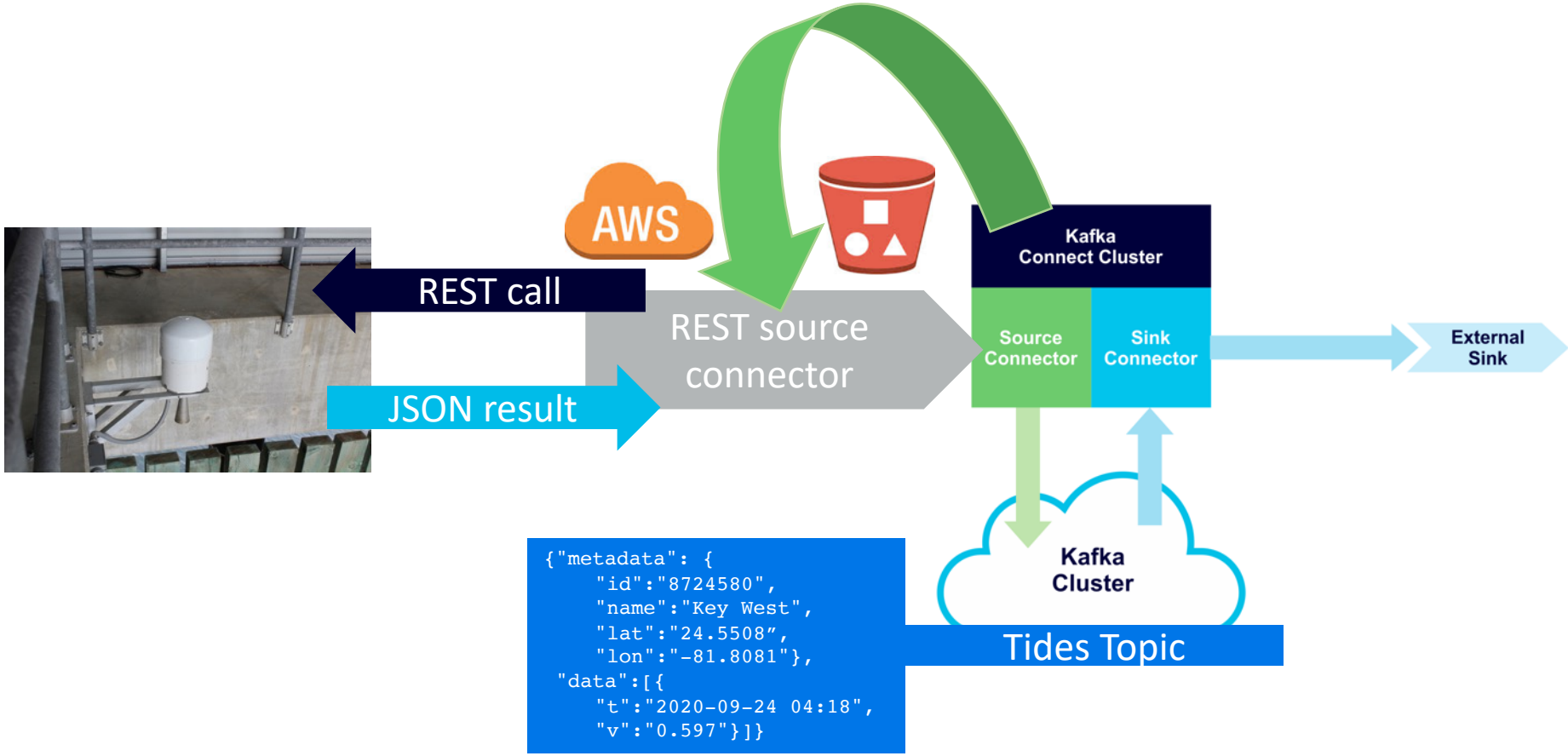
(Automatically created)

REST source connector configuration including connector name, class, URL, topic

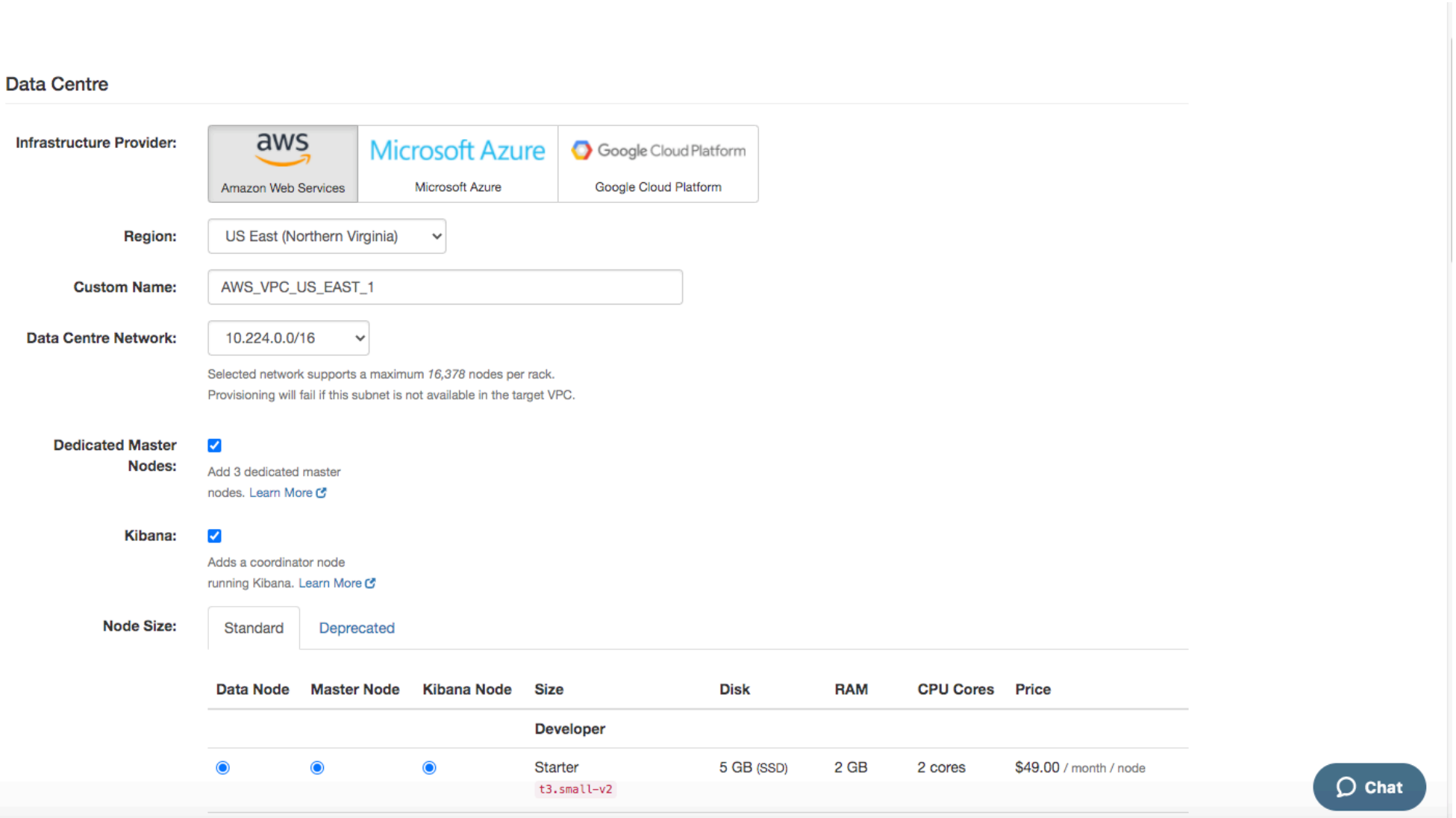
```
curl https://connectorClusterIP:8083/connectors -k -u name:password -X POST -H 'Content-Type: application/json' -d '
{
   "name": "source_rest_tide_1",
   "config": {
    "key.converter":"org.apache.kafka.connect.storage.StringConverter",
    "value.converter":"org.apache.kafka.connect.storage.StringConverter",
    "connector.class": "com.tm.kafka.connect.rest.RestSourceConnector",
    "tasks.max": "1",
    "rest.source.poll.interval.ms": "600000",
    "rest.source.method": "GET",
    "rest.source.url":
"https://api.tidesandcurrents.noaa.gov/api/prod/datagetter?date=latest&station=8454000&product=water_level&datum=
msl&units=metric&time_zone=gmt&application=instaclustr&format=json",
    "rest.source.headers": "Content-Type:application/json,Accept:application/json",
    "rest.source.topic.selector": "com.tm.kafka.connect.rest.selector.SimpleTopicSelector",
    "rest.source.destination.topics": "tides-topic"
   }
}'
```

Polls every 10 minutes, writes result to Kafka topic, picked 5 sensors to use, so 5 connector instances running.

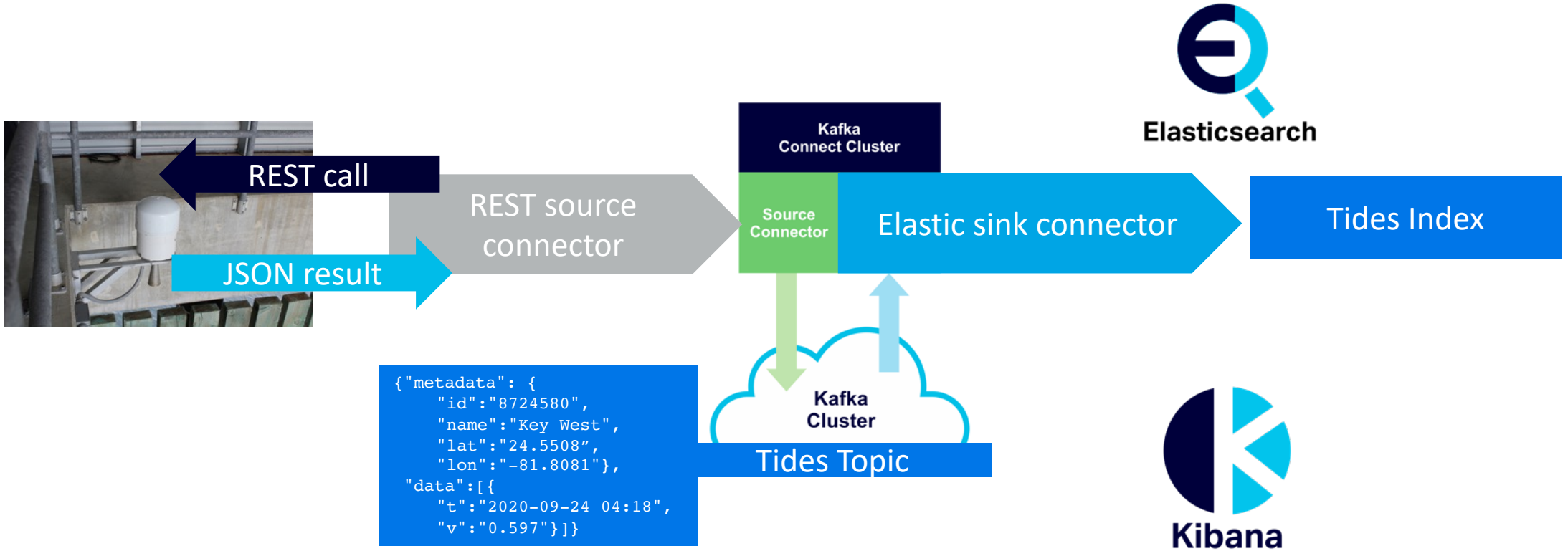
Now have tidal data coming into the tides topic, what next?



Next: Provision Elasticsearch+Kibana clusters



And configure the default (included) Elasticsearch sink connector to send data to Elasticsearch



Configure sink connector name, class, index and topic.

The index is created with default mappings if it doesn't already exist.

```
curl https://connectorClusterIP:8083/connectors -k -u name:password -X POST -H 'Content-Type: application/json' -d '
{
  "name" : "elastic-sink-tides",
  "config" :
  {
    "connector.class" : "com.datamountaineer.streamreactor.connect.elastic7.ElasticSinkConnector",
    "tasks.max" : 3,
    "topics" : "tides",
    "connect.elastic.hosts" : ”ip",
    "connect.elastic.port" : 9201,
    "connect.elastic.kcql" : "INSERT INTO tides-index SELECT * FROM tides-topic",
    "connect.elastic.use.http.username" : ”elasticName",
    "connect.elastic.use.http.password" : ”elasticPassword"
  }
}'
```

Great! It's All Working!? Sort Of!

Tide data arriving in Tides Index! But, in default index mappings, everything is a *String*. To graph them as time series by name need a *custom* mapping.

REST call

JSON result

REST source connector

Kafka Connect Cluster

Source Connector

Elastic sink connector

Elasticsearch

Tides Index

```
{"metadata": {
    "id":"8724580",
    "name":"Key West",
    "lat":"24.5508",
    "lon":"-81.8081"},
 "data":[{
    "t":"2020-09-24 04:18",
    "v":"0.597"}]}
```

Kafka Cluster

Tides Topic

Kibana

```
{"metadata": {
    "id":"String",
    "name":"String",
    "lat":"String",
    "lon":"String"},
 "data":[{
    "t":"String",
    "v":"String"}]}
```

Custom mapping ➡ “t” is a **date**, “v” is a **double**, and “name” is a **keyword.**

```
curl -u elasticName:elasticPassword ”elasticURL:9201/tides-index"  -X PUT -H 'Content-Type: application/json' -d'
{
"mappings" : {
  "properties" : {
    "data" : {
      "properties" : {
          "t" : { "type" : "date",
                "format" : "yyyy-MM-dd HH:mm"
          },
          "v" : { "type" : "double" },
          "f" : { "type" : "text" },
          "q" : { "type" : "text" },
          "s" : { "type" : "text" }
          }
      },
      "metadata" : {
        "properties" : {
          "id" : { "type" : "text" },
          "lat" : { "type" : "text" },
          "long" : { "type" : "text" },
          "name" : { "type" : ”keyword" } }}}}        }'
```

Reindexing!

Every time you

- Change an Elasticsearch index mapping, you have to
- Delete the index
- Index all the data again

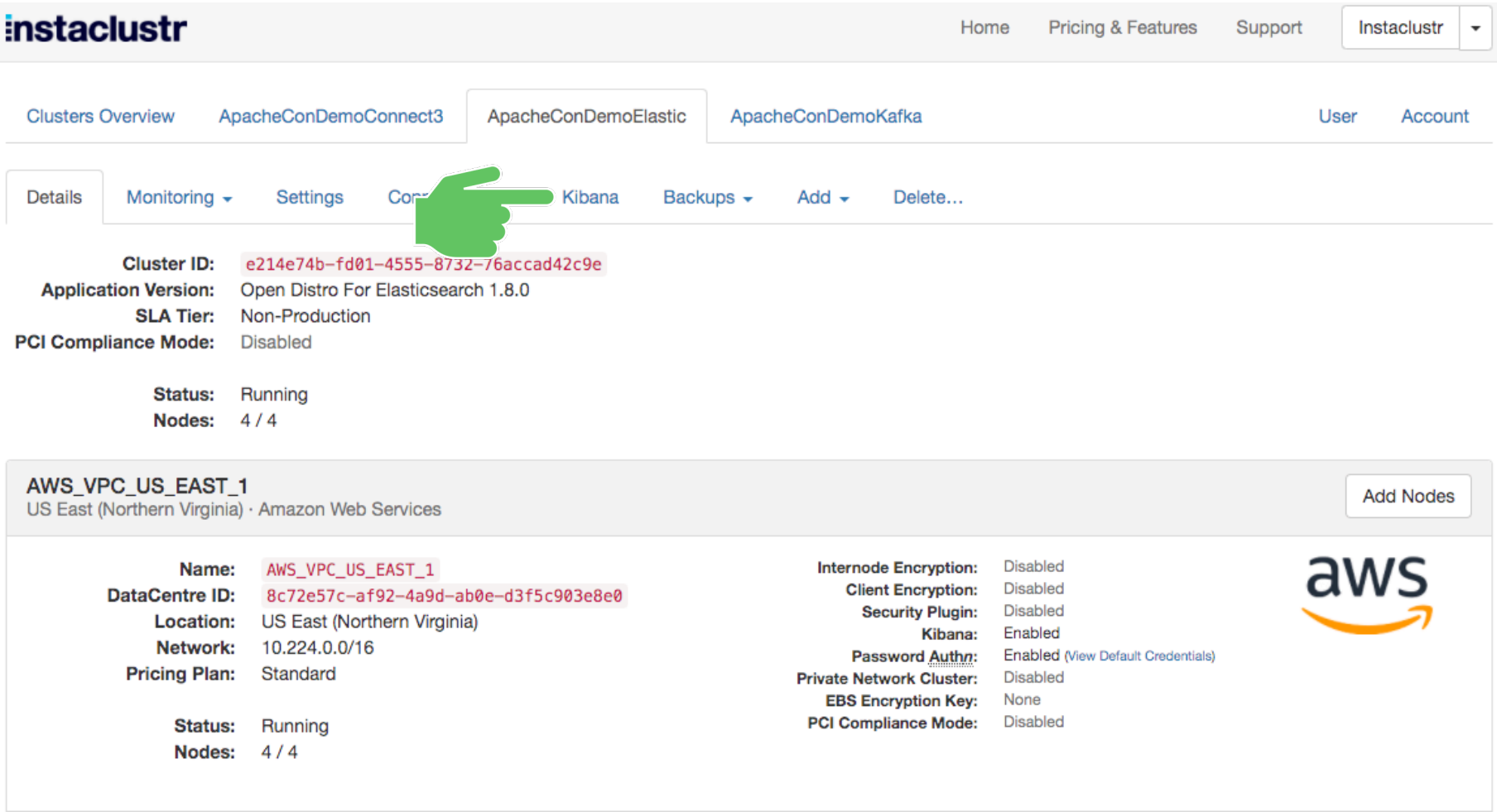
But where does the data come from?

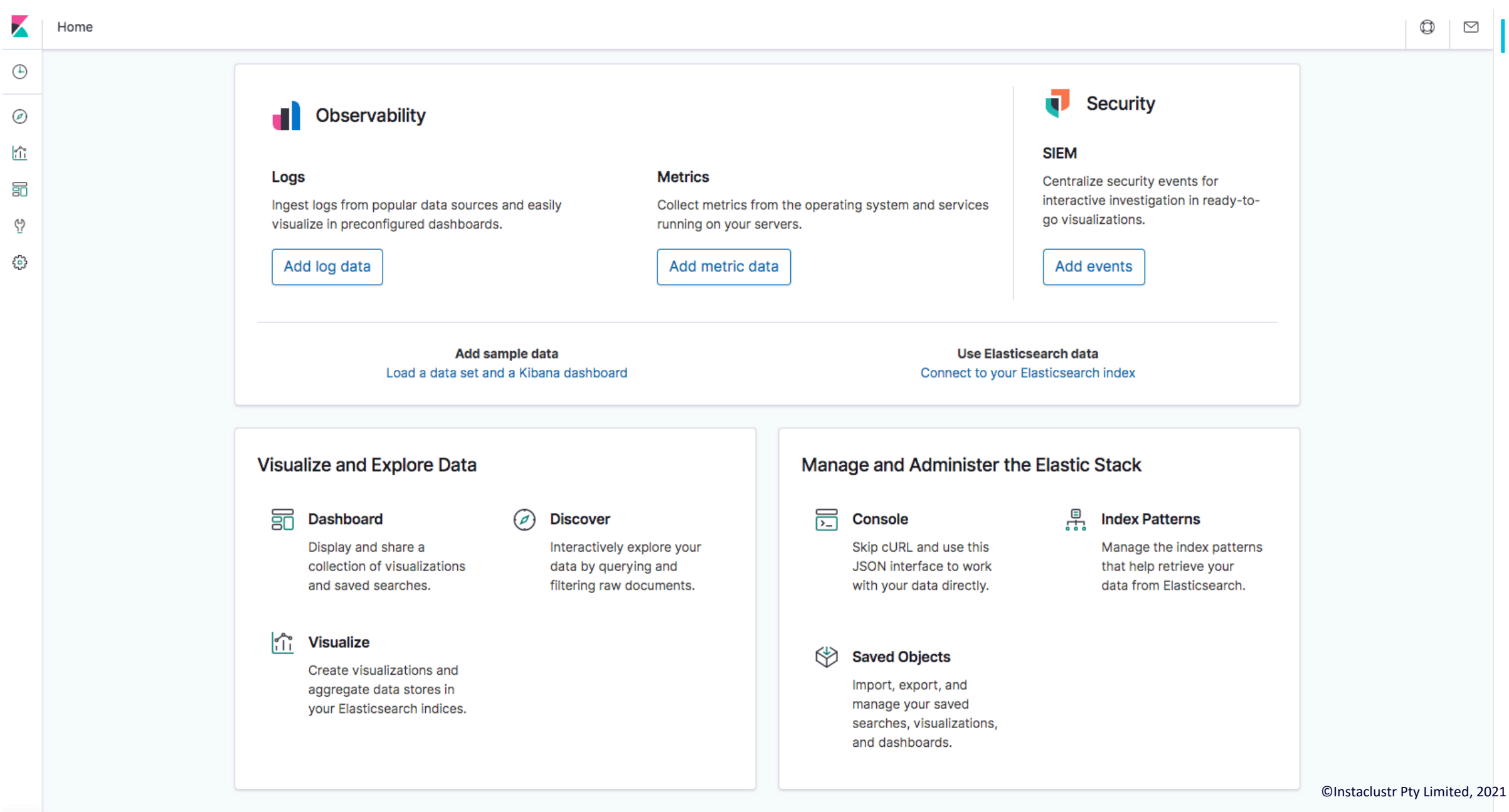
Two options:

- Using a Kafka sink connector the data is already in the Kafka topic, so just **replay** it, or,
- Use Elasticsearch **reindex** operation

The hard part is over, now…

Start Kibana With A Single Click





Visualization Steps

1: Index Pattern (to get data from Elasticsearch)

Settings -> Index Patterns -> Create Index Pattern -> Define ->
Configure with “t” as timefilter field

2. Create Visualization (to create a graph type)

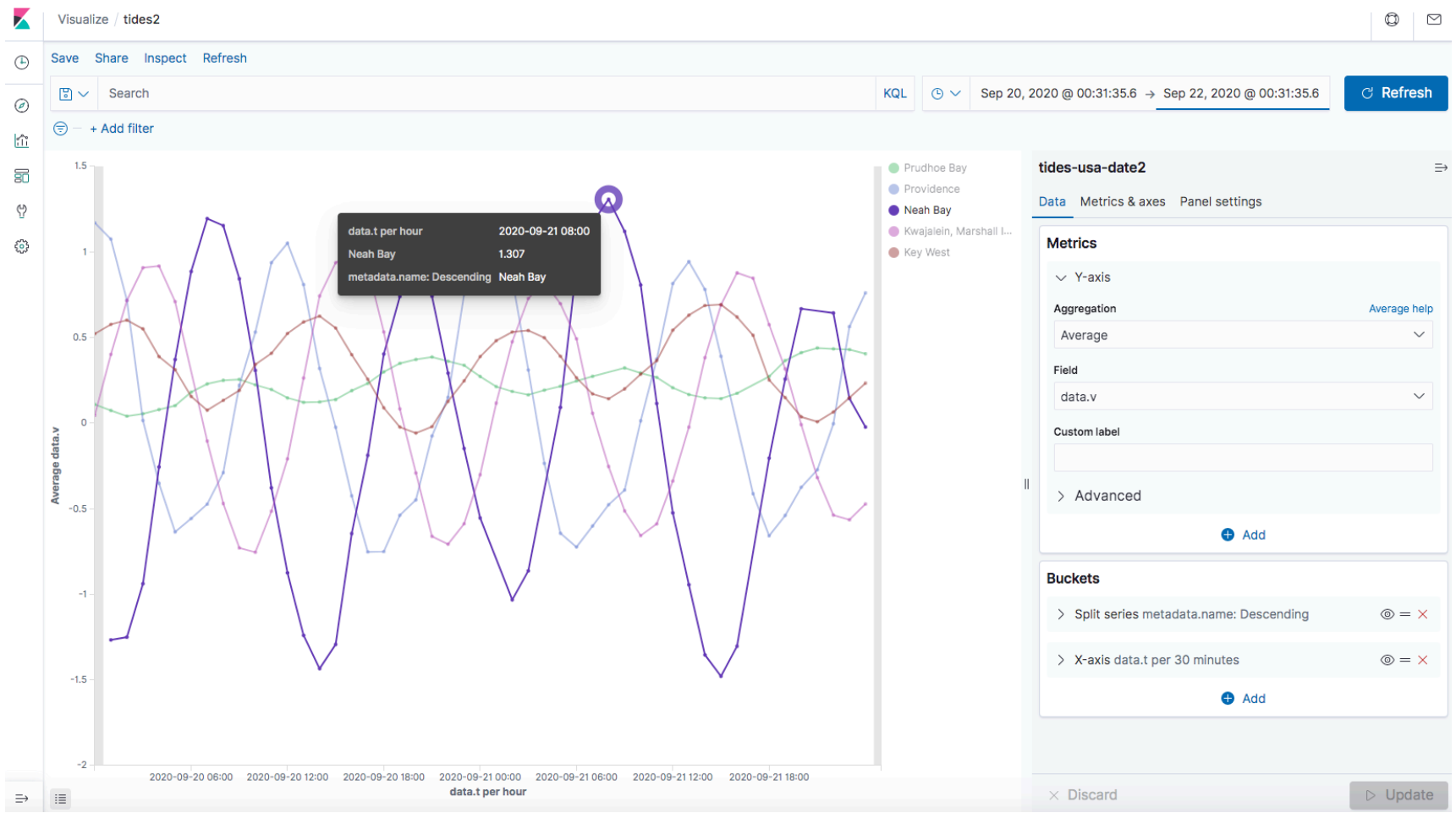
Visualizations -> Create Visualization -> New Visualization ->
Line -> Choose Source = pattern from 1

3. Configure Graph Settings (to display data correctly)

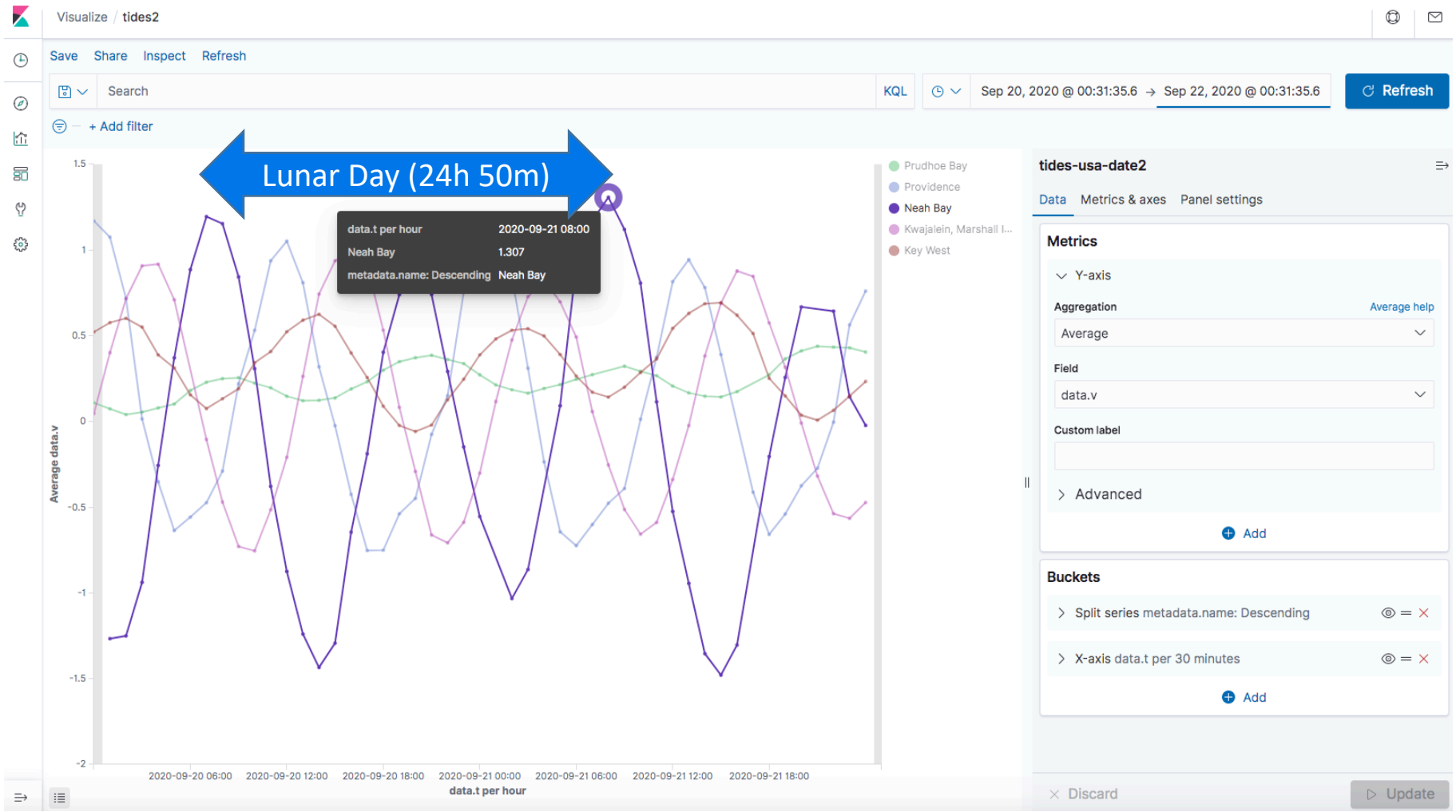
Select time range, select aggregation for y-axis = average ->
data.v -> select Buckets -> Split series metadata.name -> X-axis
-> Data Histogram = data.t

instaclustr

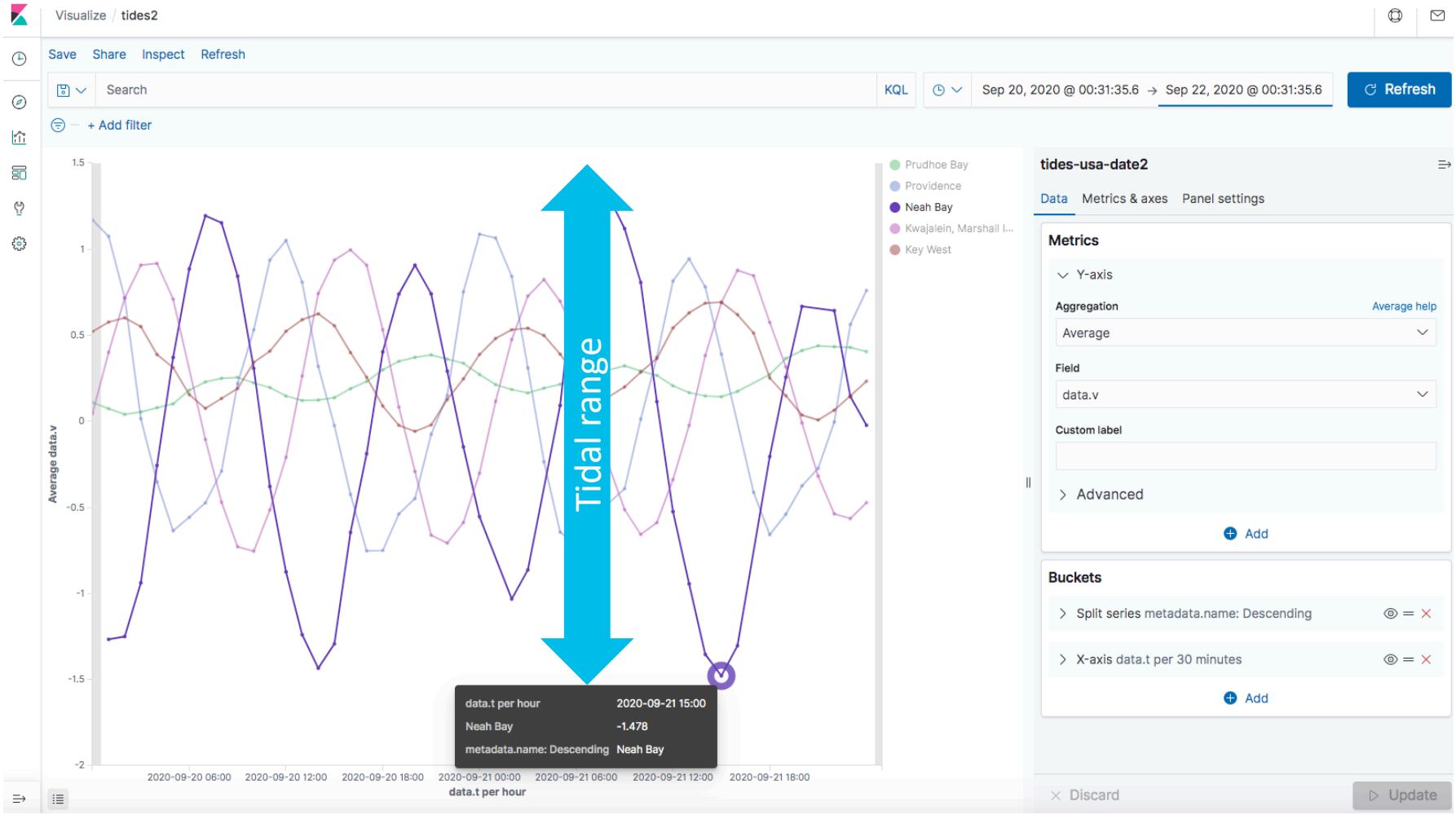
Time (x axis) vs. average (over 30m) tide level (relative to average level) in meters for the 5 sample stations



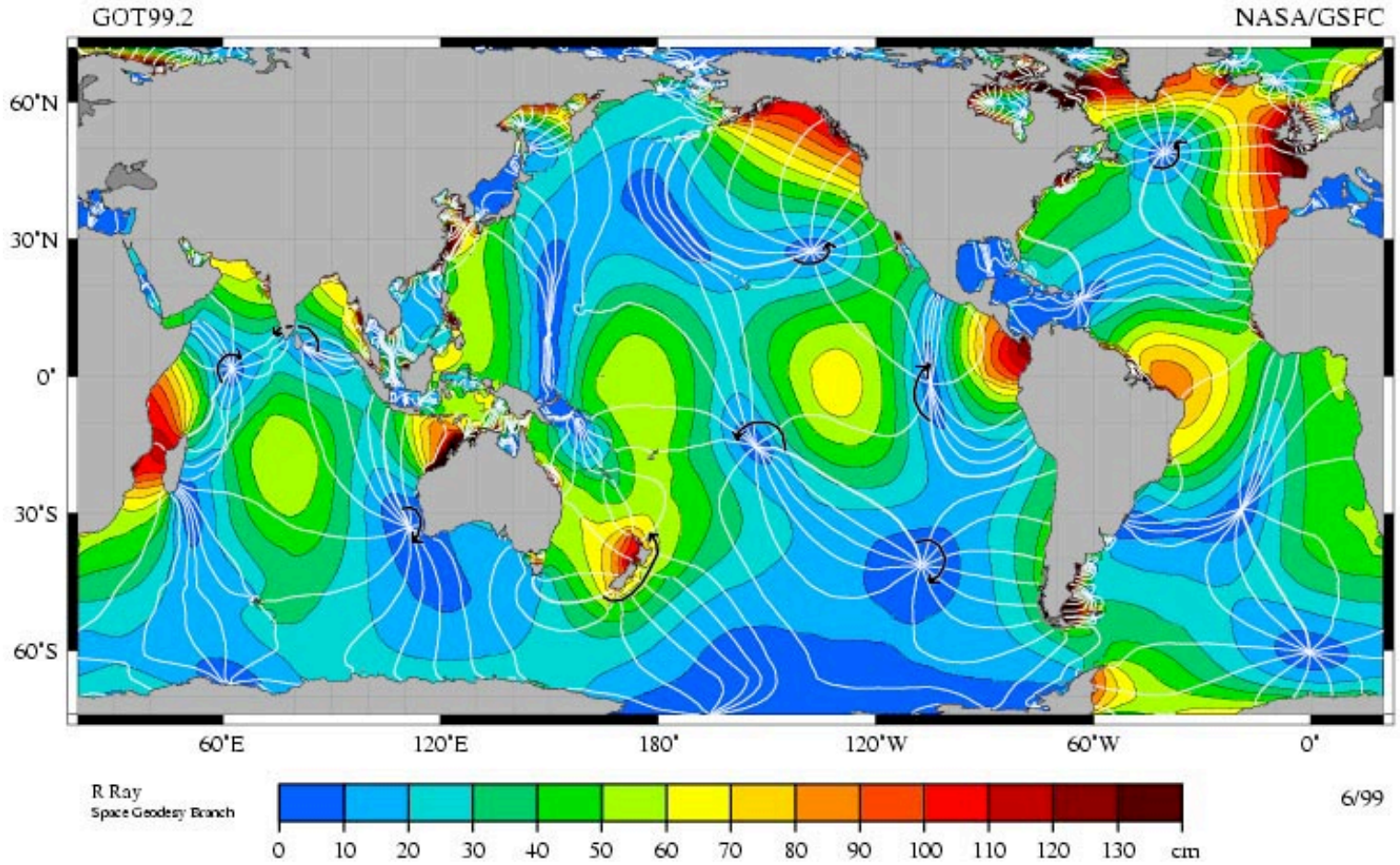
Showing Lunar Day (24 hours 50 minutes)



Showing Tidal Range (high tide – low tide)

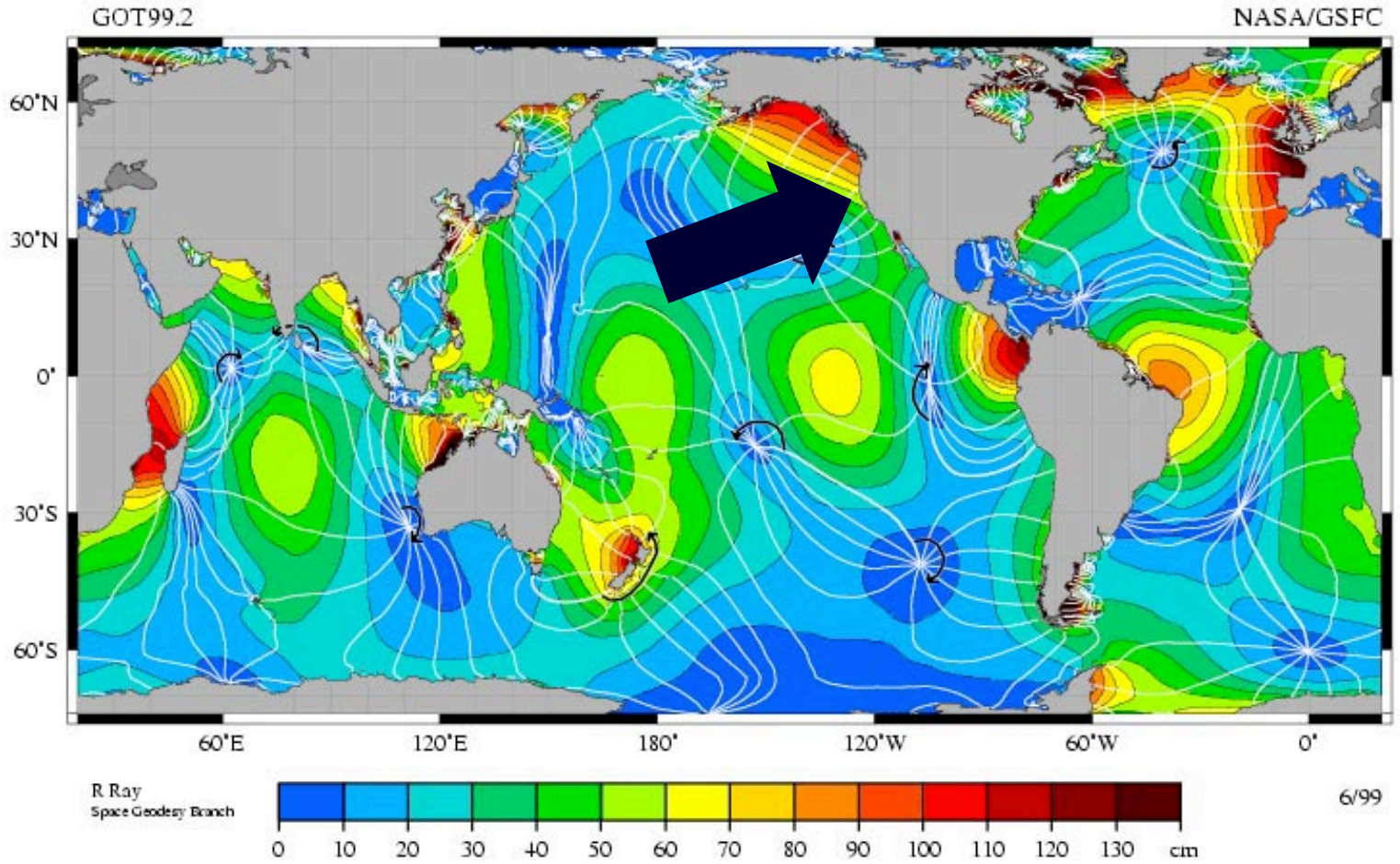


Tide range varies depending on moon, sun, local geography, and weather!



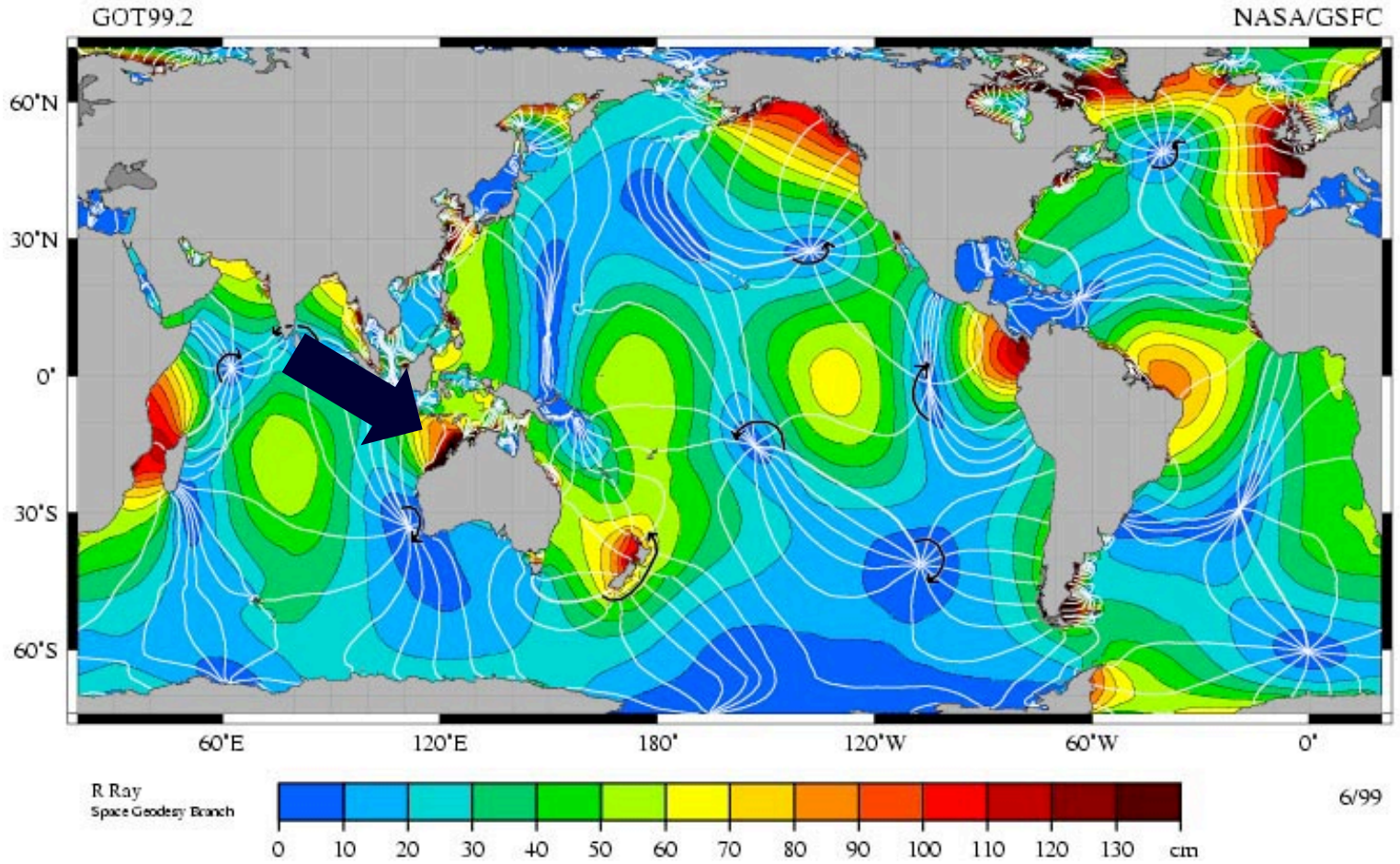
By R. Ray, NASA Goddard Space Flight Center, Jet Propulsion Laboratory, Scientific Visualization Studio - TOPEX/Poseidon: Revealing Hidden Tidal Energy, Public Domain

Neah Bay is near here



By R. Ray, NASA Goddard Space Flight Center, Jet Propulsion Laboratory, Scientific Visualization Studio - TOPEX/Poseidon: Revealing Hidden Tidal Energy, Public Domain

Australia's Biggest Tide is here



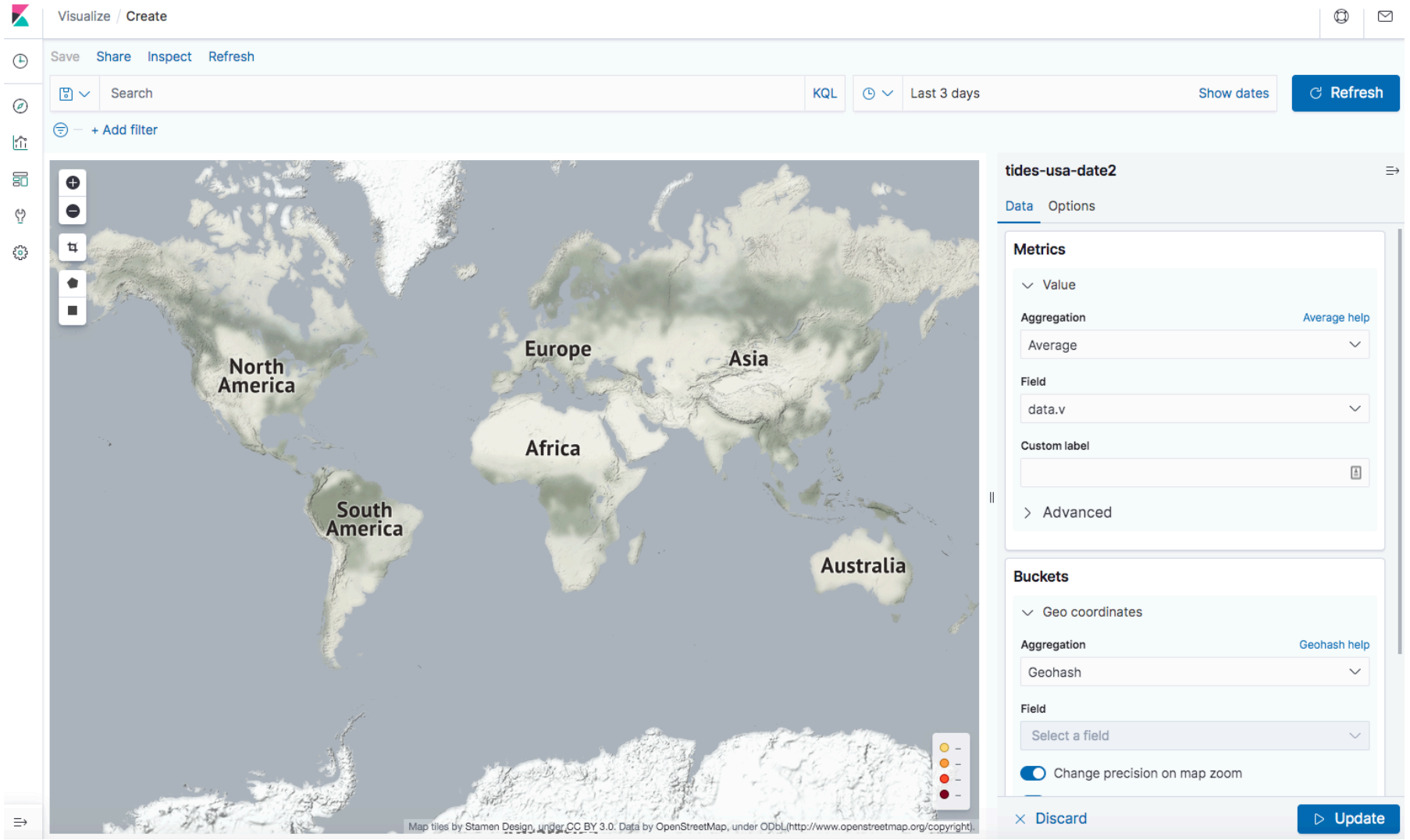
By R. Ray, NASA Goddard Space Flight Center, Jet Propulsion Laboratory, Scientific Visualization Studio - TOPEX/Poseidon: Revealing Hidden Tidal Energy, Public Domain

Tides of over 11 meters are forced through two narrow passes creating the popular tourist attraction known as the *Horizontal Waterfalls* in the Kimberley's Talbot Bay.

(Source: Shutterstock)

Next, a map to show the sensor locations to understand tidal ranges

But, there are no geo-points in the data!

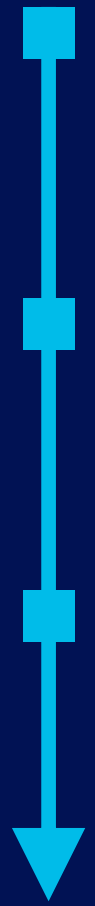


Mapping Steps

Problem:
Elasticsearch doesn't recognize separate lat lon fields as geo-points

Solution:
Add an Elasticsearch ingest pipeline to pre-process documents *before* they are indexed

(Need to reindex again)



1. Add geo-point field to index mapping

2. Create Elasticsearch ingest pipeline to construct new field

3. Add as default ingest pipeline to index

instaclustr

1. Add a new “location” field with a geo_point data type to the mapping and index

```
curl -u elasticName:elasticPassword ”elasticURL:9201/tides-index"  -X PUT -H 'Content-Type: application/json' -d'
{
"mappings" : {
 "properties" : {
   "data" : {
     "properties" : {
        "t" : { "type" : "date",
              "format" : "yyyy-MM-dd HH:mm"
        },
        "v" : { "type" : "double" },
        "f" : { "type" : "text" },
        "q" : { "type" : "text" },
        "s" : { "type" : "text" }
        }
    },
    "metadata" : {
      "properties" : {
        "id" : { "type" : "text" },
        "lat" : { "type" : "text" },
        "long" : { "type" : "text" },
       "location" : { "type" : "geo_point" },
        "name" : { "type" : ”keyword" } }}}}       }'
```

2. Create new ingest pipeline to construct new location geo-point String from existing lat lon fields

```
curl -u elasticName:elasticPassword ”elasticURL:9201/ _ingest/pipeline/locationPipe"  -X PUT -H 'Content-Type:
application/json' -d'
{
 "description" : ”construct geo-point String field",
 "processors" : [
  {
   "set" : {
    "field": "metadata.location",
    "value": "{{metadata.lat}},{{metadata.lon}}"
   }
  }
 ]
}
'
```

3. Add locationPipe as default pipeline to the index

```
curl -u elasticName:elasticPassword ”elasticURL:9201/tides-index/_settings?pretty"  -X PUT -H 'Content-Type:
application/json' -d'
{
  "index" : {
    "default_pipeline" : ”locationPipe"
  }
}
'
```

Now we have a pipeline transforming the raw data and adding geo-point location data in Elasticsearch

REST call

JSON result

REST source connector

Kafka Connect Cluster

Source Connector

Elastic sink connector

Elasticsearch

LocationPipe ingestor

Tides Index

Kafka Cluster

Tides Topic

```
{"metadata": {
    "id":"8724580",
    "name":"Key West",
    "lat":"24.5508",
    "lon":"-81.8081"},
 "data":[{
    "t":"2020-09-24 04:18",
    "v":"0.597"}]}
```

Kibana

```
{"metadata": {
    "id":"8724580",
    "name":"Key West",
    "lat":"24.5508",
    "lon":"-81.8081",
    "location": "24.5508,-81.8081"},
 "data":[{
    "t":"2020-09-24 04:18",
    "v":"0.597"}]}
```

Mapping Visualization Steps

Reuse existing index pattern

1. Create Visualization

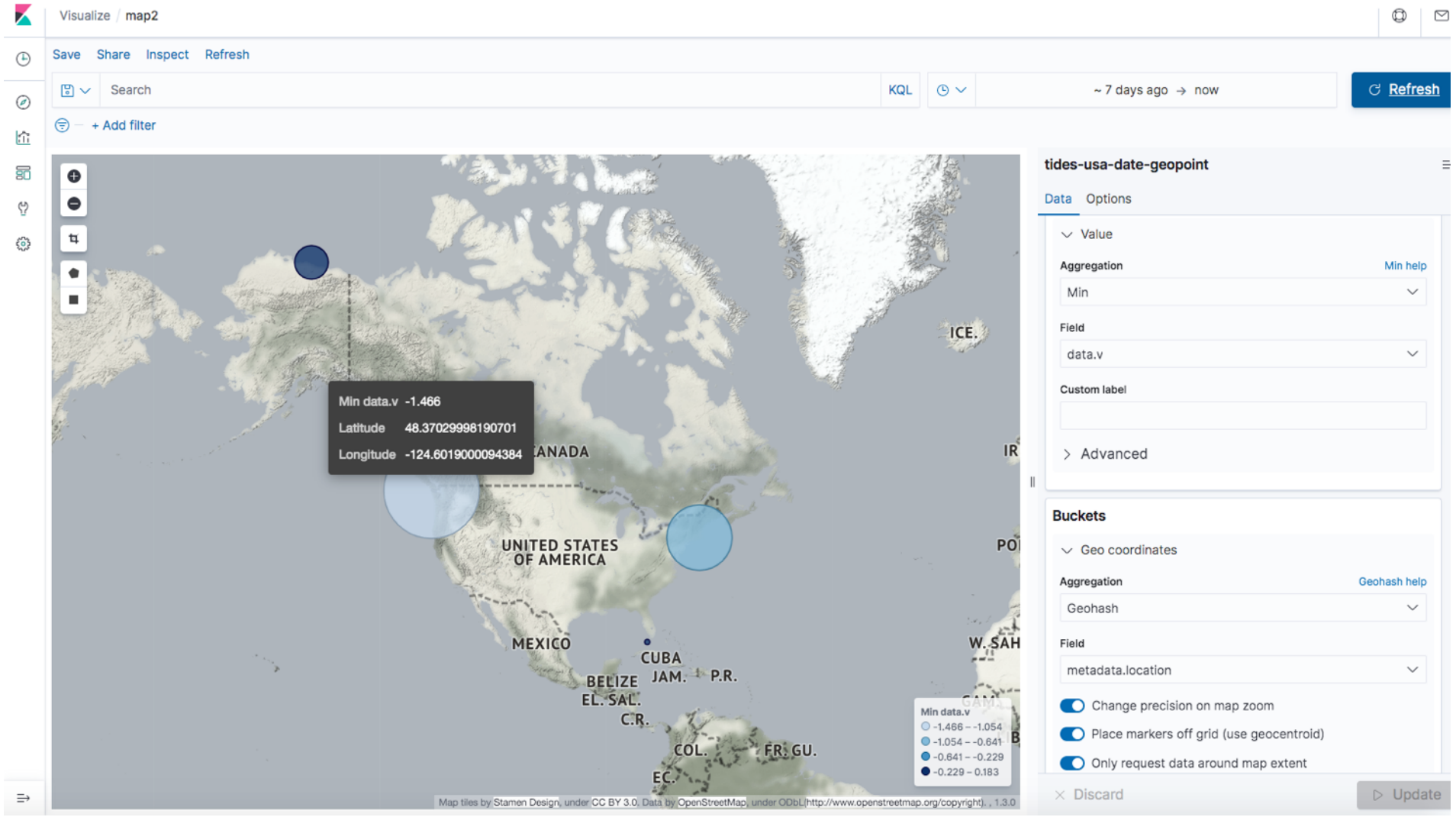
Visualizations -> Create visualization -> New Coordinate Map -> Select index patterns -> Visualization with default map

2. Configure Graph Settings (to display data correctly)

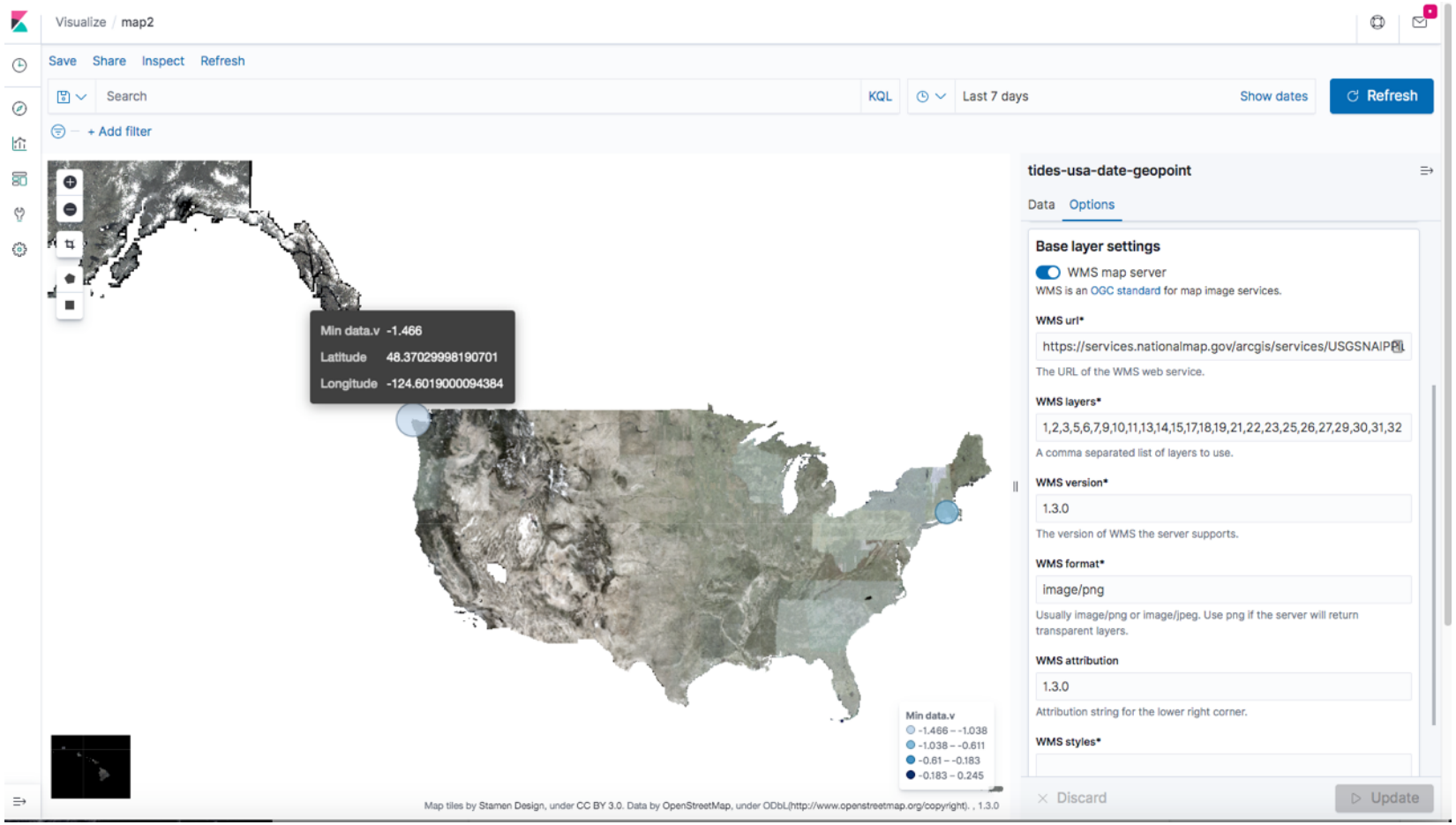
Select Metrics -> Aggregation (min) -> Field -> data.v -> Buckets -> Geo coordinates -> Geohash -> Field -> metadata.location

instaclustr

Map showing sensor locations and min values over last week



Add your own custom Web Map Service (WMS) layers



URL https://services.nationalmap.gov/arcgis/services/USGSNAIPPlus/MapServer/WMSServer

Layers 1,2,3,5,6,7,9,10,11,13,14,15,17,18,19,21,22,23,25,26,27,29,30,31,32

But What Can Go Wrong?!

(Source: Shutterstock)

1. REST call can return JSON error message, but doesn't treat it as an error, so it's sent to Tides Topic and to Elastic index.

REST call

JSON result

REST source connector

Kafka Connect Cluster

Source Connector

Elastic sink connector

Elasticsearch

LocationPipe ingestor

Tides Index


```
{"error": {"message":"No
    data was found. This
    product may not be
    offered at this station
    at the requested
    time."}}
```

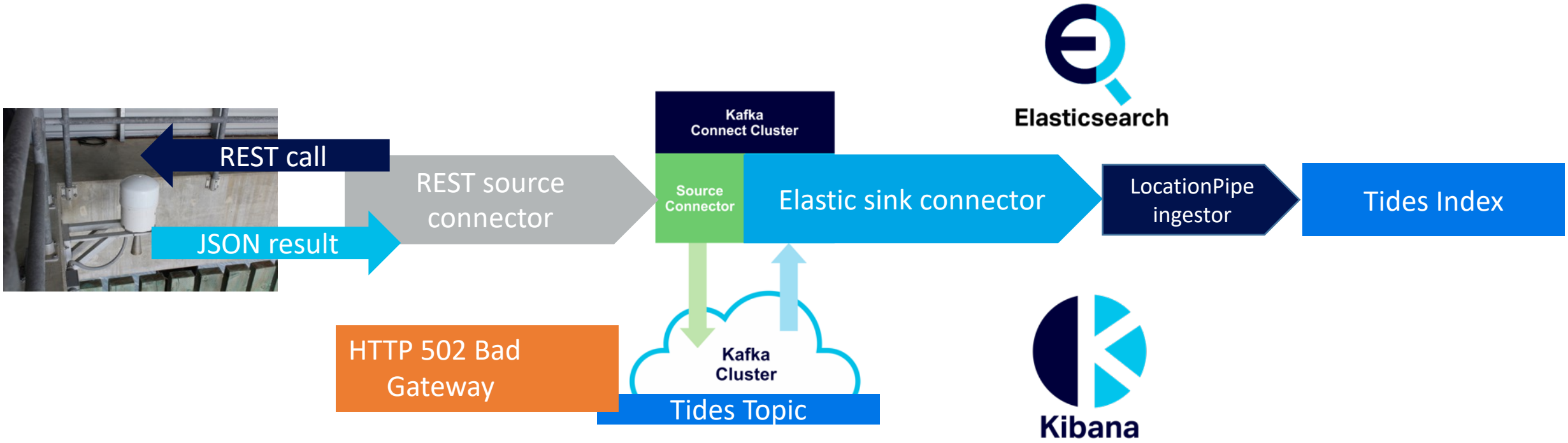

Kafka Cluster

Tides Topic


```
{"error": {"message":"No
    data was found. This
    product may not be
    offered at this station
    at the requested
    time."}}
```


Kibana

2. REST call can also return HTTP error message, but doesn't treat it as an error so it's sent to Tides Topic.



Elastic sink connector tries to read the HTTP error message and fails to FAILED state.

Exceptions viewable in the Kafka connect logs topic.

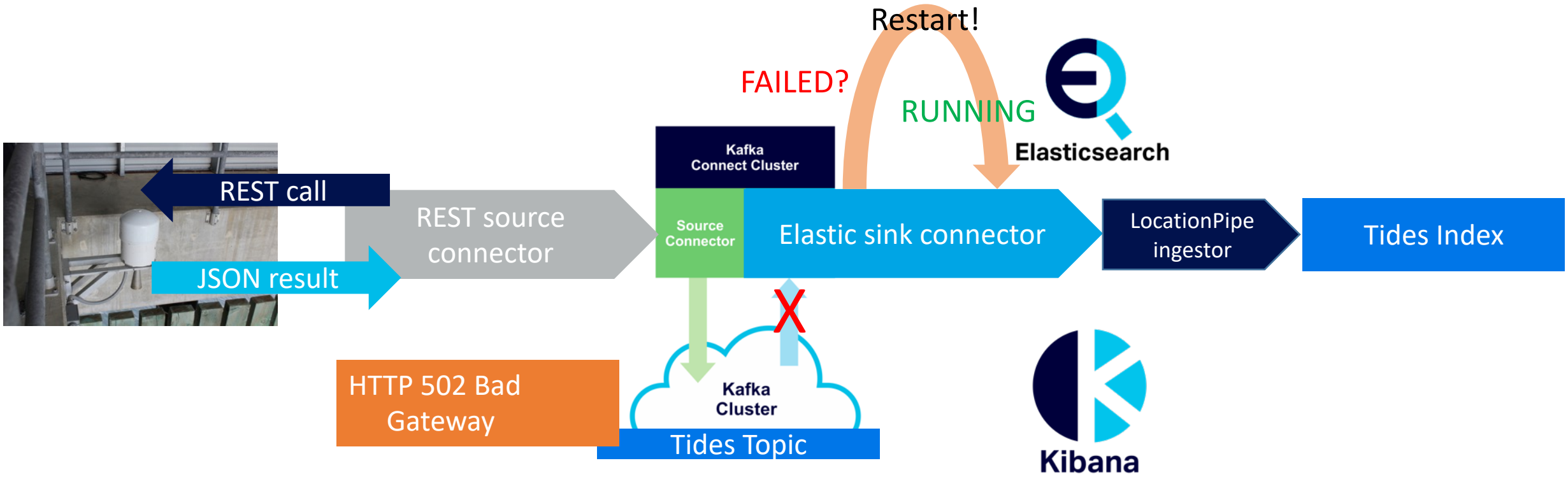
Kafka Connect framework

automatically restarts workers if they are “killed”

but “surprisingly” not if they FAIL

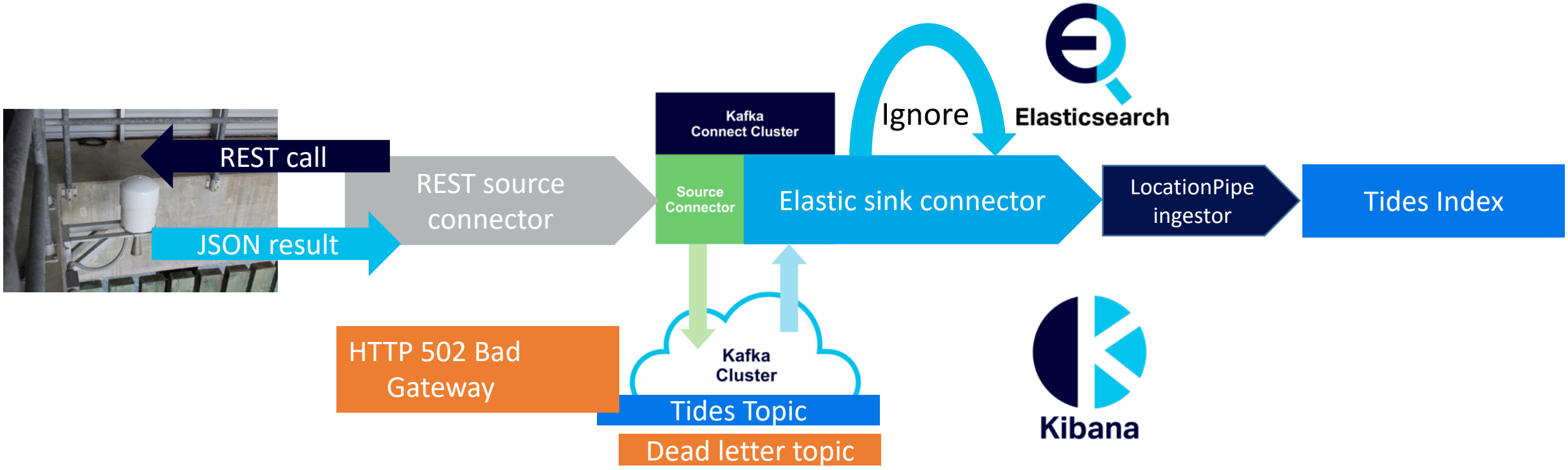
(Source: Shutterstock)

A workaround is to monitor and regularly restart failed connectors.



Better solution—if connectors support KIP-298 “Error Handling in Connect” (not all do) then configure to ignore input errors.

Errors sent to ”dead letter” topic.



Where Can We Find Connectors With More Robust Error Handling?

(Source: Shutterstock)

Where Can We Find Connectors With More Robust Error Handling?

instaclustr

(Source: Shutterstock)

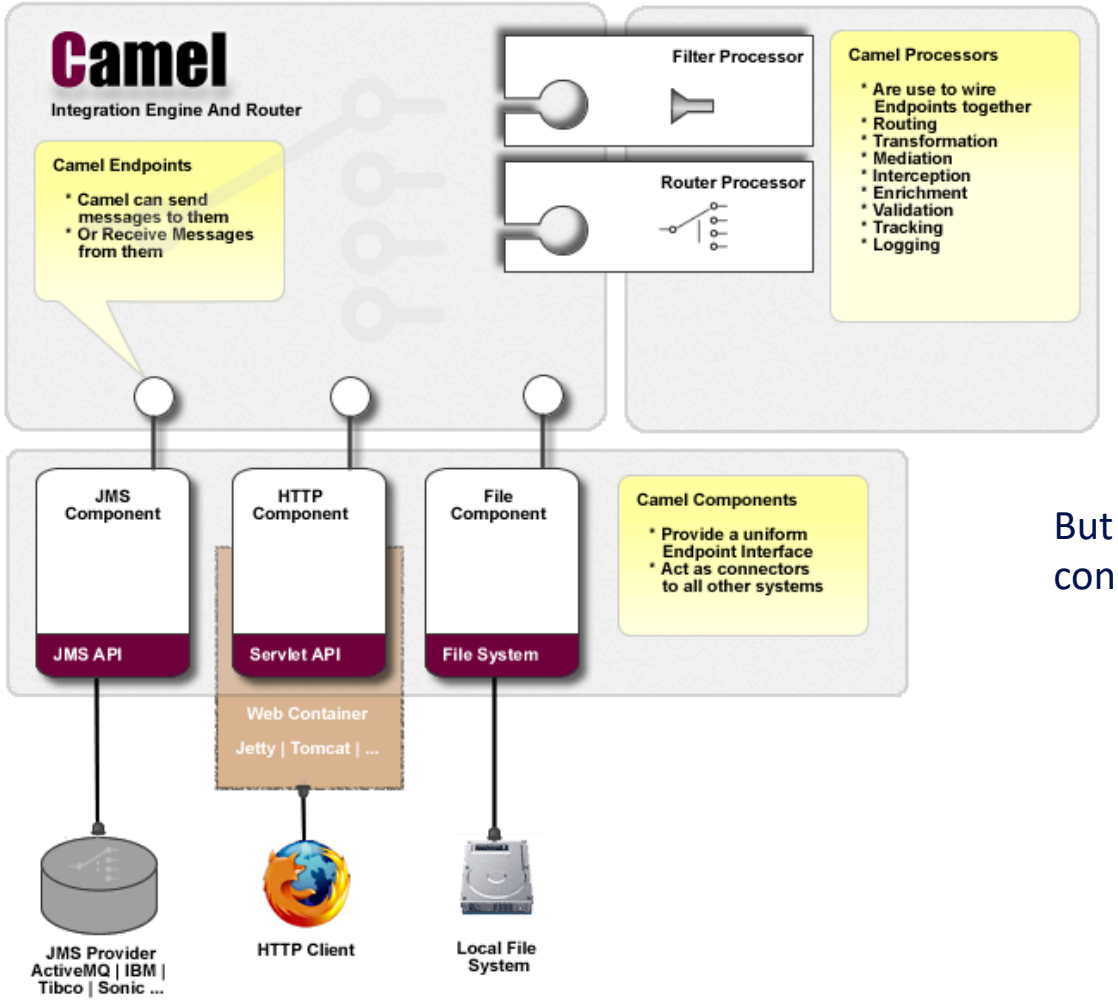
Australia Has the Most Wild Camels in the World, and Apache Camel Has 346 Kafka Connectors

instaclustr

(Source: Shutterstock)

Apache CAMEL: 10 Years Old, for Integration

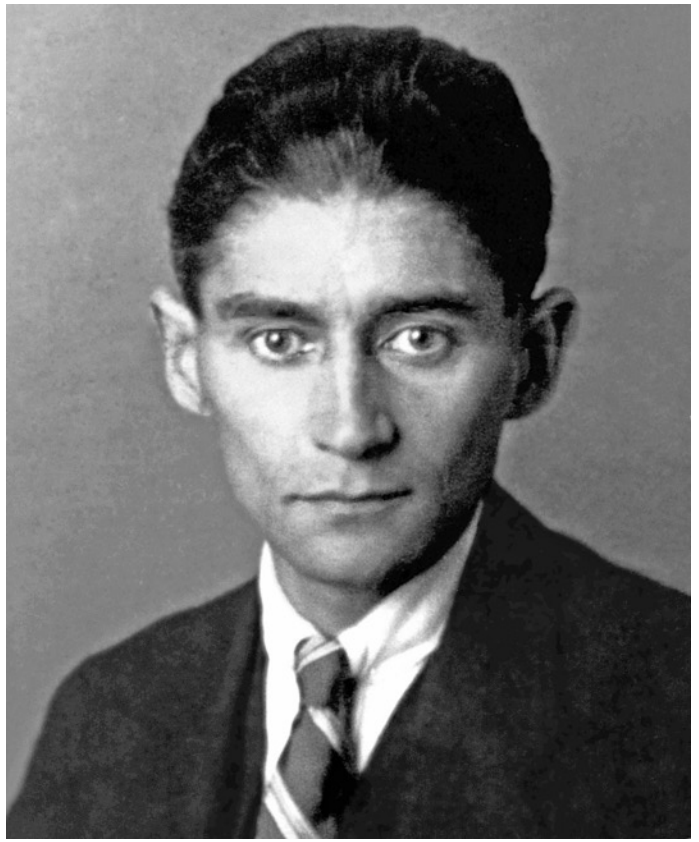
Lots of components (355), to integrate with everything



But where did so many Kafka connectors come from?

Camel Meets Kafka

(Source: Shutterstock)



(Source: Wikipedia)

Camel Kafka Connector

Camel Kafka Connector

New sub-project enables Camel components to be used as Kafka Connect Connectors.

How?

- Automatic generation of Connectors from Camel Components
- Source and/or Sink connectors (depending on Camel components)
- Kafka connector configurations contain a mixture of Kafka connect and Camel configuration
- Need to read the documentation for the Camel component **and** the Camel Kafka connector
- Trial and error to find all the required fields.

Test Ride

(Source: Shutterstock)

Camel Kafka Elasticsearch Sink Connector Configuration

instaclustr

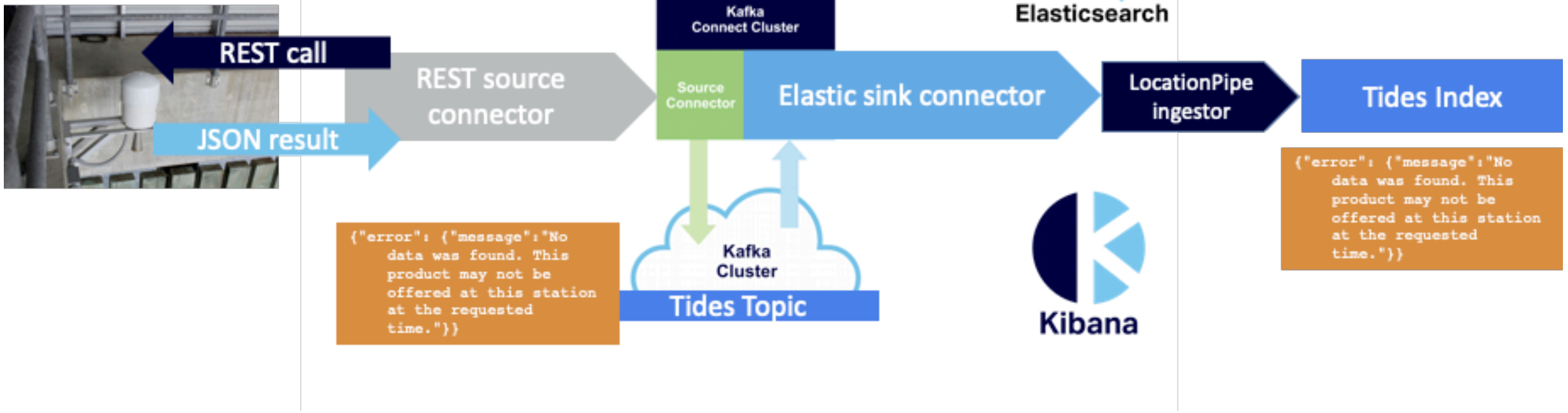
```
curl https://ip:port/connectors/camel-elastic-stocks20p/config -k -u user:password -X PUT -H 'Content-
Type: application/json' -d '
{
   "connector.class" :
"org.apache.camel.kafkaconnector.elasticsearchrest.CamelElasticsearchrestSinkConnector",
   "tasks.max" : 1,
   "topics" : "tides-topic",
   "camel.sink.endpoint.operation" : "Index",
   "camel.sink.endpoint.indexName" : "tides-index",
   "camel.sink.path.clusterName" : "elasticsearch",
   "camel.sink.endpoint.hostAddresses" : "ip:port",
   "camel.component.elasticsearch-rest.user" : "user",
   "camel.component.elasticsearch-rest.password" : "password",
   "errors.tolerance": "all",
   "errors.deadletterqueue.topic.name" : "camel-elastic-deadletter",
   "errors.log.enable" : "true",
   "errors.log.include.messages" : "true",
   "value.converter.schemas.enable": "false",
   "value.converter": "org.apache.kafka.connect.json.JsonConverter"
}'
```

Note: ”-X PUT” creates or updates a connector

clusterName is a Camel convention for endpoints

Now Have Robust Handling of HTTP Errors Connector Keeps Running

But JSON Error Messages Still Passed Through



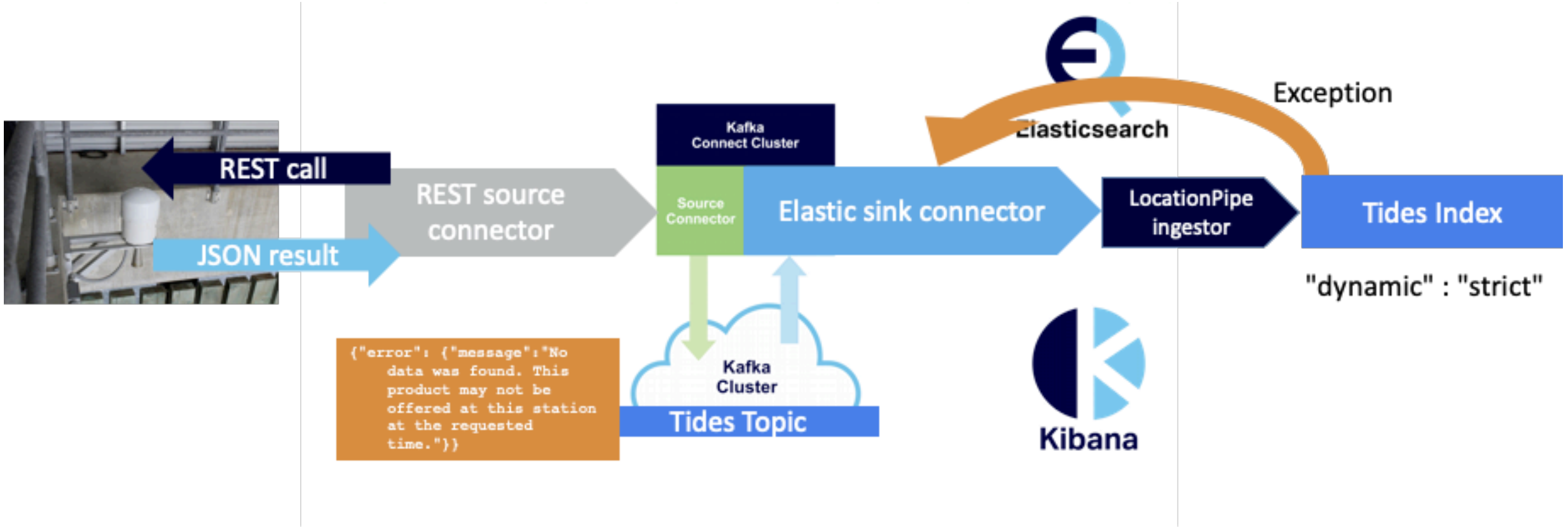
Can we get Elasticsearch to Reject “Error” JSON With “Schema Validation”?

instaclustr

```
curl -u elasticName:elasticPassword ”elasticURL:9201/tides-index"  -X PUT -H 'Content-Type:
application/json' -d'
{
"mappings" : {
  "dynamic" : "strict",
  "properties" : {
    "data" : {
      "properties" : {
          "t" : { "type" : "date",
                  "format" : "yyyy-MM-dd HH:mm"
          },
          "v" : { "type" : "double" },
          "f" : { "type" : "text" },
          "q" : { "type" : "text" },
          "s" : { "type" : "text" }
          }
      },
      "metadata" : {
        "properties" : {
          "id" : { "type" : "text" },
          "lat" : { "type" : "text" },
          "long" : { "type" : "text" },
         "location" : { "type" : "geo_point" },
          "name" : { "type" : ”keyword" } }}}}        }'
```

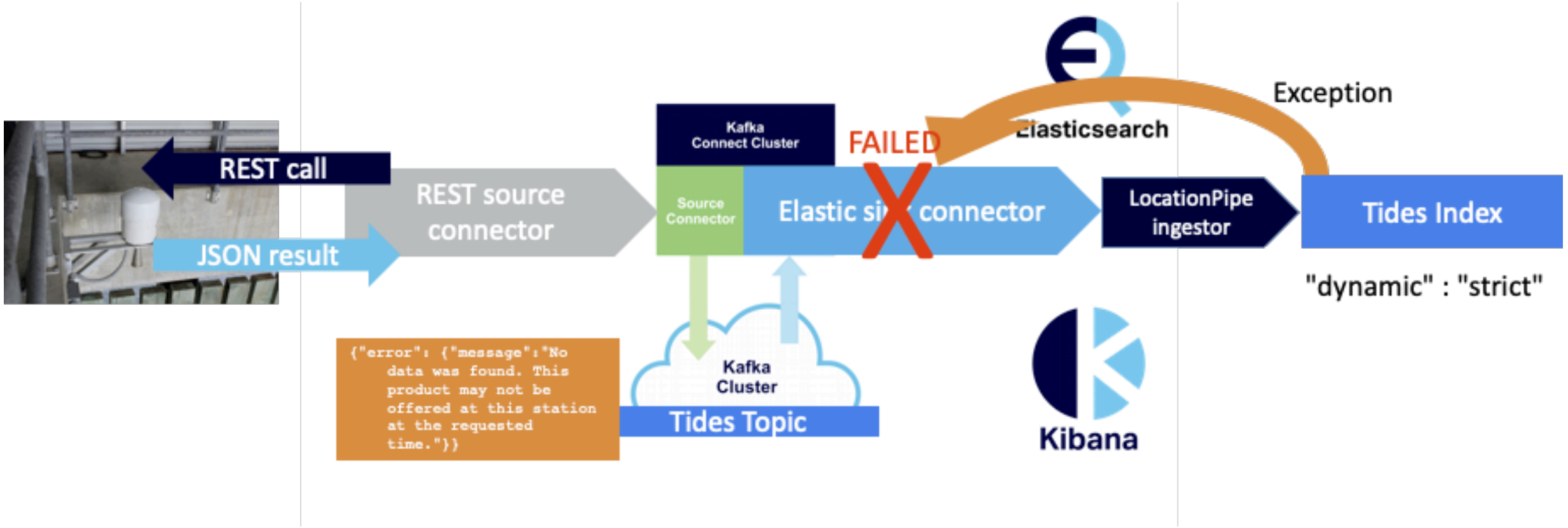
“strict” = don’t allow new fields (such as “error”)

Elasticsearch Rejects JSON With "error" Field



But the Connector Fails Can't Be Configured to Ignore Sink Errors (Future Improvement)

instaclustr



Camel Kafka Connect Conclusions? More Robust Than Other Open Source Connectors So Continue Using

instaclustr

(Source: Shutterstock)

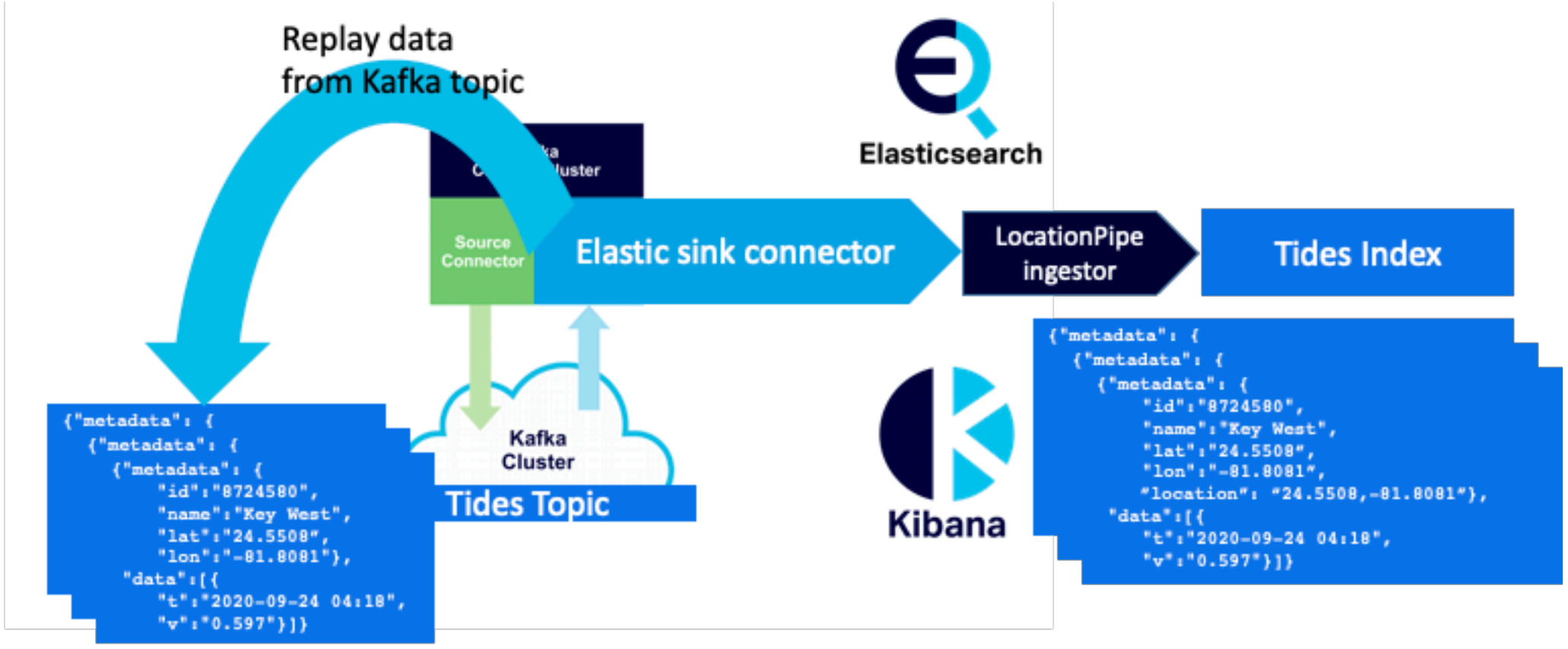
Scaling the Pipeline—
Open the Floodgates

instaclustr

(Source: Shutterstock)

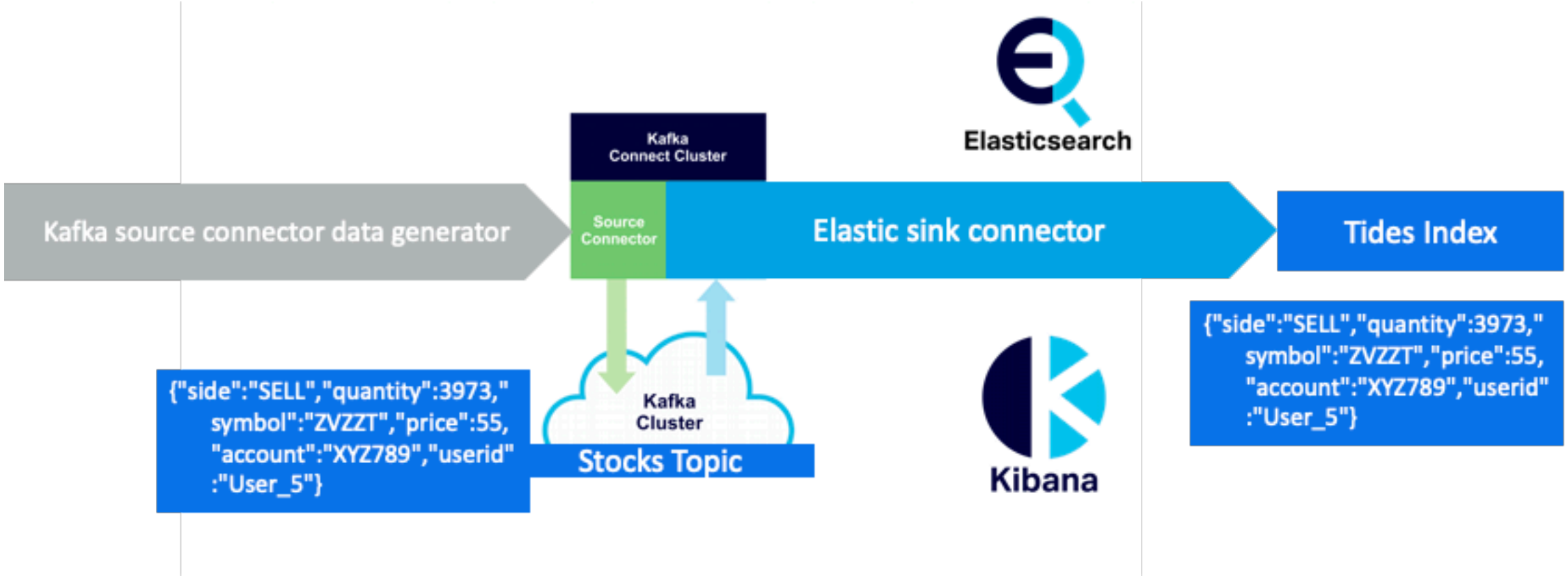
How to Increase Load? 1 - Replay Events

But can’t control rate

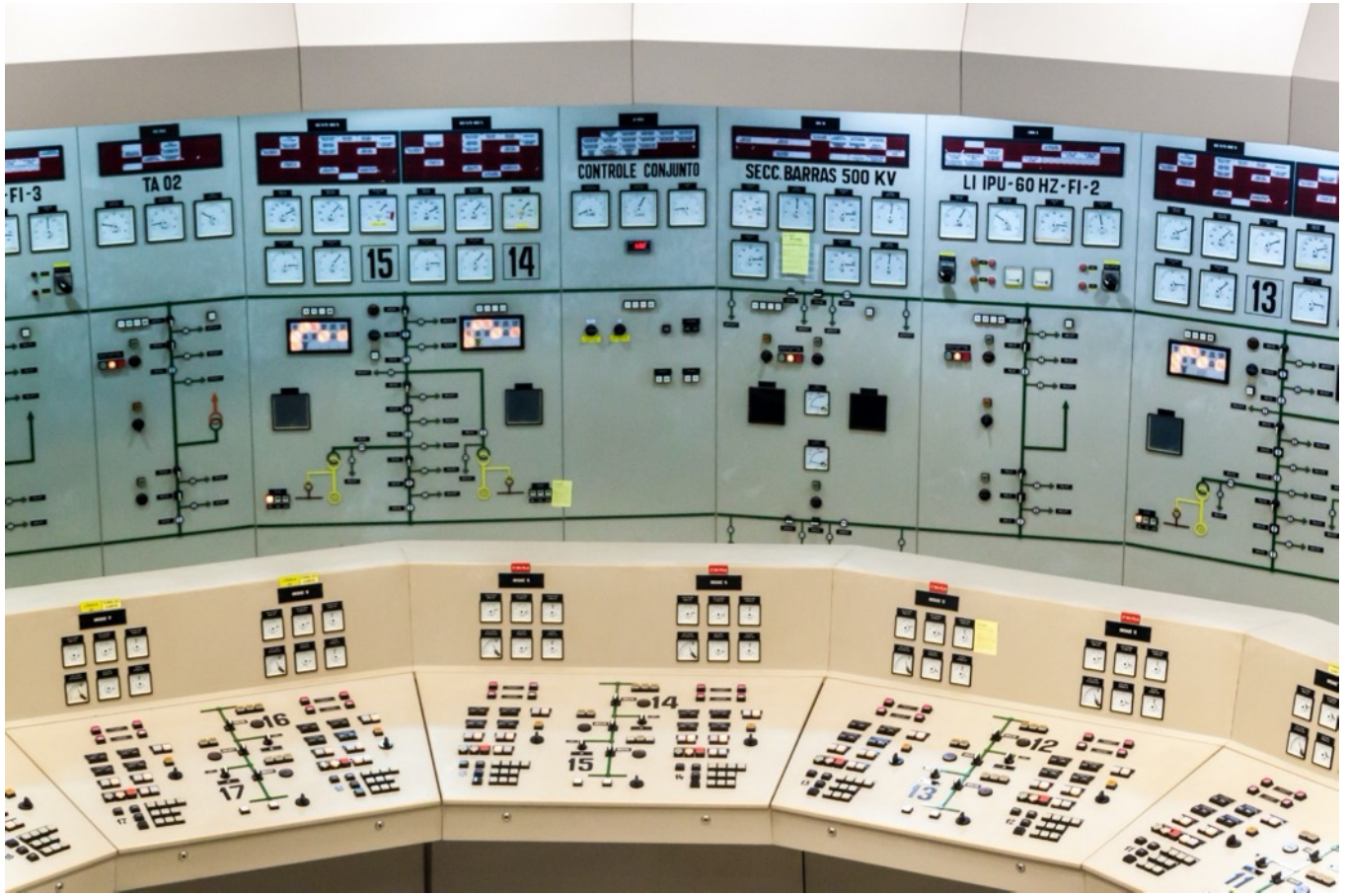


How to Increase Load? 2 - kafka-connect-datagen

Easy to control load (source tasks or time), but change from Tides to Stocks story

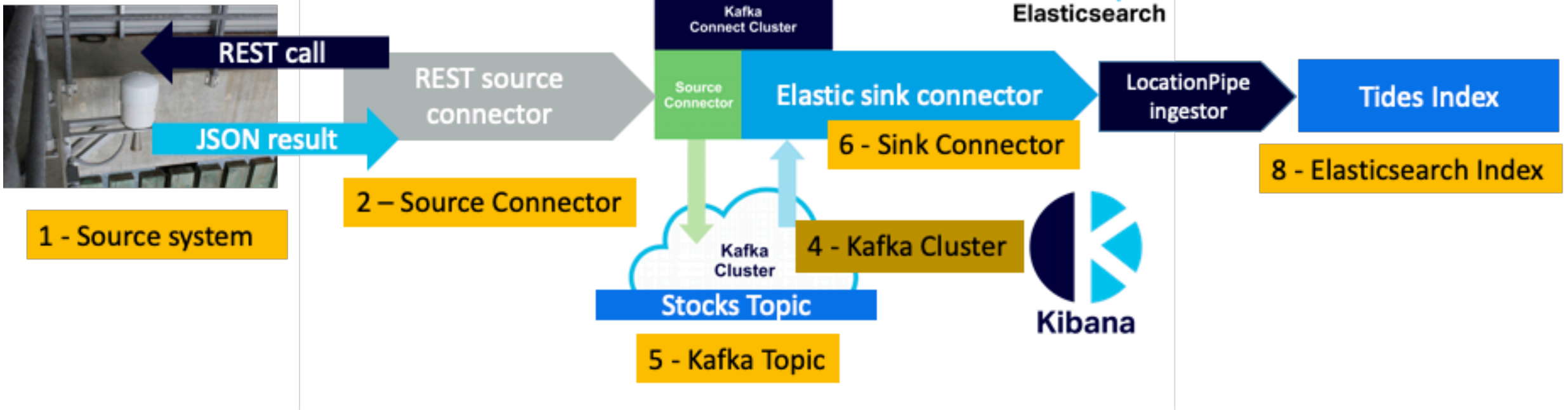


What to Measure?



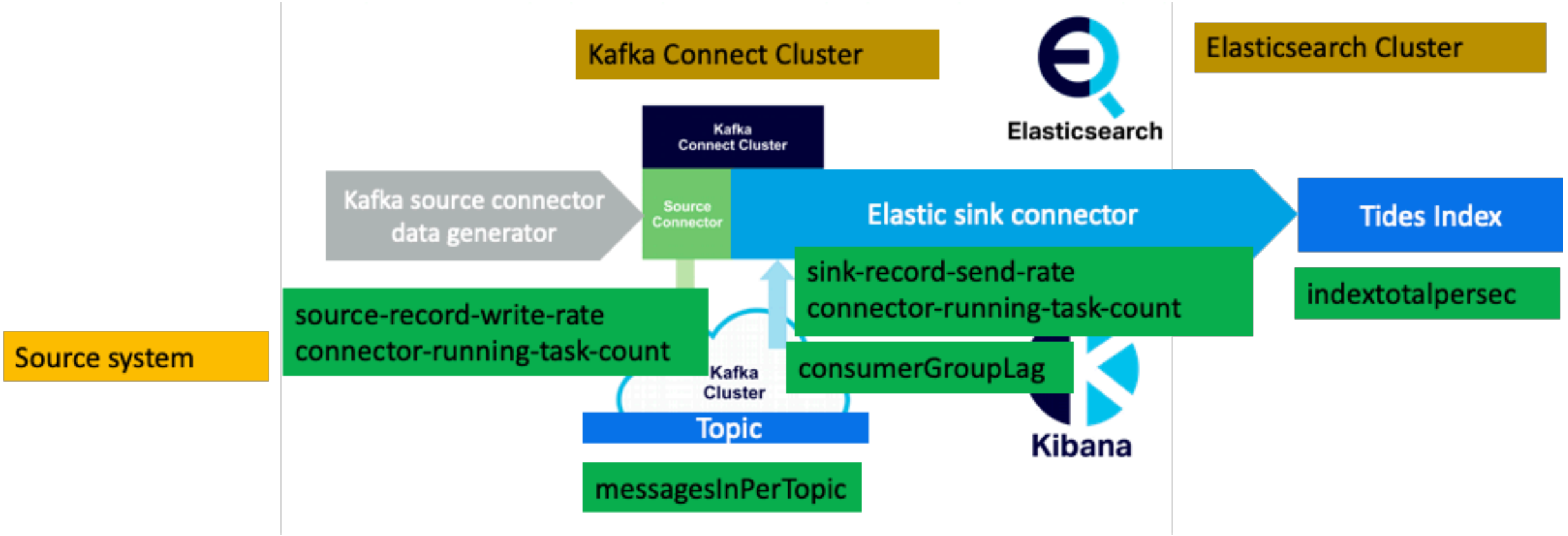
(Source: Shutterstock)

Lots of Systems and Possible Metrics



Metrics? source, sink, cluster, connector, tasks, topics, etc

7 Relevant Metrics Selected, Collected With Prometheus From Instaclustr Monitoring API

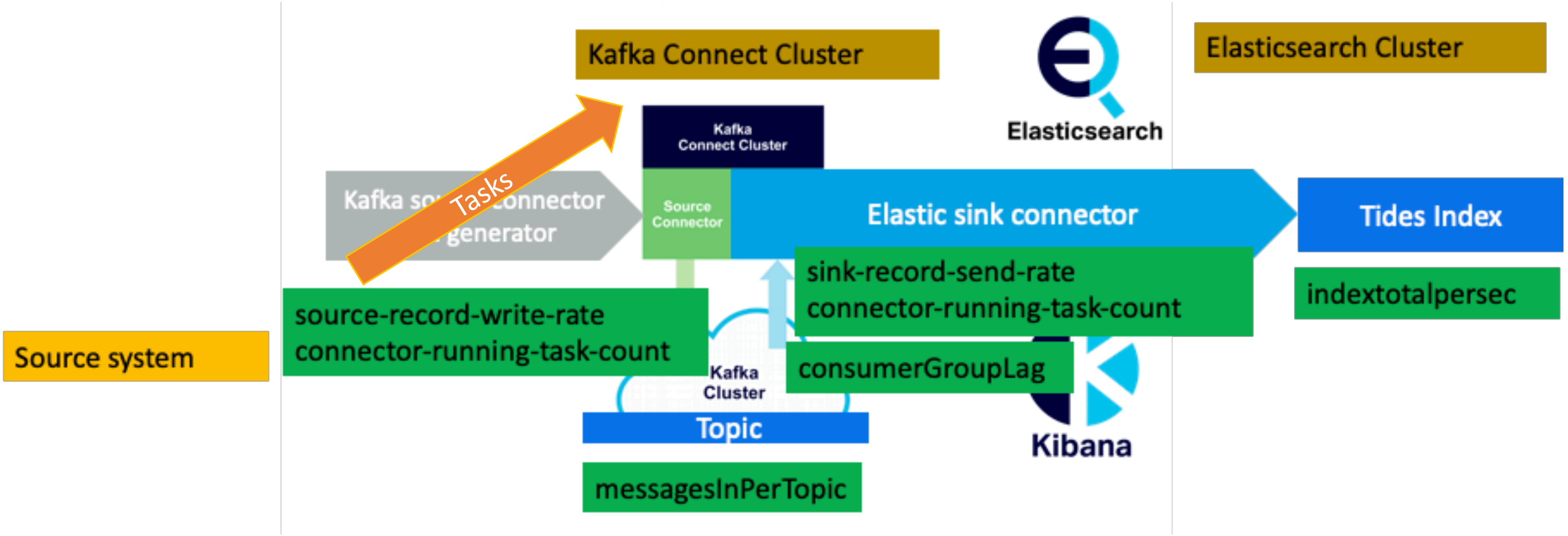


And Displayed on a Grafana Dashboard

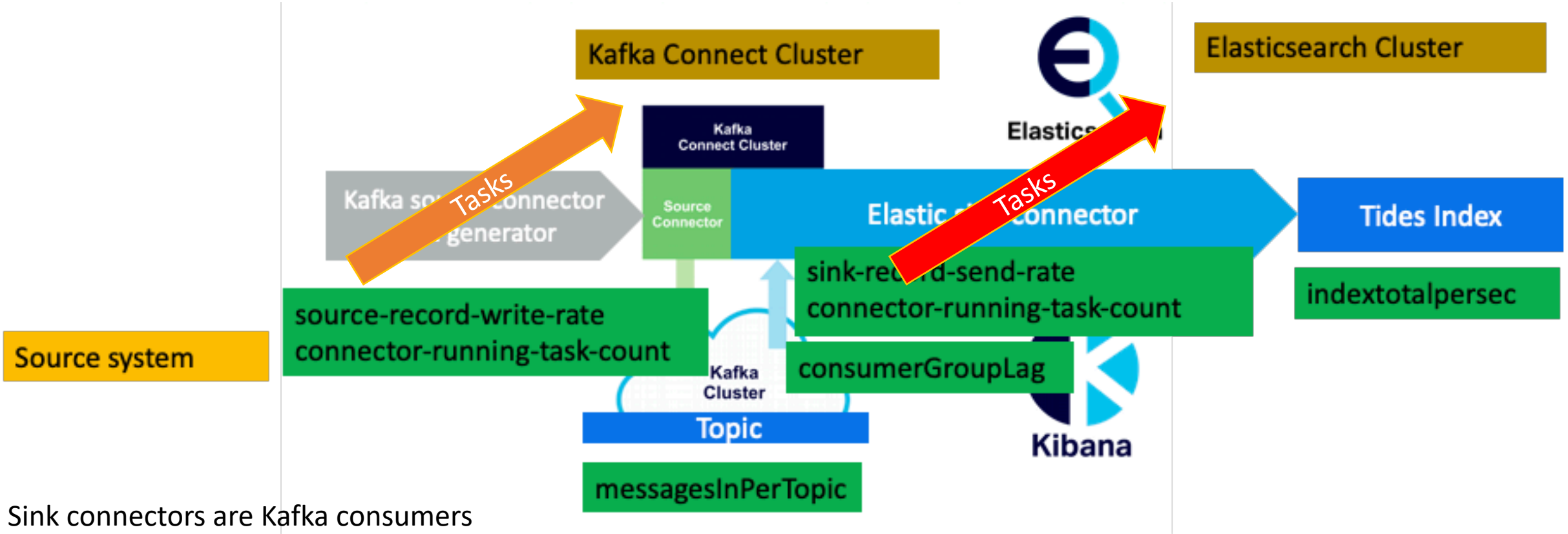
Scaling the Pipeline: Tasks

(Source: Shutterstock)

To Scale, Increase Load With Source Connector Tasks



Increase Sink Connector Tasks to Keep up (May Need Lots More)



Sink connectors are Kafka consumers

Number of Kafka consumers <= number of topic partitions

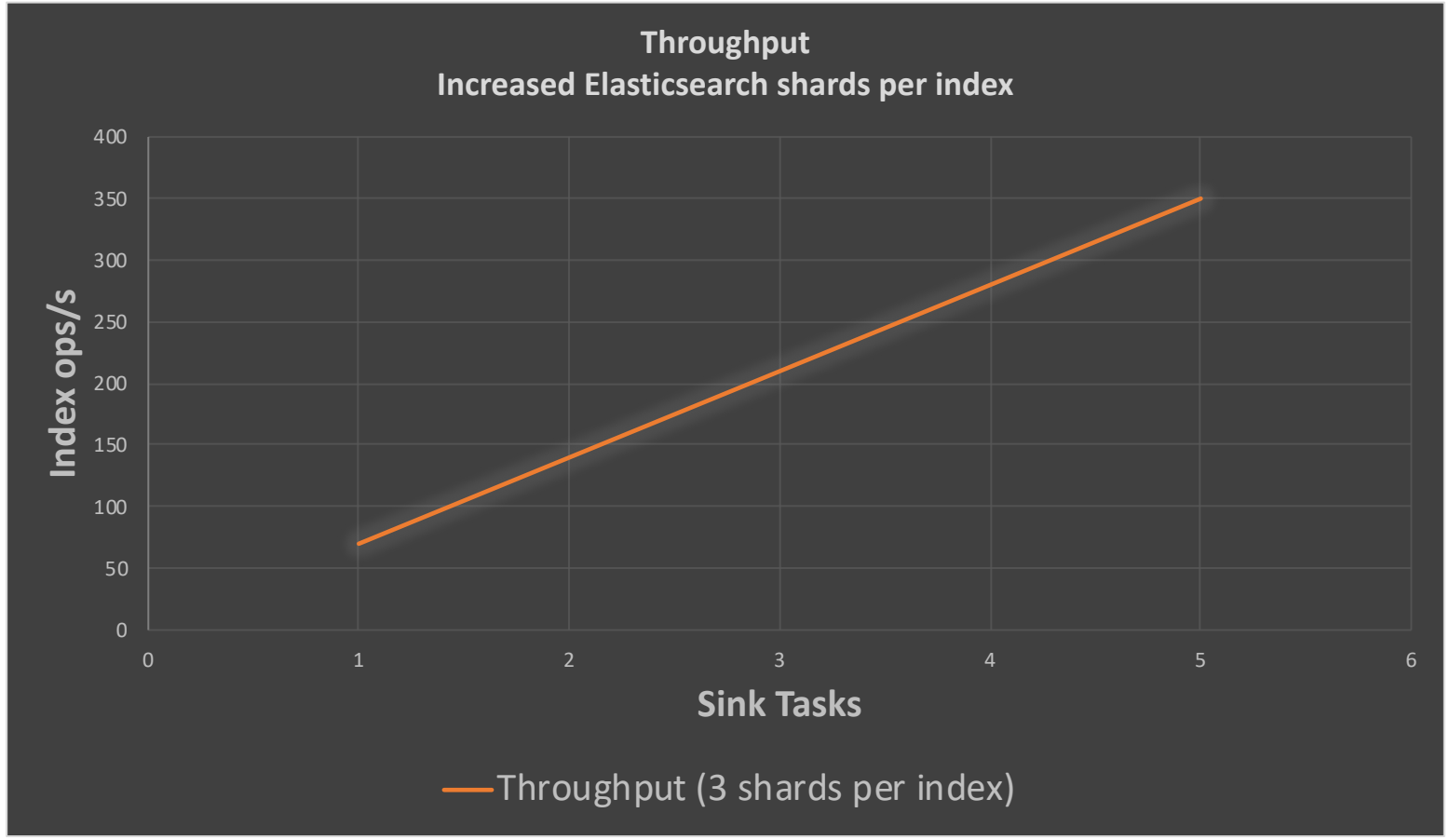
Speed Humps? 1

(Source: Shutterstock)

Speed Humps? 2

(Source: Shutterstock)

Scaling – linear with increasing Sink Tasks



Build Your Own Pipeline

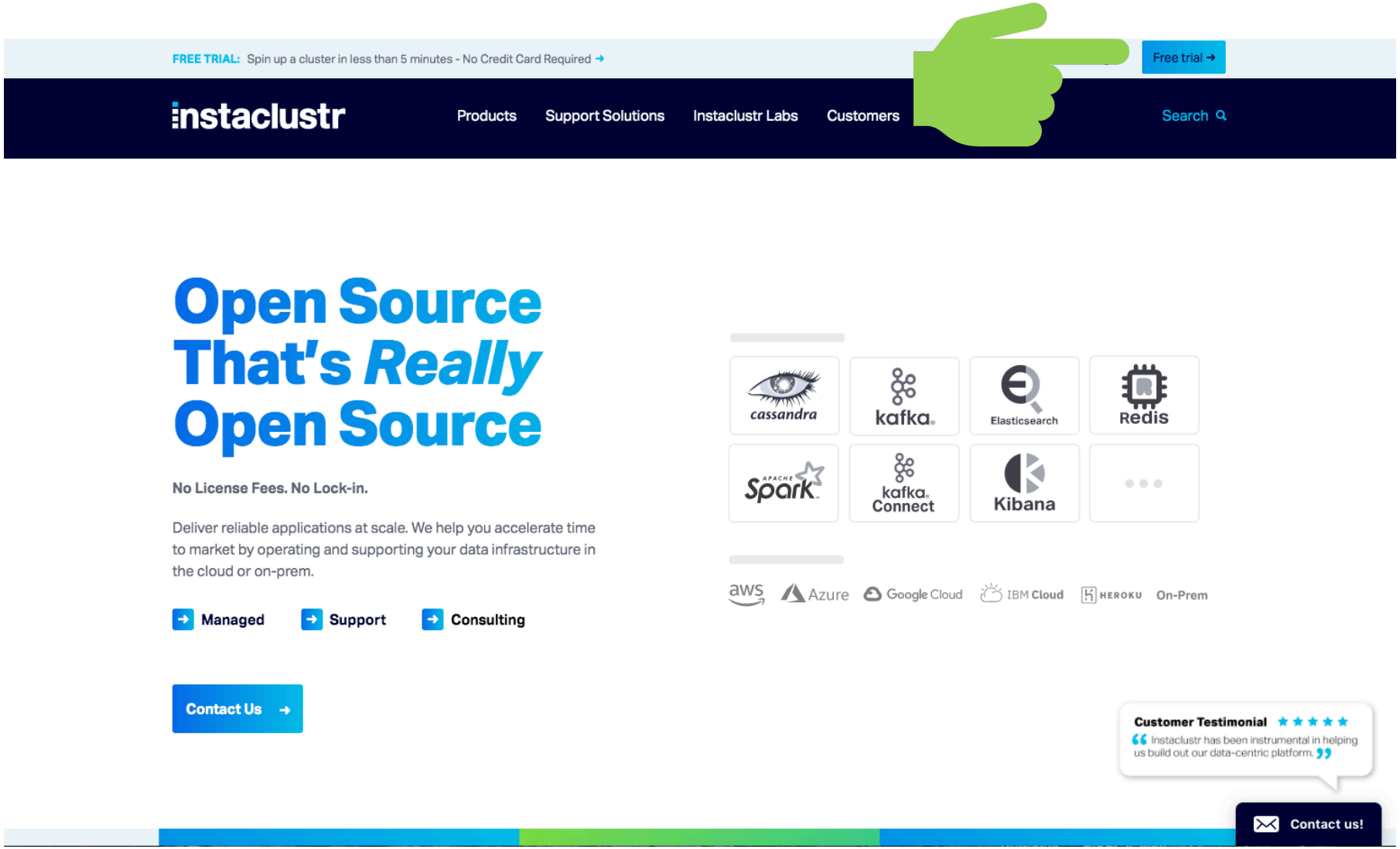
(Source: Shutterstock)

Things to watch out for

instaclustr

- Find a connector (buy, open source, or write your own)
- Configure it
  - configurations are different, trial and error, lots of failures, eventually runs
  - transformation on Sink side may be required
- Test it
  - introduce source, sink and other errors to check if it’s robust enough
- Monitor and Scale it
- Watch out for

  - unexpected failures
  - lag
  - slow sink systems -> more sink tasks -> more partitions -> reduced capacity
  - capacity of sink systems



THANK YOU!

instaclustr

www.instaclustr.com

info@instaclustr.com

@instaclustr

Further information at instaclustr.com/paul-brebner
5 part ”pipeline” blog series and github