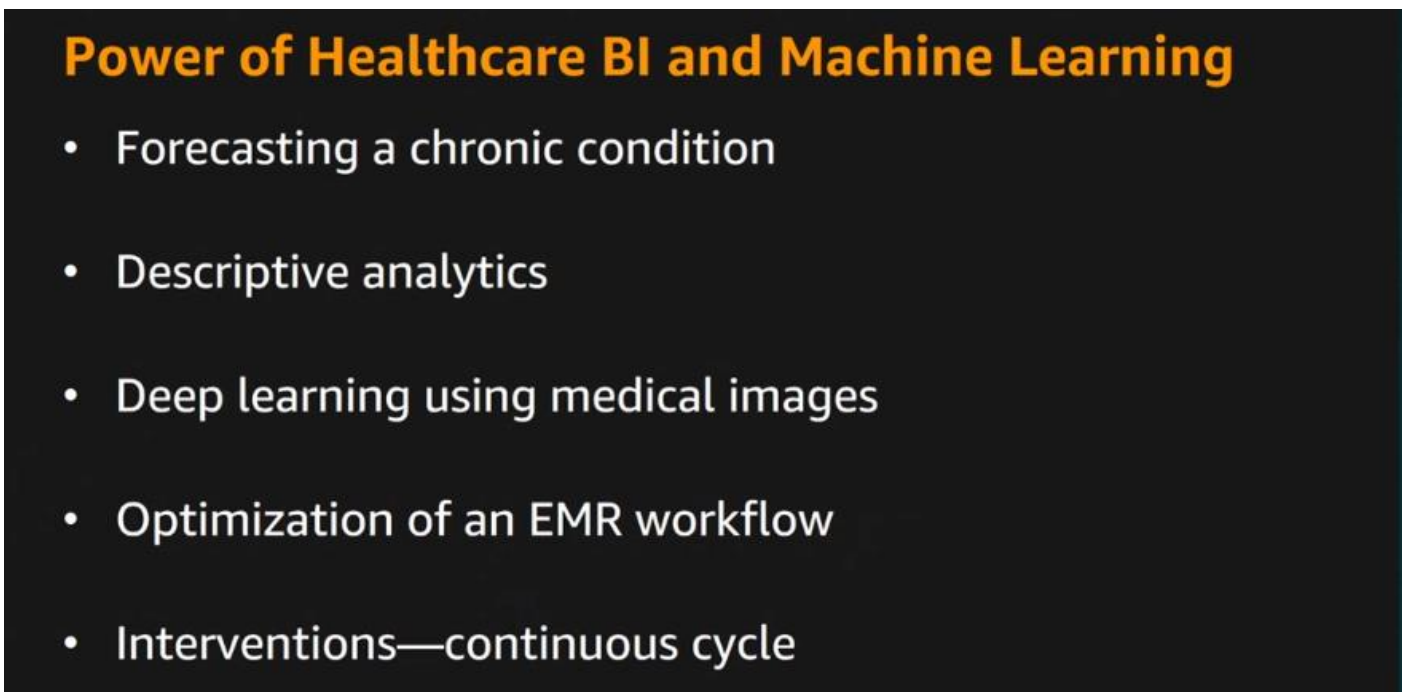
Healthcare data is often large and complex enough to be complete, or simple enough to be usable: choose one. In this session, you will learn how **Cerner** tackled this problem for healthcare and other complex data sets on ***AWS***. Cerner started with patterns to quickly evolve an infrastructure based on AWS to meet new demands, move into data engineering techniques on ***Apache Spark*** to make otherwise unmanageable data sets simple and usable, and connect that through to popular front-end tools for analysts and data scientists like ***Jupyter***. You will also learn how Cerner is helping healthcare informaticians focus more on their analysis and less on undifferentiated heavy data lifting.



Main Challenges

- Infrastructure
- Complex and noisy data
- Usability and adoption

Infrastructure and Data Challenges

- Multiple data sources
- Various data models
- Data transfer rate and network bandwidths
- Elasticity

Usability and Adoption Challenges

- How do I connect?
    - Permissions, drivers, networking, encryption
- How do I share?
- How do I reuse my research?
- How do I find new packages?
- How do I package?

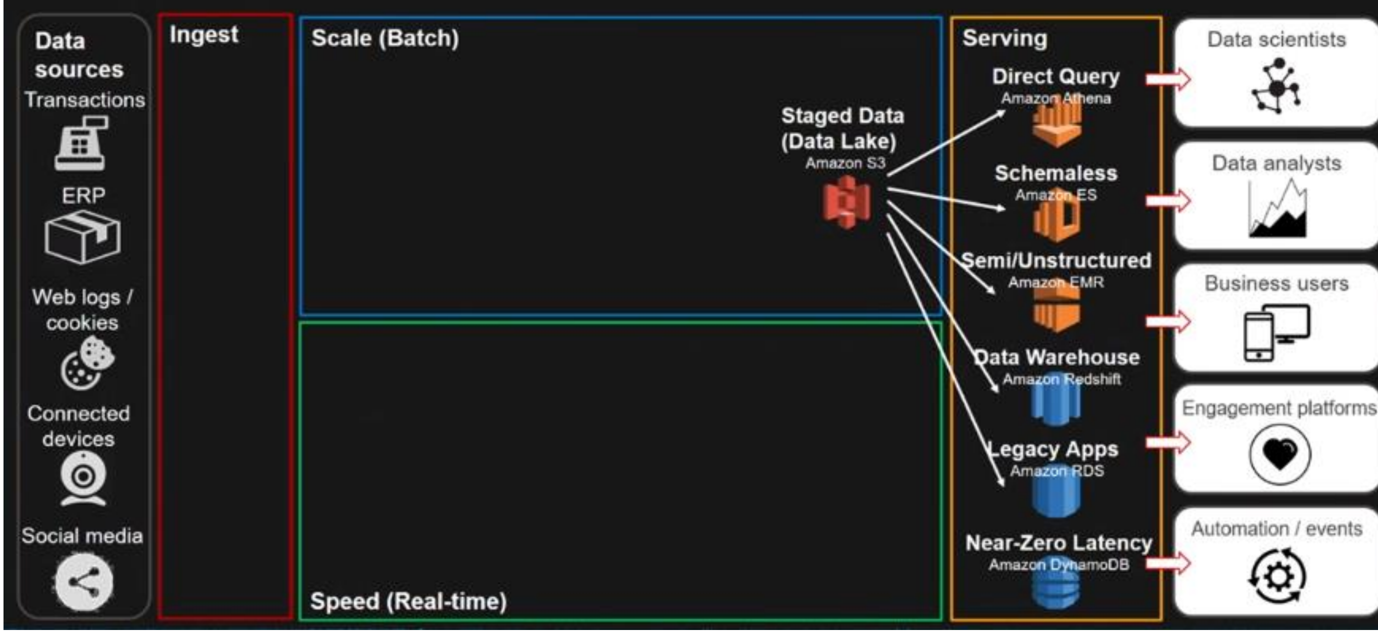
Amazon S3 Data Lake

Amazon S3 as a “data lake”

Trading on-premises HDFS for durability and scalability with Amazon S3 in the cloud

Modern data architecture

Insights to enhance business applications, new digital services



Amazon EMR

Amazon EMR provides a managed Hadoop framework that makes it easy, fast, and cost-effective to process vast amounts of data across dynamically scalable Amazon EC2 instances

Encryption

- Amazon S3—data at rest
- Hadoop
  - HDFS data transfer protocol *(dfs.encrypt.data.transfer)*
  - Hadoop RPC *(hadoop.rpc.protection)*
- MapReduce
  - SSL for encrypted shuffle
- Spark
  - SSL for Akka and HTTP (for broadcast and file server)
  - SASL—Block transfer service

Monitoring

- Amazon S3
  - Bucket access logs
- Amazon EMR
  - Archives various log files to Amazon S3 at 5-minute intervals
  - Log files are available after the cluster terminates
- Amazon CloudWatch Metrics
  - Updated every 5 minutes and archived for 2 weeks

Our Speaker

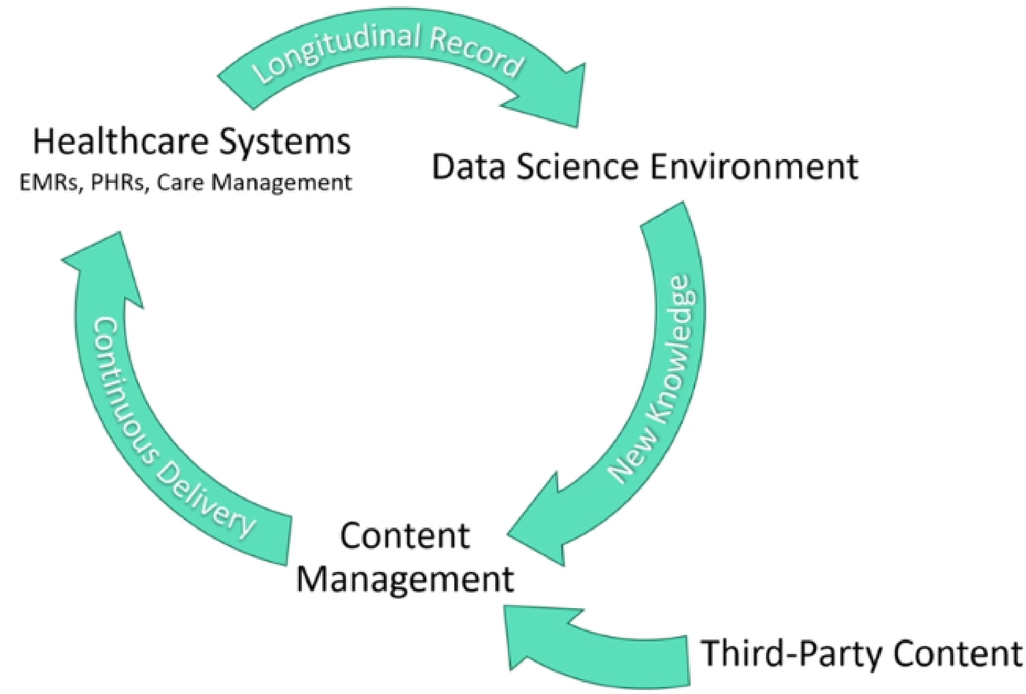
- Ryan Brush, Principal Architect at Cerner
  - Created Clara, an open source rules engine
  - Data engineering, analysis, and the application of very large healthcare datasets
  - Renowned speaker
  - *97 Things Every Programmer Should Know* and *Hadoop: The Definitive Guide*

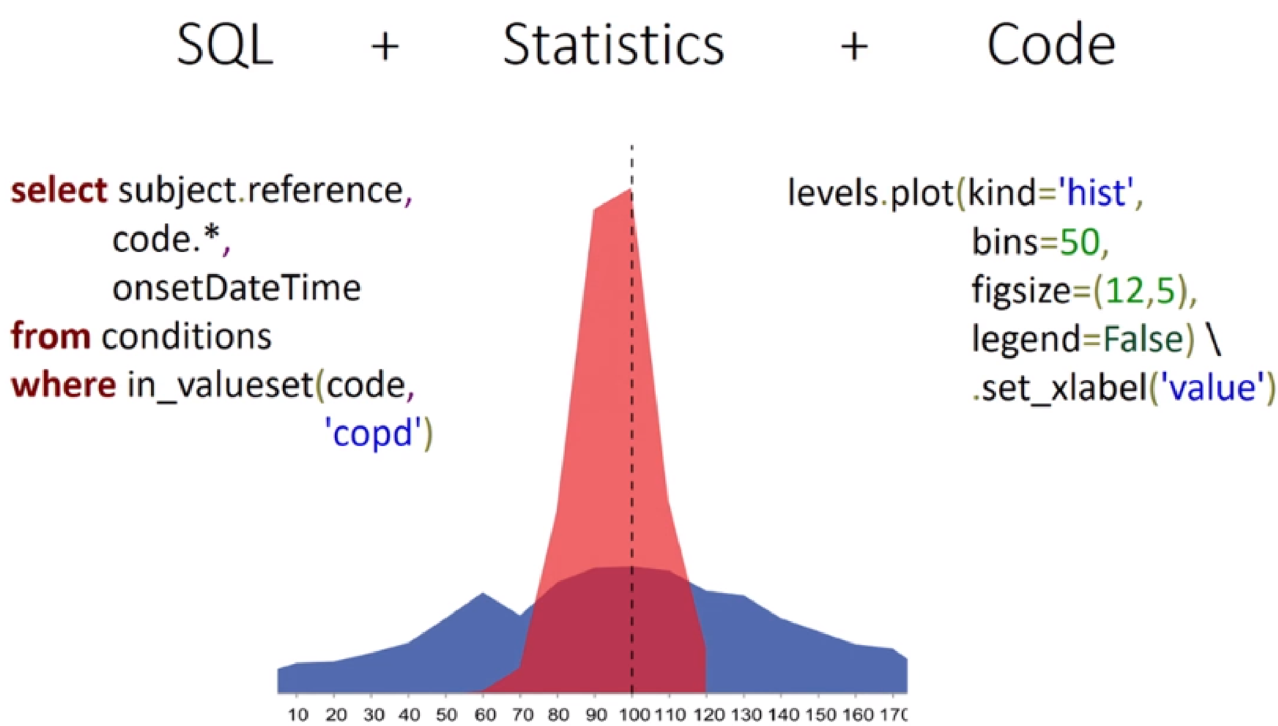
Untangling Healthcare on AWS

Ryan Brush

Cerner Corporation

Cerner is mainly into bringing together a broad set of almost anything that is relevant in healthcare.





We want to bring an SQL view of the data to our data scientists so that they can do what they want

```
glucose_levels = spark.sql("""
select subject.reference,
       code.coding[0].code code,
       valueQuantity.value,
       effectiveDateTime
from observations
where in_valueset(code, 'glucose_level') and
      valueQuantity.value is not null
""")

glucose_levels.limit(10).toPandas()
```

	reference	code	value	effectiveDateTime
0	urn:cerner:empi:a7943425-2420-4403-9fcf-0fae4e...	2345-7	91.0000	2010-11-26T16:00:00Z
1	urn:cerner:empi:30ced248-c37d-485f-92ee-184a09...	2345-7	101.0000	2010-11-29T06:00:00Z
2	urn:cerner:empi:91a843a8-9be1-41df-87df-fae86d...	2345-7	89.0000	2010-09-13T06:00:00Z
3	urn:cerner:empi:1cacfba5-647b-425c-bdd3-70b32a...	2345-7	84.0000	2010-04-05T06:00:00Z
4	urn:cerner:empi:91a31172-5557-47ff-936d-63020e...	2345-7	72.0000	2010-04-08T16:35:00Z
5	urn:cerner:empi:98ad8355-239b-4ce3-b41f-d5baf2...	2345-7	108.0000	2010-11-24T06:00:00Z
6	urn:cerner:empi:edae1c7b-5fb3-4e81-8570-4ff335...	2345-7	82.0000	2010-08-13T19:00:00Z
7	urn:cerner:empi:98ad8355-239b-4ce3-b41f-d5baf2...	2345-7	178.0000	2010-07-27T06:00:00Z
8	urn:cerner:empi:a00f8d5e-942e-4445-bf4c-c6e0ad...	2345-7	68.0000	2010-01-04T20:00:00Z
9	urn:cerner:empi:1ada9444-3f01-45a8-9767-dda8a9...	2345-7	88.0000	2010-04-28T06:00:00Z

- Petabyte scale
- Standard models
- Cataloged
- Collaborative
- Ontology support
- Extensible
- Secure

We give them ***Jupyter notebooks*** to do their jobs with datasets and data catalogs readily available, secured, discoverable as above

So how do we go about building such a system from scratch?

No Silver Bullet
—Essence and Accident in Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

Connectivity

Spark Cluster

Accident

Storage

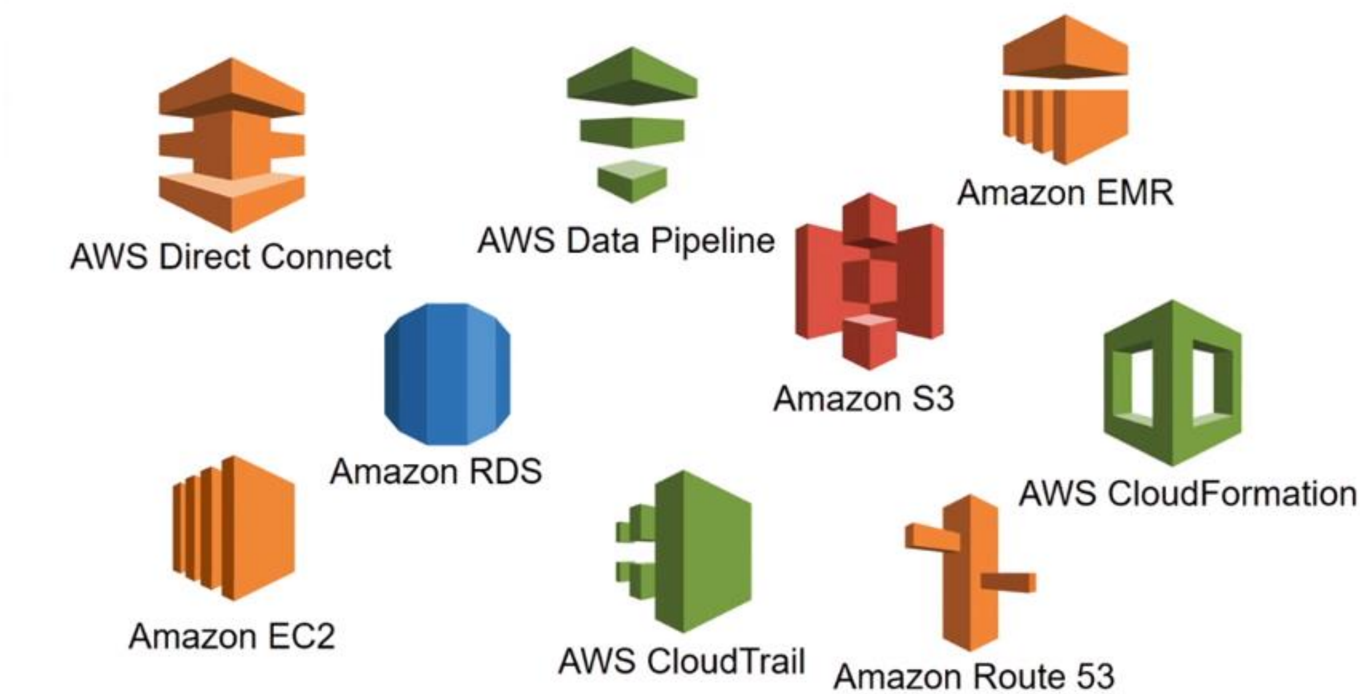
Metadata

Provisioning

Notebook server

Auditing

All of these are important for access to be granted to a user



AWS provides so many building block services that we can design and build our architecture upon.

There is **no theoretical reason** that anything is hard to change about software.

—*Martin Fowler*

Who Needs An Architect?, 2003

Establish Hard Constraints

Optimize for Fast Iteration

Establish Hard Constraints

Security

User Experience

Reproducibility

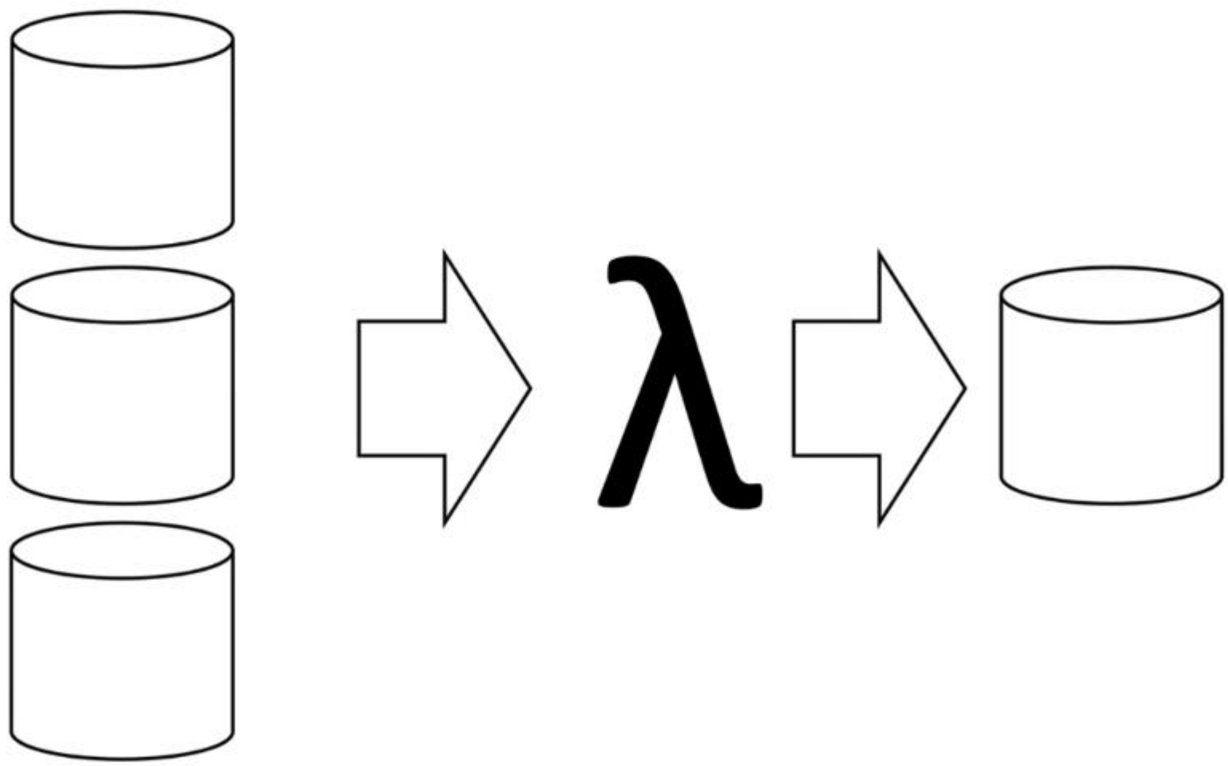
The same input produces the same output every time

```
glucose_levels = spark.sql("""
select subject.reference,
       code.coding[0].code code,
       valueQuantity.value,
       effectiveDateTime
from observations
where in_valueset(code, 'glucose_level') and
      valueQuantity.value is not null
""")

glucose_levels.limit(10).toPandas()
```

	reference	code	value	effectiveDateTime
0	urn:cerner:empi:a7943425-2420-4403-9fcf-0fae4e...	2345-7	91.0000	2010-11-26T16:00:00Z
1	urn:cerner:empi:30ced248-c37d-485f-92ee-184a09...	2345-7	101.0000	2010-11-29T06:00:00Z
2	urn:cerner:empi:91a843a8-9be1-41df-87df-fae86d...	2345-7	89.0000	2010-09-13T06:00:00Z
3	urn:cerner:empi:1cacfba5-647b-425c-bdd3-70b32a...	2345-7	84.0000	2010-04-05T06:00:00Z
4	urn:cerner:empi:91a31172-5557-47ff-936d-63020e...	2345-7	72.0000	2010-04-08T16:35:00Z
5	urn:cerner:empi:98ad8355-239b-4ce3-b41f-d5baf2...	2345-7	108.0000	2010-11-24T06:00:00Z
6	urn:cerner:empi:edae1c7b-5fb3-4e81-8570-4ff335...	2345-7	82.0000	2010-08-13T19:00:00Z
7	urn:cerner:empi:98ad8355-239b-4ce3-b41f-d5baf2...	2345-7	178.0000	2010-07-27T06:00:00Z
8	urn:cerner:empi:a00f8d5e-942e-4445-bf4c-c6e0ad...	2345-7	68.0000	2010-01-04T20:00:00Z
9	urn:cerner:empi:1ada9444-3f01-45a8-9767-dda8a9...	2345-7	88.0000	2010-04-28T06:00:00Z

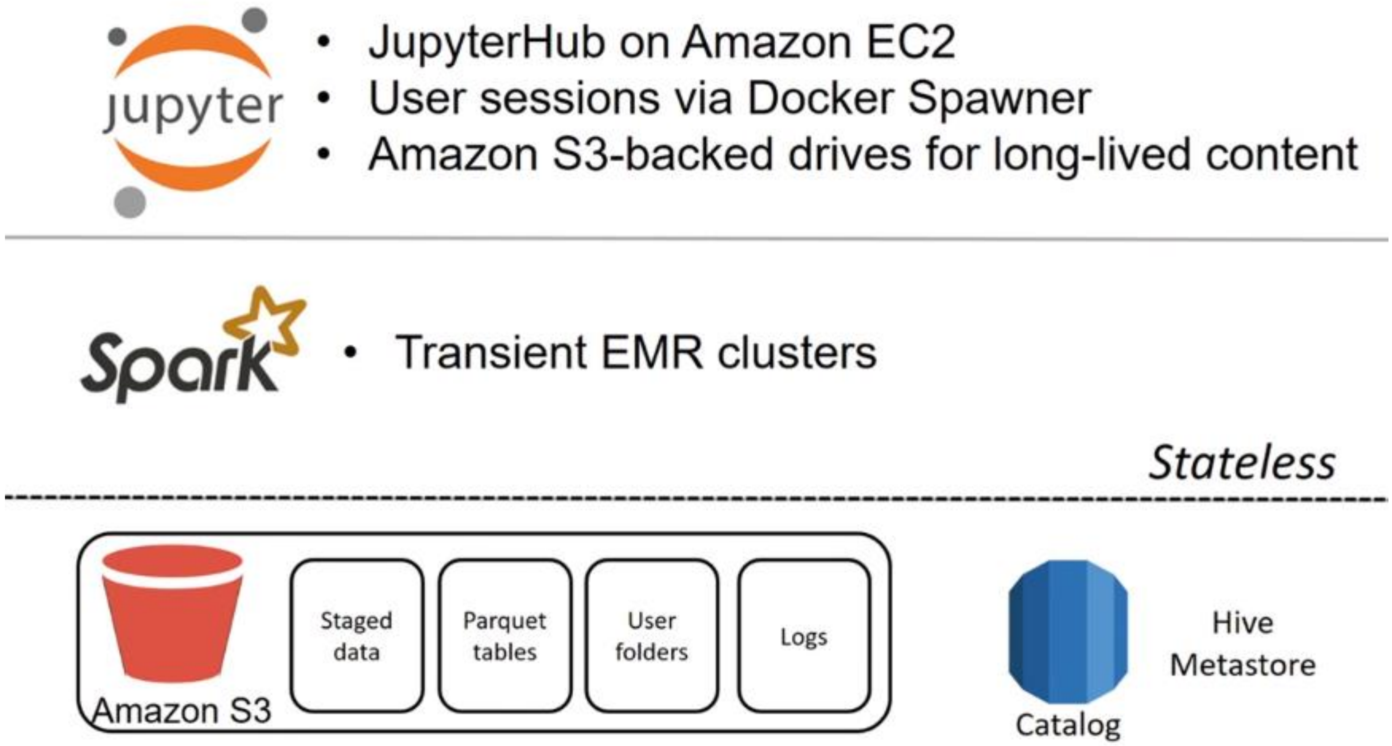
The same query input should return the same result every time.



User Experience

Processing Engine

Storage and Metadata

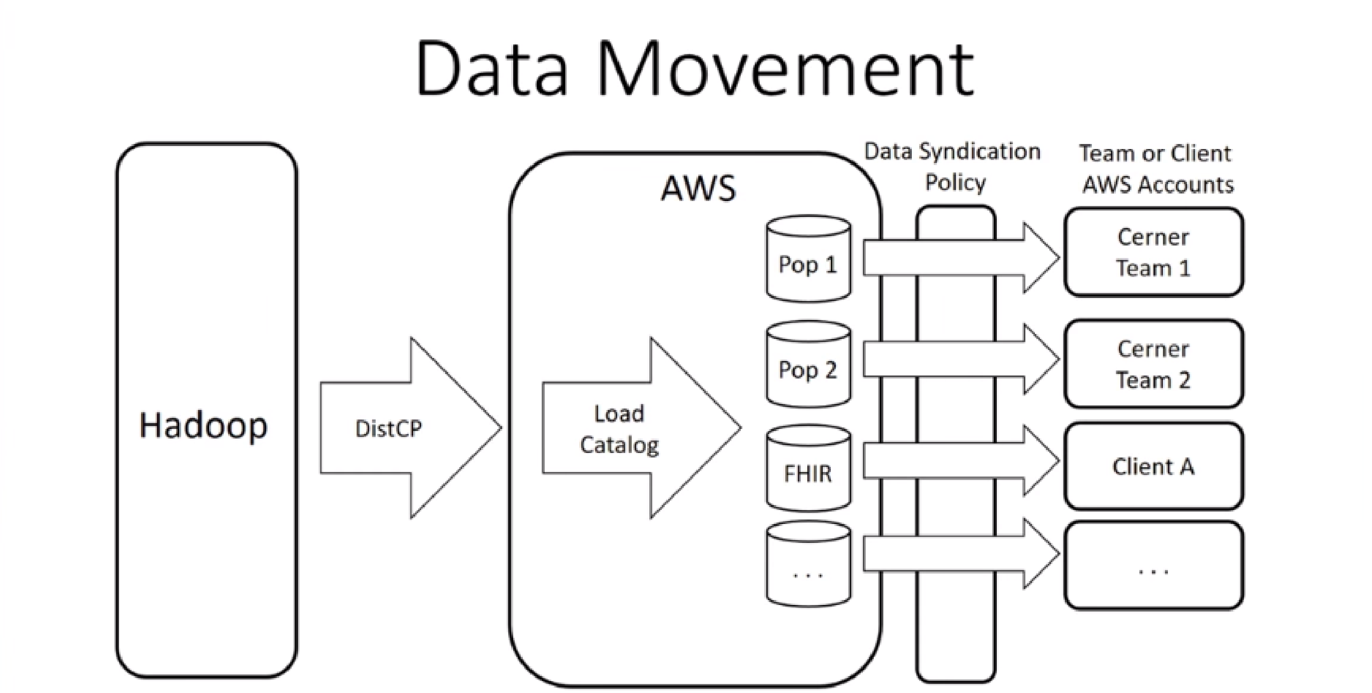


For user experience we use Jupyter notebooks, Spark for the processing engine, and S3 for storing data and Catalog for storing metadata. This is the highest-level view of our architecture. Things below the stateless lines are long lived and anything above is short lived and can be discarded and reproduced from an initial state. We run JupyterHub on EC2 but all the actual user data are stored in a folder in S3 by mapping that S3 drive to the JupyterHub EC2 instance, that is a way to keep things stateless

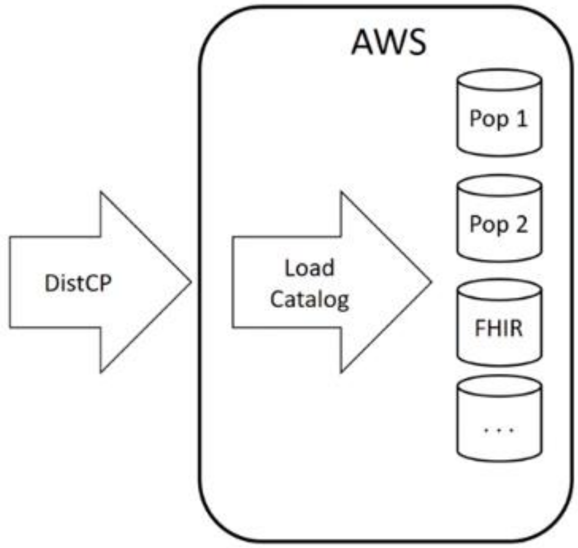
Stateful	Stateless
• State lasts longer than a session	• Can be discarded on an upgrade
• Must consider user and system data	• Fully reproducible from source
• Infrequent updates	• Frequent updates
• Security groups, Amazon Relational Database Service (Amazon RDS), S3	• Everything else

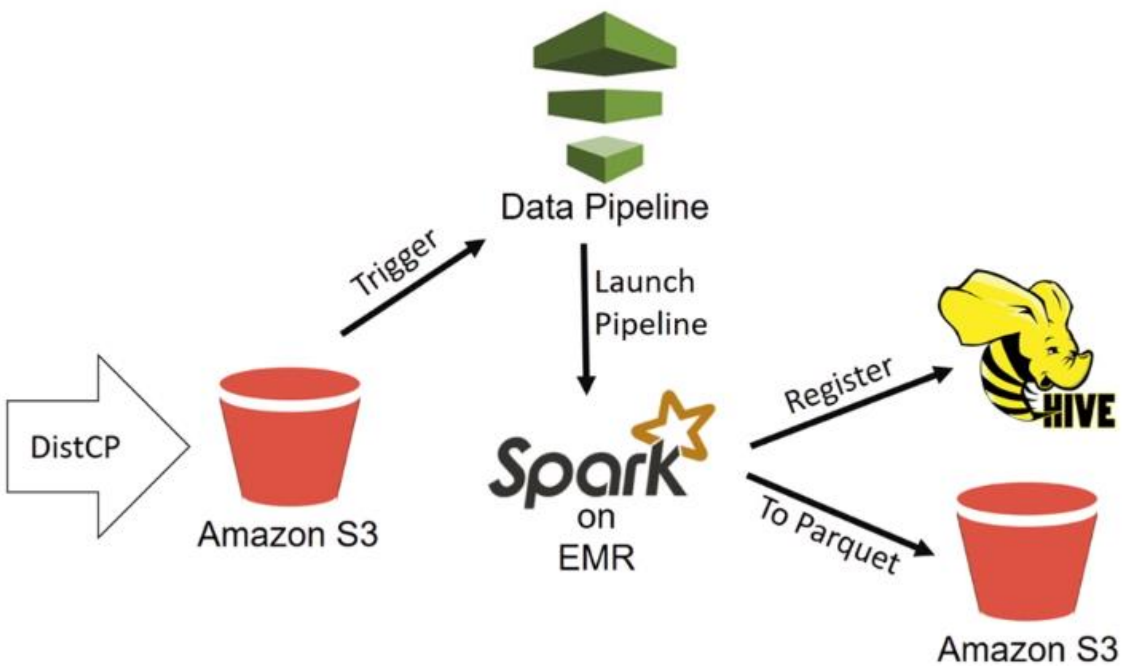
AWS CloudFormation

We extensively use CloudFormation to define a deployment template as APIs for deploying our stacks. We have separate templates for our stateless and stateful stacks.

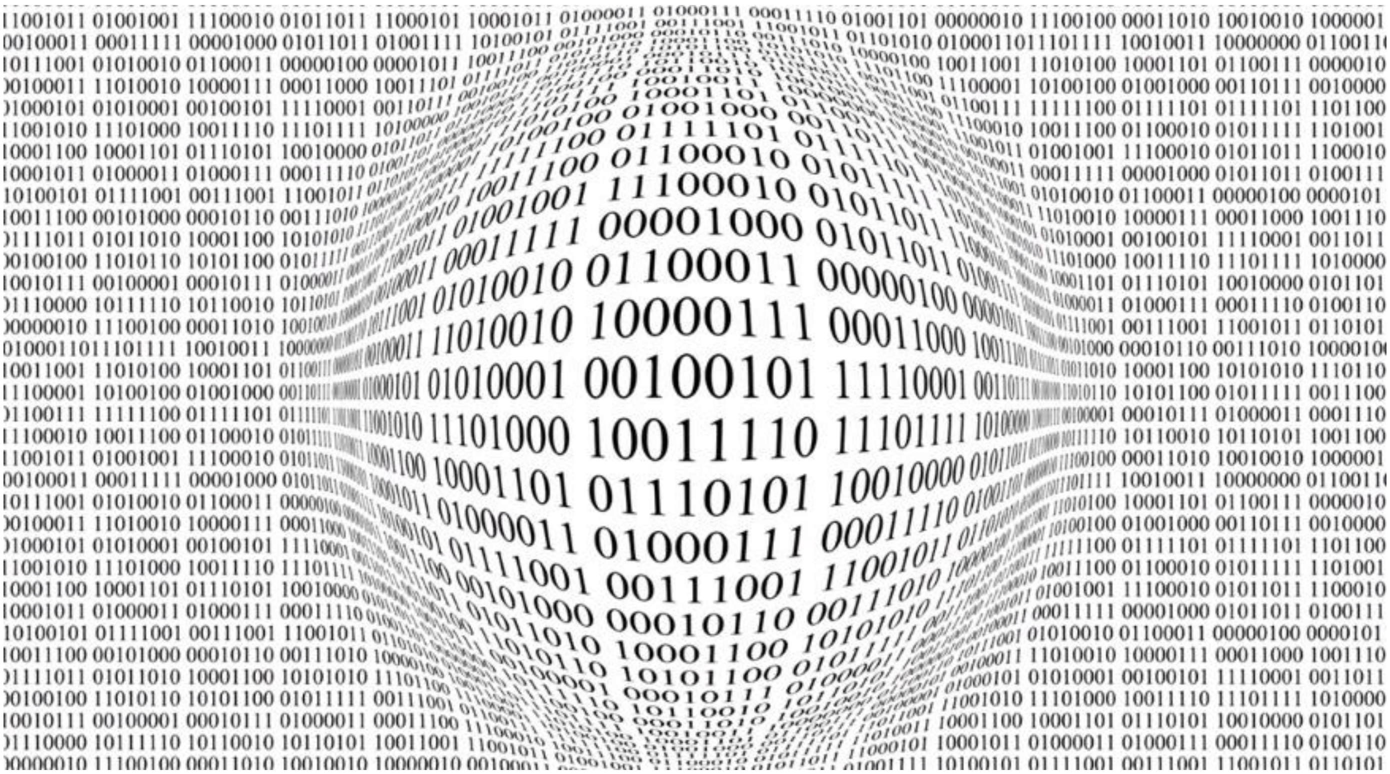


Data Movement





Our users don't know that the data is written in Parquet file format that is a columnar data format, they just discover the data using the data catalog.



Now we need a way to sort and ingest the huge and complex data sources and types in healthcare data

8,000 CPT Codes

72,000 ICD-10 Codes

Different meanings in different contexts

Incomplete, conflicting data sets

Human working memory is five to nine items

All of this must be reconciled across many sources

No common person identifier

Standard data models and codes interpreted inconsistently

63,000 SNOMED disease codes

First, we put all the data into a data catalog that has well defined data schemas for all the types of data that gets into it. Then we have the challenge of making sense of all the data and datasets in the data catalog

```
root
 |-- id: string (nullable = true)
 |-- meta: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |    |-- versionId: string (nullable = true)
 |    |-- lastUpdated: timestamp (nullable = true)
 |    |-- profile: array (nullable = true)
 |    |    |-- element: string (containsNull = true)
 |    |-- security: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- id: string (nullable = true)
 |    |    |    |-- system: string (nullable = true)
 |    |    |    |-- version: string (nullable = true)
 |    |    |    |-- code: string (nullable = true)
 |    |    |    |-- display: string (nullable = true)
 |    |    |    |-- userSelected: boolean (nullable = true)
 |    |-- tag: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- id: string (nullable = true)
 |    |    |    |-- system: string (nullable = true)
 |    |    |    |-- version: string (nullable = true)
 |    |    |    |-- code: string (nullable = true)
 |    |    |    |-- display: string (nullable = true)
 |    |    |    |-- userSelected: boolean (nullable = true)
 |-- implicitRules: string (nullable = true)
 |-- language: string (nullable = true)
 |-- text: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |    |-- status: string (nullable = true)
 |    |-- div: string (nullable = true)
 |-- identifier: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- use: string (nullable = true)
 |    |    |-- type: struct (nullable = true)
 |    |    |    |-- id: string (nullable = true)
 |    |    |    |-- coding: array (nullable = true)
 |    |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |    |-- id: string (nullable = true)
 |    |    |    |    |    |-- system: string (nullable = true)
 |    |    |    |    |    |-- version: string (nullable = true)
 |    |    |    |    |    |-- code: string (nullable = true)
 |    |    |    |    |    |-- display: string (nullable = true)
 |    |    |    |    |    |-- userSelected: boolean (nullable = true)
 |    |    |    |-- text: string (nullable = true)
 |    |    |-- system: string (nullable = true)
 |    |    |-- value: string (nullable = true)
 |    |    |-- period: struct (nullable = true)
```

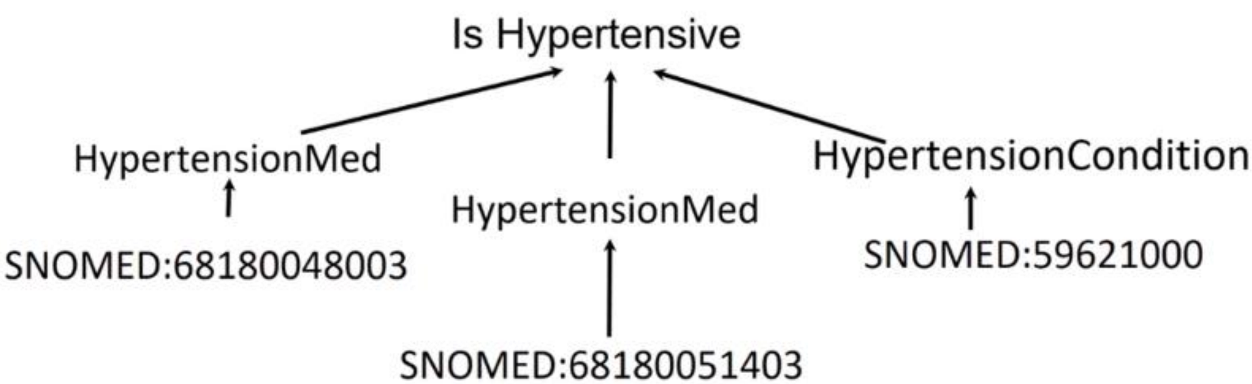
We then build ***FIHR data models*** from the schema of the datasets and data models we have.

```
 |    |    |    |-- id: string (nullable = true)
 |    |    |    |-- start: string (nullable = true)
 |    |    |    |-- end: string (nullable = true)
 |    |    |-- assigner: struct (nullable = true)
 |    |    |    |-- reference: string (nullable = true)
 |    |    |    |-- display: string (nullable = true)
 |-- clinicalStatus: string (nullable = true)
 |-- verificationStatus: string (nullable = true)
 |-- category: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- coding: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- id: string (nullable = true)
 |    |    |    |    |-- system: string (nullable = true)
 |    |    |    |    |-- version: string (nullable = true)
 |    |    |    |    |-- code: string (nullable = true)
 |    |    |    |    |-- display: string (nullable = true)
 |    |    |    |    |-- userSelected: boolean (nullable = true)
 |    |    |-- text: string (nullable = true)
 |-- severity: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |    |-- coding: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- id: string (nullable = true)
 |    |    |    |-- system: string (nullable = true)
 |    |    |    |-- version: string (nullable = true)
 |    |    |    |-- code: string (nullable = true)
 |    |    |    |-- display: string (nullable = true)
 |    |    |    |-- userSelected: boolean (nullable = true)
 |    |-- text: string (nullable = true)
 |-- code: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |    |-- coding: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- id: string (nullable = true)
 |    |    |    |-- system: string (nullable = true)
 |    |    |    |-- version: string (nullable = true)
 |    |    |    |-- code: string (nullable = true)
 |    |    |    |-- display: string (nullable = true)
 |    |    |    |-- userSelected: boolean (nullable = true)
 |    |-- text: string (nullable = true)
 |-- bodySite: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- coding: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- id: string (nullable = true)

 |    |    |    |    |-- system: string (nullable = true)
 |    |    |    |    |-- version: string (nullable = true)
 |    |    |    |    |-- code: string (nullable = true)
 |    |    |    |    |-- display: string (nullable = true)
 |    |    |    |    |-- userSelected: boolean (nullable = true)
 |    |    |-- text: string (nullable = true)
 |-- subject: struct (nullable = true)
 |    |-- reference: string (nullable = true)
 |    |-- display: string (nullable = true)
 |-- context: struct (nullable = true)
 |    |-- reference: string (nullable = true)
 |    |-- display: string (nullable = true)
 |-- onsetDateTime: string (nullable = true)
 |-- onsetRange: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |    |-- low: struct (nullable = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- value: decimal(12,4) (nullable = true)
 |    |    |-- comparator: string (nullable = true)
 |    |    |-- unit: string (nullable = true)
 |    |    |-- system: string (nullable = true)
 |    |    |-- code: string (nullable = true)
 |    |-- high: struct (nullable = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- value: decimal(12,4) (nullable = true)
 |    |    |-- comparator: string (nullable = true)
 |    |    |-- unit: string (nullable = true)
 |    |    |-- system: string (nullable = true)
 |    |    |-- code: string (nullable = true)
 |-- onsetString: string (nullable = true)
 |-- onsetAge: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |    |-- value: decimal(12,4) (nullable = true)
 |    |-- comparator: string (nullable = true)
 |    |-- unit: string (nullable = true)
 |    |-- system: string (nullable = true)
 |    |-- code: string (nullable = true)
 |-- onsetPeriod: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |    |-- start: string (nullable = true)
 |    |-- end: string (nullable = true)
 |-- abatementDateTime: string (nullable = true)
 |-- abatementRange: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |    |-- low: struct (nullable = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- value: decimal(12,4) (nullable = true)
 |    |    |-- comparator: string (nullable = true)
```

```
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- id: string (nullable = true)
 |    |    |    |    |-- coding: array (nullable = true)
 |    |    |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |    |    |-- id: string (nullable = true)
 |    |    |    |    |    |    |-- system: string (nullable = true)
 |    |    |    |    |    |    |-- version: string (nullable = true)
 |    |    |    |    |    |    |-- code: string (nullable = true)
 |    |    |    |    |    |    |-- display: string (nullable = true)
 |    |    |    |    |    |    |-- userSelected: boolean (nullable = true)
 |    |    |    |    |-- text: string (nullable = true)
 |    |    |-- detail: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- reference: string (nullable = true)
 |    |    |    |    |-- display: string (nullable = true)
 |-- note: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- authorString: string (nullable = true)
 |    |    |-- authorReference: struct (nullable = true)
 |    |    |    |-- reference: string (nullable = true)
 |    |    |    |-- display: string (nullable = true)
 |    |    |-- time: string (nullable = true)
 |    |    |-- text: string (nullable = true)
```

Reproducible Projections of Complex Data



A way to work and use the data models is to take a complex data model and just project it onto a model that is simpler to work with using lossy, less data fields that we actually need. The projection is a much simpler dataset that contains the exact dataset we need. This results in a type of giant spreadsheet

data engineering, *n.*
The act of turning the entire world into giant spreadsheet

Rules

```
(defquery is-hypertensive []
  (:or [Condition (in_valueset code 'hypertension')
                  (= clinicalStatus 'active')]
       [MedicationStatement
           (in_valueset medicationCodeableConcept,
                        'hypertension_meds')
           (< 180 (days-since effectiveDateTime))]))
```

We can use a rules-based approach using a ***Rule engine*** for projecting data into the giant spreadsheet that we need.

Clinical Quality Language

```
context Patient

define HypertensiveMeds
          [MedicationStatement: medicationCodeableConcept in
                                 "hypertensive_meds"] M
          where M.effectiveDateTime > Today() - 180 days

define HypertensionCond [Condition: code in "hypertension"] C
                         where C.clinicalStatus = 'active'

define isHypertensive: exists (HypertensionCond OR
                                HypertensiveMeds)
```

We can also use the ***Quality query language*** that is finer tuned

SQL

```
select subject.reference reference
from conditions
where in_valueset(code, 'hypertension') and
      clinicalStatus == 'active'
union
select subject.reference reference
from medicationstatements
where in_valueset(medicationCodeableConcept,
                  'hypertension_meds') and
      datediff(current_timestamp(), effectivedatetime) < 180
```

We can just use ***SQL*** to create the projection also.

SQL

```
spark.sql(“””
select subject.reference reference
from conditions
where in_valueset(code, 'hypertension') and
      clinicalStatus == 'active'
union
select subject.reference reference
from medicationstatements
where in_valueset(medicationCodeableConcept,
                  'hypertension_meds') and
      datediff(effectivedatetime, current_timestamp()) < 180
“””).distinct() \
    .registerTempTable(‘hypertensive_patients’)
```

SQL queries are easily turned into ***Spark SQL*** that can then be run on the data within our cluster

```
spark.sql("""
select * from hypertensive_patients
""").count()
```

```
44719
```

```
spark.sql("""
select distinct(hp.reference)
from hypertensive_patients hp,
     conditions c
where in_valueset(c.code, 'diabetes') and
      hp.reference = c.subject.reference
""").count()
```

```
16317
```

Once we have our giant spreadsheet projections, we can use queries like above from our Jupyter notebooks to do some analysis quickly with a few lines of code. ***We create projections based on the use case to get the data from much larger and complex datasets, then let your data scientists manipulate them interactively from their Jupyter notebooks.***

```
glucose_levels = spark.sql("""
select valueQuantity.value,
       effectiveDateTime
from observations
where in_valueset(code, 'glucose_level') and
      valueQuantity.value is not null""")
```

```
glucose_levels.limit(10).toPandas()
```

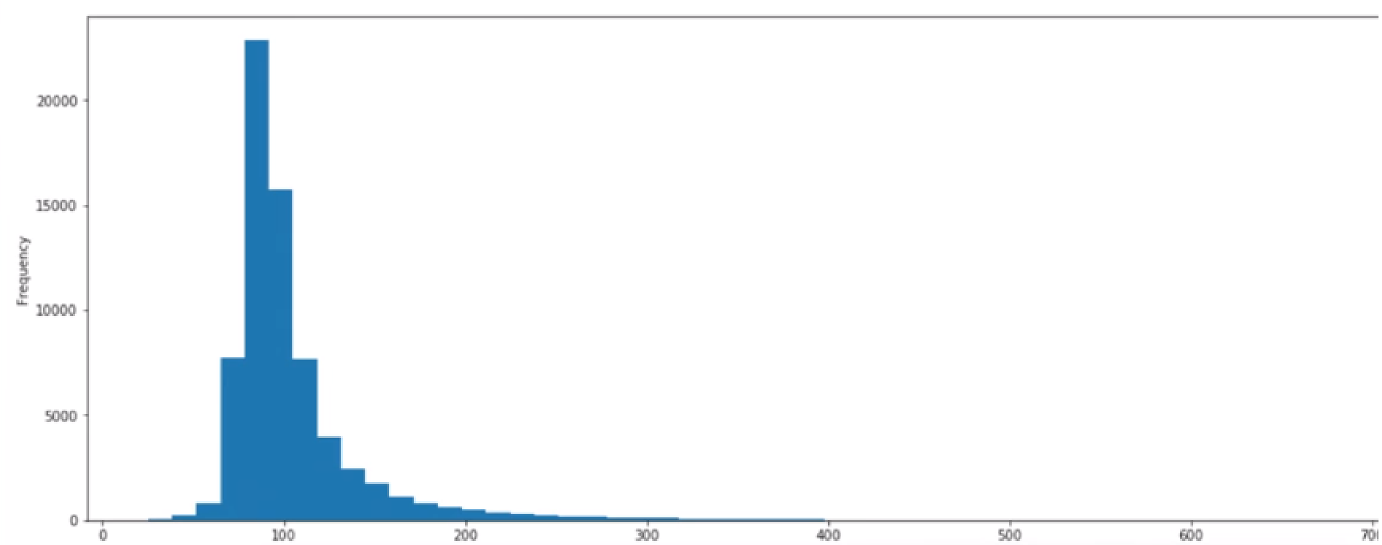
	value	effectiveDateTime
0	91.0000	2010-11-26T16:00:00Z
1	101.0000	2010-11-29T06:00:00Z
2	89.0000	2010-09-13T06:00:00Z
3	84.0000	2010-04-05T06:00:00Z

```
glucose_levels = spark.sql("""
select valueQuantity.value,
       effectiveDateTime
from observations
where in_valueset(code, 'glucose_level') and
      valueQuantity.value is not null""")
```

```
glucose_levels.describe().toPandas()
```

	summary	value	effectiveDateTime
0	count	67955	67955
1	mean	104.08580678	None
2	stddev	38.43865644934375	None
3	min	25.0000	2009-01-04T15:46:00Z
4	max	691.0000	2010-12-31T16:10:00Z

```
glucose_levels.select(col('value').cast('float')) \
              .toPandas() \
              .plot(kind='hist',
                    bins=50,
                    figsize=(18,7),
                    legend=False) \
             .set_xlabel("value");
```




```
in_valueset(code, 'hypertension')
```

User-defined-functions are very helpful helper functions written in a JVM language like Java that we use within our code to work on our datasets, we then express this in simple, fast SQL queries

Spark SQL
+
User-Defined Functions
+
Broadcast Reference Data

```
hypertension_meds = \
  [('http://snomed.info/sct', '68180051403'), # Lisinopril
   (' http://snomed.info/sct ', '68180048003')] # Simvastatin
```

```
push_valuesets(spark,
               {'hypertension'  : isa_snomed('59621000'),
                'glucose_level' : isa_loinc('2345-7'),
                'hypertension_meds' : hypertension_meds})
```

```
spark.sql("""select count(*) from conditions
             where in_valueset(code, 'hypertension')""")
```

`in_valueset`

`extract_terms`

`call_rules`

```
spark.sql("""
select count(*)
from conditions
where in_valueset(code, 'hypertension')""")
```

300 Million Encounters
8-Node EMR Cluster
~4 Seconds

Applying the Pattern

```
patients = spark.sql("""
select id patient_id,
       gender,
       birthDate
from patients""")
```

We are again following the pattern of creating giant spreadsheets to be stored as projections in our data catalog and then comprising them together like about when doing a query using standard SQL.

```
observations = spark.sql("""
select subject.reference patient_id



                    avg_glucose_level,




from observations
group by subject.reference

                                  """)
```

```
observations = spark.sql("""
select subject.reference patient_id


       avg(if(in_valueset(code, 'glucose_level'),
              valueQuantity.value,
              null)) avg_glucose_level,



from observations
group by subject.reference

                                     """)


observations = spark.sql("""
select subject.reference patient_id


       avg(if(in_valueset(code, 'glucose_level'),
              valueQuantity.value,
              null)) avg_glucose_level,



from observations
group by subject.reference,
         year(effectiveDateTime),
         month(effectiveDateTime)""")
```

```
observations = spark.sql("""
select subject.reference patient_id,
       year(effectiveDateTime) year,
       month(effectiveDateTime) month
       avg(if(in_valueset(code, 'glucose_level'),
              valueQuantity.value,
              null)) avg_glucose_level,



from observations
group by subject.reference,
         year(effectiveDateTime),
         month(effectiveDateTime)""")
```

```
observations = spark.sql("""
select subject.reference patient_id,
       year(effectiveDateTime) year,
       month(effectiveDateTime) month
       avg(if(in_valueset(code, 'glucose_level'),
              valueQuantity.value,
              null)) avg_glucose_level,
       avg(if(in_valueset(code, 'bun'),
              valueQuantity.value,
              null)) avg_bun
from observations
group by subject.reference,
         year(effectiveDateTime),
         month(effectiveDateTime)""")
```

```
conditions = spark.sql("""
select subject.reference patient_id,
       year(onsetDateTime) year,
       month(onsetDateTime) month,
       max(if(in_valueset(code, 'hypertension'),
              true,
              false)) hypertension,
       max(if(in_valueset(code, 'chest_pain'),
              true,
              false)) chest_pain
from conditions
group by subject.reference,
         year(onsetDateTime),
         month(onsetDateTime)""")
```

```
ts = patients.join(observations, 'patient_id') \
             .join(conditions,
                   ['patient_id', 'year', 'month']) \
             .orderBy('patient_id', 'year', 'month')
```

patient_id	year	month	gender	birthDate	avg_glucose_level	avg_bun	hypertension
a4b93a-e1b1-45dd-8885-e353d8...	2010	1	female	1973-01-01	None	None	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	2	female	1973-01-01	95.00000000	6.00000000	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	3	female	1973-01-01	None	None	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	4	female	1973-01-01	None	None	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	6	female	1973-01-01	84.00000000	8.00000000	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	7	female	1973-01-01	98.00000000	8.00000000	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	8	female	1973-01-01	81.00000000	10.00000000	False

We then simply join the resulting 2 tables together to get the view above that we can feed directly into our machine learning use case and models of other use cases.

```
# Save for future reference
ts.write.saveAsTable('patient_history')
```

In the collaborative space where we can save our time series data as a table of all our patient history information, and then we can pick it up later from there and use.

```
# Save for future reference
ts.write.saveAsTable('patient_history')
```

Data Engineering

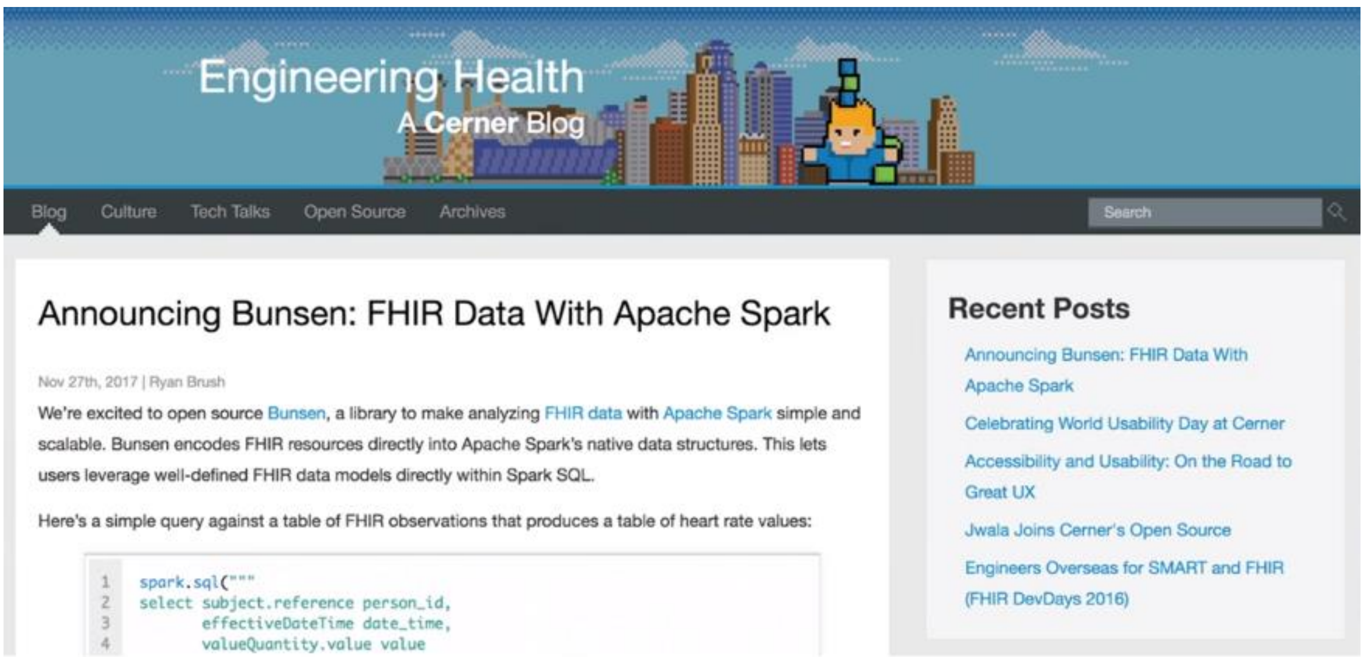
Machine Learning

```
# Build the Spark ML pipeline
pipeline = Pipeline(...)

(train,test) = spark.sql(
    'select * from patient_history') \
    .randomSplit([0.6, 0.4])

model = pipeline.fit(train)
```

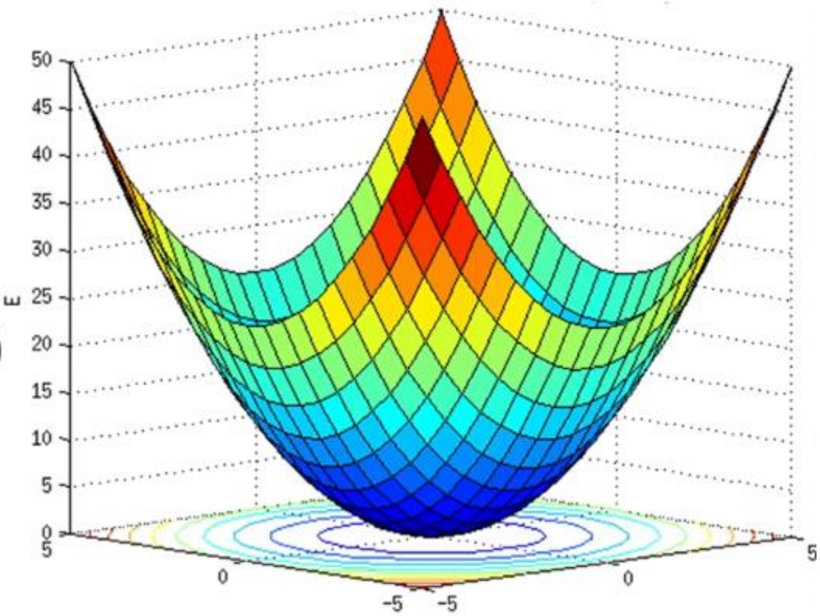
Having the data in a tabular form allows us to use Spark's in-built ML capabilities or other ML systems to tap into the data and do analysis. We can take our high-quality data, put it into a Pipeline, and build our ML model for our problem.



http://engineering.cerner.com/bunsen

This ***Bunsen*** OSS project takes all our ***FIHR data*** and natively represent it in the Apache Spark project.

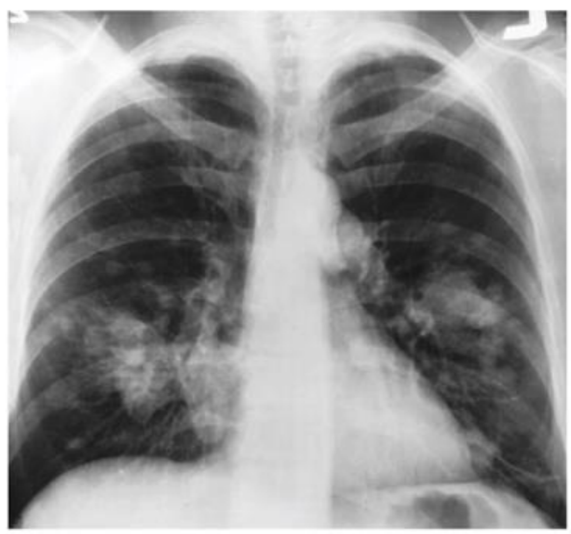
Make the Machine Learning Job Simpler



https://commons.wikimedia.org/wiki/File:Error_surface_of_a_linear_neuron_with_two_input_weights.png

Function Approximation

albumin level > 5.5



Function Approximation

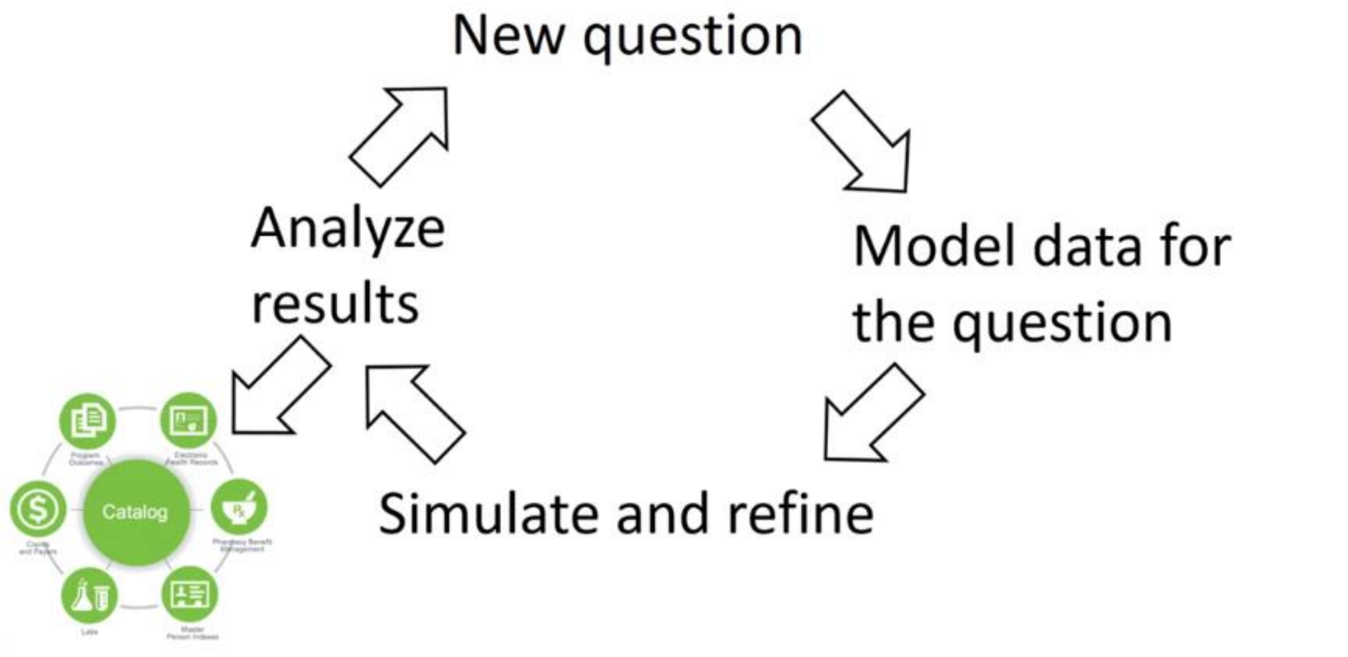
```
 |    |    |    |    |-- display: string (nullable = true)
 |    |    |    |-- userSelected: boolean (nullable = true)
 |    |-- text: string (nullable = true)
 |-- code: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |    |-- coding: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- id: string (nullable = true)
 |    |    |    |-- system: string (nullable = true)
 |    |    |    |-- version: string (nullable = true)
 |    |    |    |-- code: string (nullable = true)
 |    |    |    |-- display: string (nullable = true)
 |    |    |    |-- userSelected: boolean (nullable = true)
 |    |-- text: string (nullable = true)
 |-- bodySite: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- id: string (nullable = true)
 |    |    |-- coding: array (nullable = true)
 |    |    |    |-- element: struct (containsNull = true)
 |    |    |    |    |-- id: string (nullable = true)
```

albumin level > 5.5

Reproducibility Mitigates Complexity

```
ts = patients.join(observations, 'patient_id') \
           .join(conditions,
                 ['patient_id', 'year', 'month']) \
           .orderBy('patient_id', 'year', 'month')
```

patient_id	year	month	gender	birthDate	avg_glucose_level	avg_bun	hypertension
a4b93a-e1b1-45dd-8885-e353d8...	2010	1	female	1973-01-01	None	None	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	2	female	1973-01-01	95.00000000	6.00000000	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	3	female	1973-01-01	None	None	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	4	female	1973-01-01	None	None	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	6	female	1973-01-01	84.00000000	8.00000000	False
a4b93a-e1b1-45dd-8885-e353d8...	2010	7	female	1973-01-01	98.00000000	8.00000000	False



What is exciting about this cycle is that we can now do it on-demand and at scale very fast.