Scaling to a million ML models using Kubernetes, Apache Spark and Apache Arrow

Sandeep Pombra
Head of Data Science

Jakub Pavlik
Director Engineering

Volterra

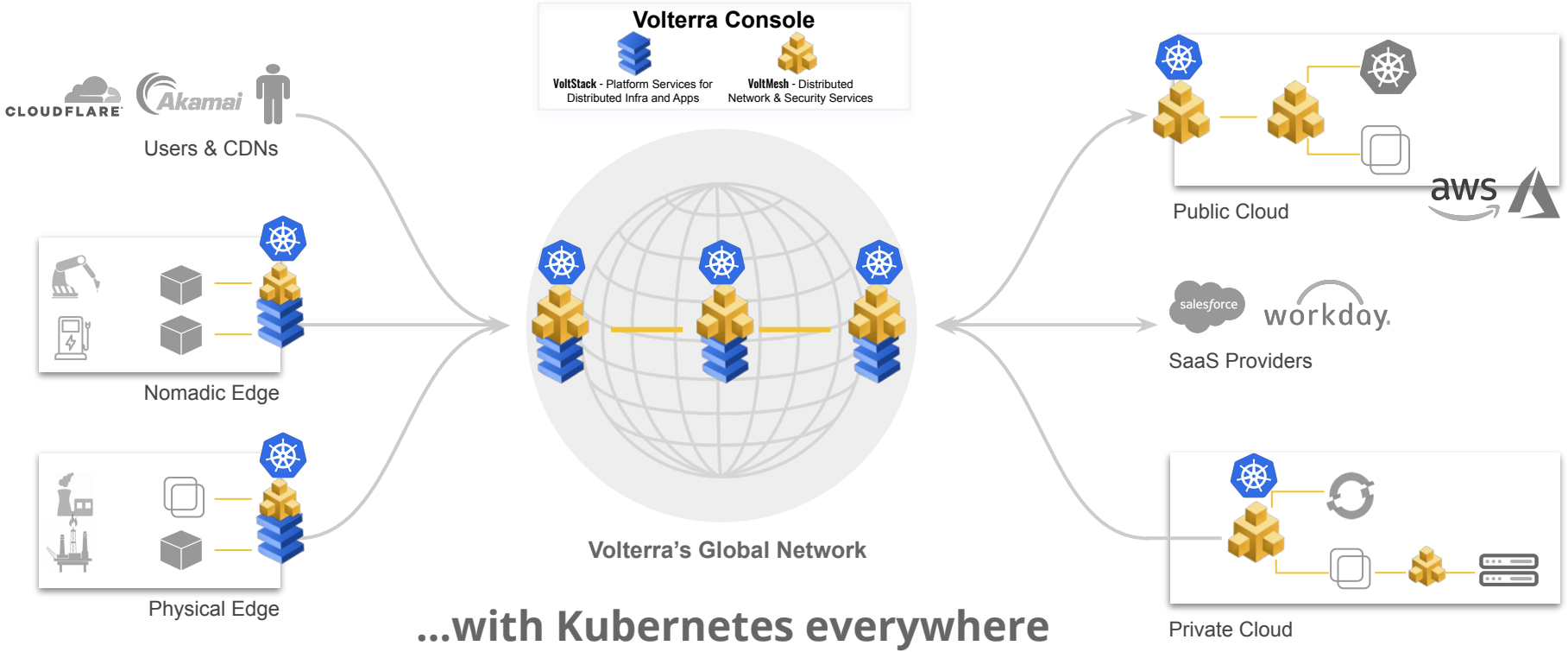
Enable a Distributed Cloud for Everyone, Everywhere

Agenda

- Overview
- Volterra ML Functions & Model Explosion
- ML Infrastructure Evolution Journey
- Model Scaling Challenges

WE BUILT VOLTERRA

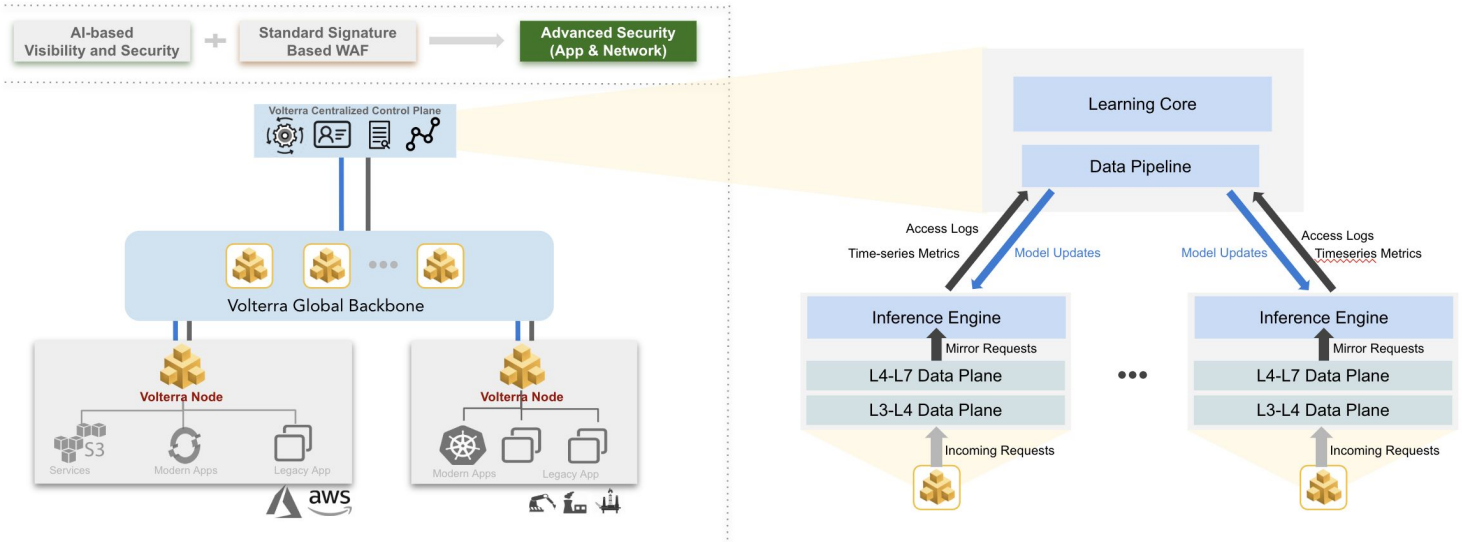
...to distribute cloud services wherever your apps & data need them



...with Kubernetes everywhere

Volterra

VoltMesh - Application Security

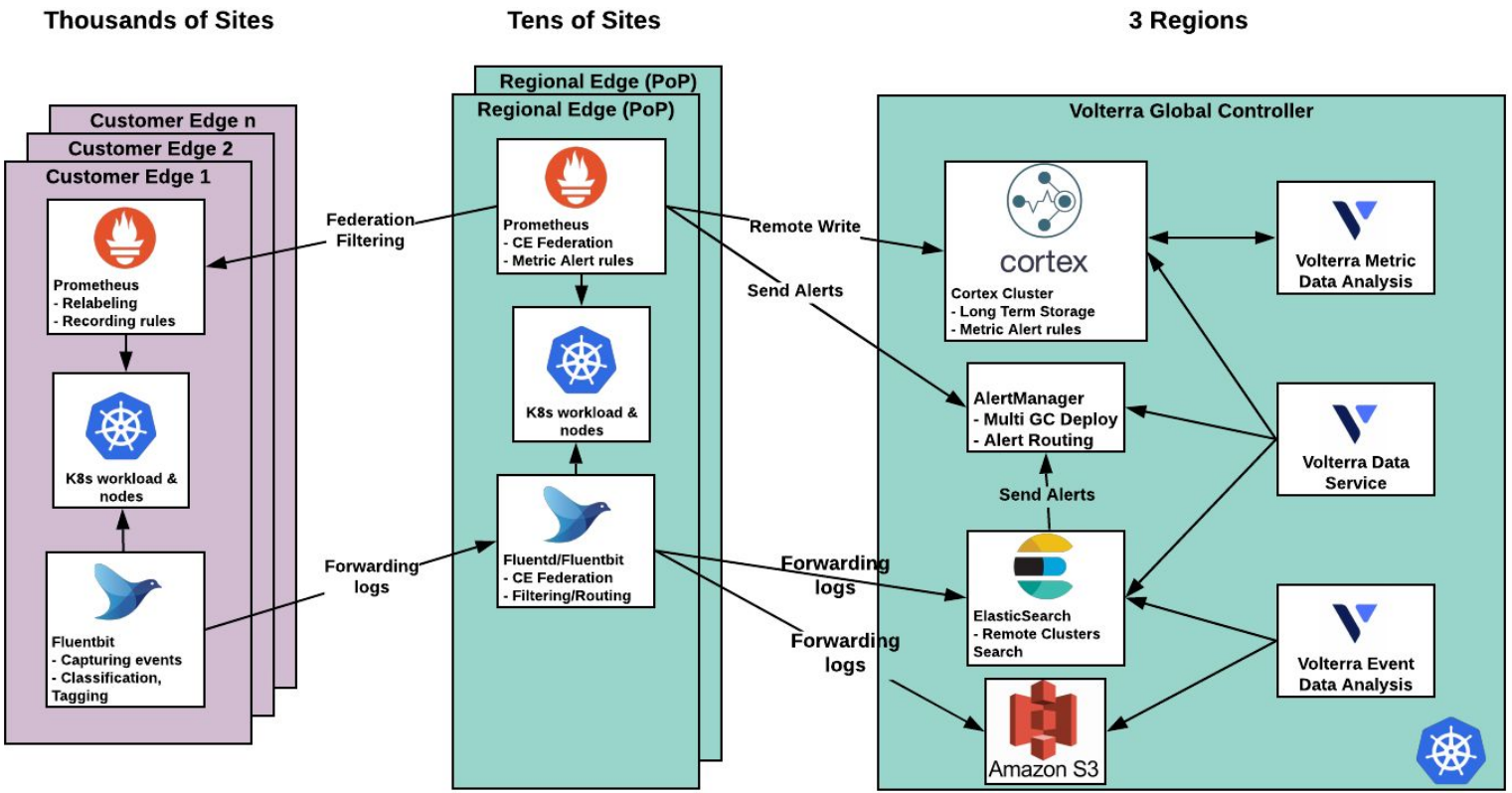


VoltMesh - Application Security

- Rule-based WAFs
- Application DoS & BOT Detection
- Application Anomaly Detection
- Time-series Anomaly Detection
- API Endpoint Markup
- API Behavioral Analysis & Anomalies

Volterra

Volterra Metrics and Logging infrastructure

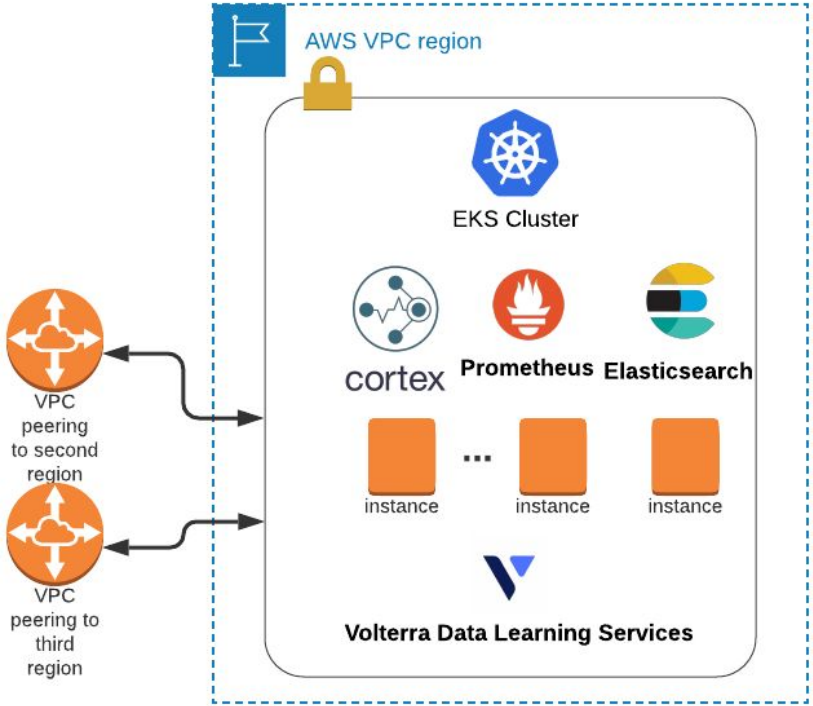


Volterra ML Model Explosion and Scaling

- Volterra ML models has to be done across several dimensions: Application, API, Virtual Host, Source, Destination etc
- This leads to large cardinality of models for each ML module
- We need to scale as we deploy more complex applications in several locations
- Our ML models was running on a single instance in a serialized manner; one after the other
- The training of all models simultaneously took very long. This alone can cause a scenario that once a model was ready and deployed it was already obsolete

Where We Started

- Single Kubernetes cluster with continuous static running learning jobs
- Inefficient CPU and RAM usage
- Resizing to bigger and bigger VM flavors
- Managing scaling to multiple instances
- Autoscaling processing and memory
- Handling very large dataset ingestion and training



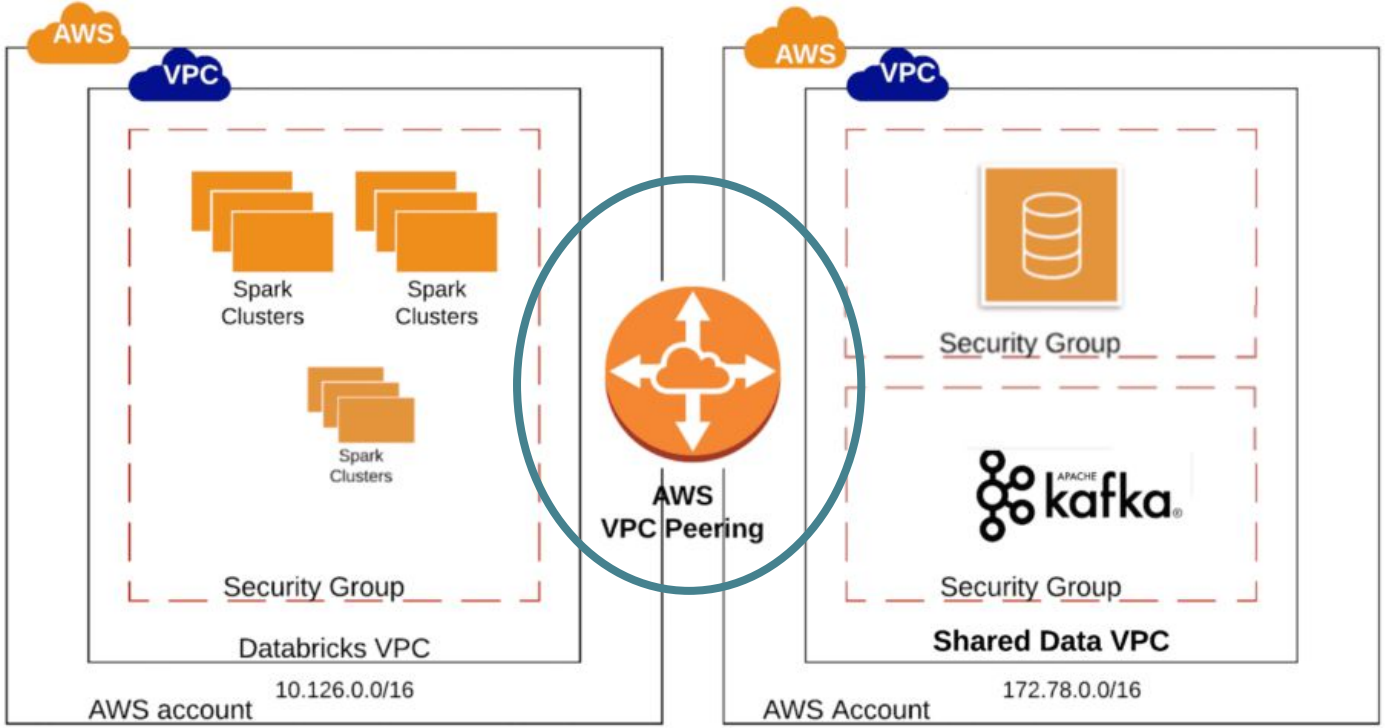
Volterra

What Would Be an Ideal Solution?

- Run several models in parallel with high performance
- Autoscale with increasing number of applications. Seamlessly handle varying levels of scale per customer.
- Optimize CPU Resources, Cost effective
- Automation with CI/CD, Minimize Infrastructure Management
- Secure and Seamless Data Ingestion

⇒ End-to-end integration of existing Kubernetes infra with Spark Scaling and Parallelization (Databricks)

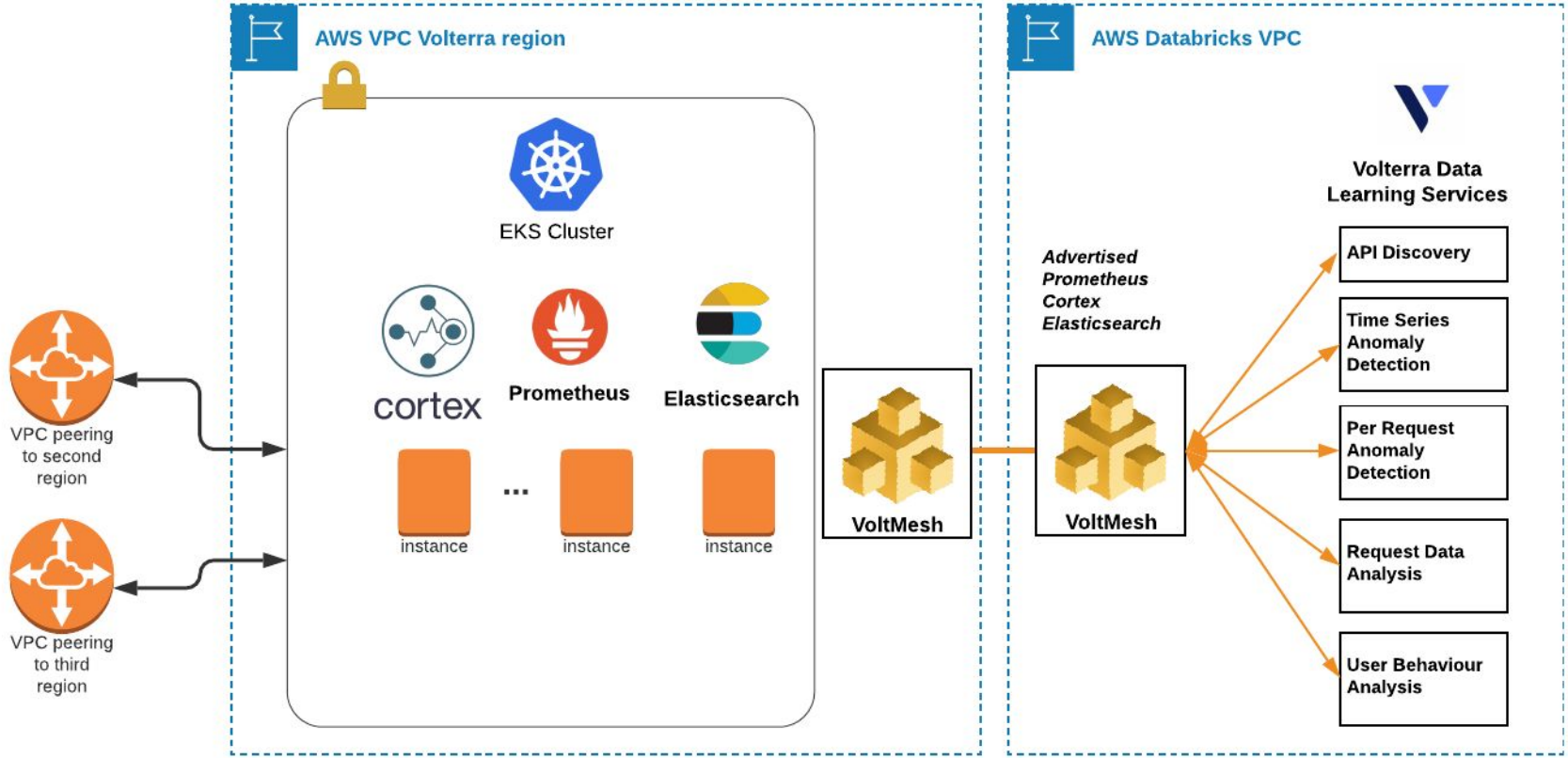
Standard Databricks Integration



Databricks requires VPC Peering, which opens new security concerns…

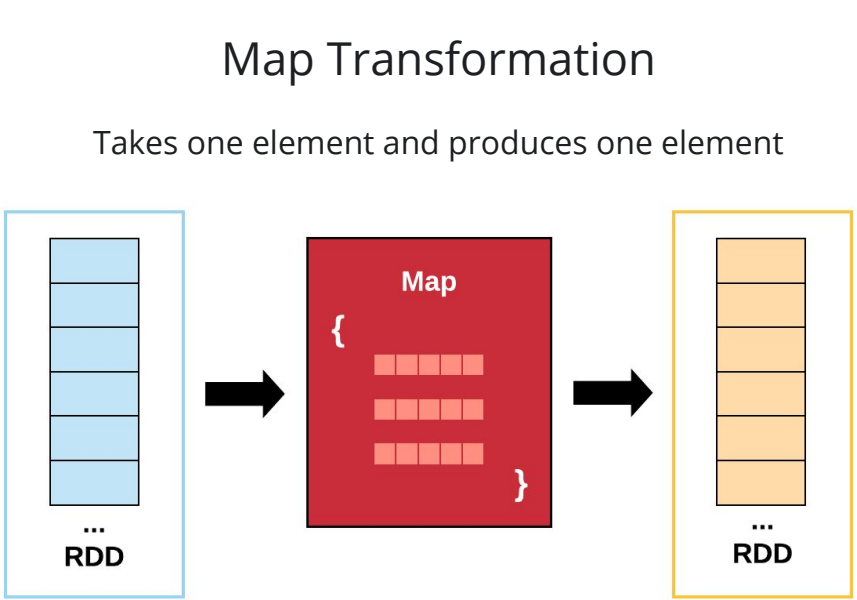
Volterra

Final Design Using VoltMesh Secure Ingress GW



Model Parallelization Approach 1

Create Spark Dataframe of Model Keys and use *map/collect* to apply the Model Function in Parallel



Spark provides a *resilient distributed dataset* (RDD), that can be operated on in parallel
RDDs transformations: *map*
RDD action: *collect*

```
def outer_func(key_tuples, var1, var2):
  # Build Pandas Dataframe from keys
  pd_df = pd.DataFrame.from_records(key_tuples)
  # Build the Spark Dataframe from Pandas
  spark_df = spark.createDataFrame(pd_df)
  # Model Function inside an Outer Function
  def my_func(pd_frame):
    key1 = pd_frame['key1']
    key2 = pd_frame['key2']
    return model_func(key1, key2, var1, var2))
  results = spark_df.rdd.map(my_func).collect()
```

Volterra

Model Parallelization Approach 2

Use Pandas UDFs with Apache Arrow

Pandas UDFs allow not only to scale out their workloads, but also to leverage the Pandas APIs in Apache Spark.

The user-defined functions are executed by:

- Apache Arrow, to exchange data directly between JVM and Python driver/executors with near-zero (de)serialization cost.
- The Pandas UDFs work with Pandas APIs inside the function and Apache Arrow for exchanging data.
- It allows vectorized operations that can increase performance up to 100x, compared to row-at-a-time Python UDFs.
- Grouped Map Pandas UDFs: split a Spark DataFrame into groups based on the conditions specified in the group by operator, applies a UDF (pandas.DataFrame > pandas.DataFrame) to each group, combines and returns the results as a new Spark DataFrame.

Apache Arrow

Being columnar in memory, Apache Arrow manages memory more efficiently than row storage and takes advantage of modern CPU and GPU.

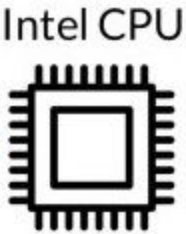
	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

Traditional Memory Buffer

Row 1	1331246660
	3/8/2012 2:44PM
	99.155.155.225
Row 2	1331246351
	3/8/2012 2:38PM
	65.87.165.114
Row 3	1331244570
	3/8/2012 2:09PM
	71.10.106.181
Row 4	1331261196
	3/8/2012 6:46PM
	76.102.156.138

Arrow Memory Buffer

session_id	1331246660
	1331246351
	1331244570
	1331261196
timestamp	3/8/2012 2:44PM
	3/8/2012 2:38PM
	3/8/2012 2:09PM
	3/8/2012 6:46PM
source_ip	99.155.155.225
	65.87.165.114
	71.10.106.181
	76.102.156.138

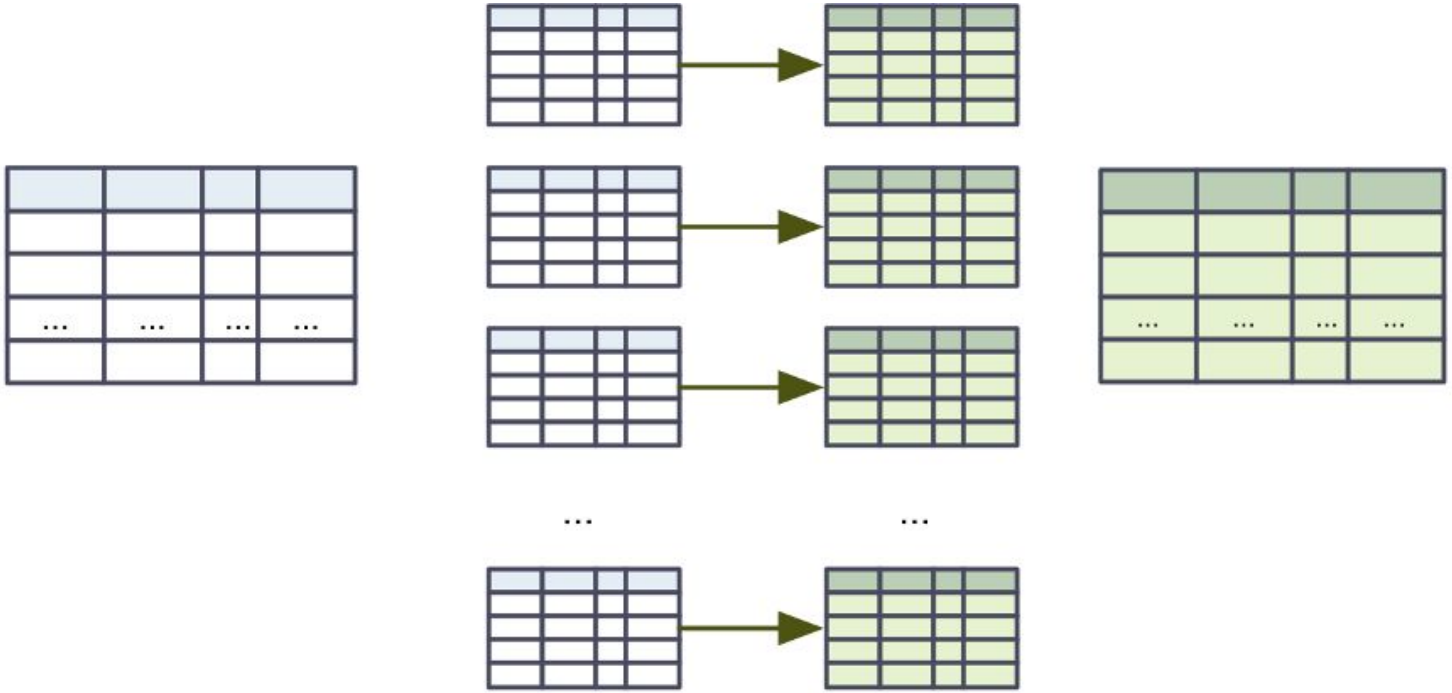


Arrow leverages the data parallelism (SIMD) in modern Intel CPUs:

```
SELECT * FROM clickstream WHERE
session_id = 1331246351
```

Volterra

Grouped Map Pandas UDF



Original data frame

Split
(the data frame in in-memory manageable chunks)

Apply
(the transformation to each chunk independently)

Combine
(each chunk back into a data frame)

Volterra

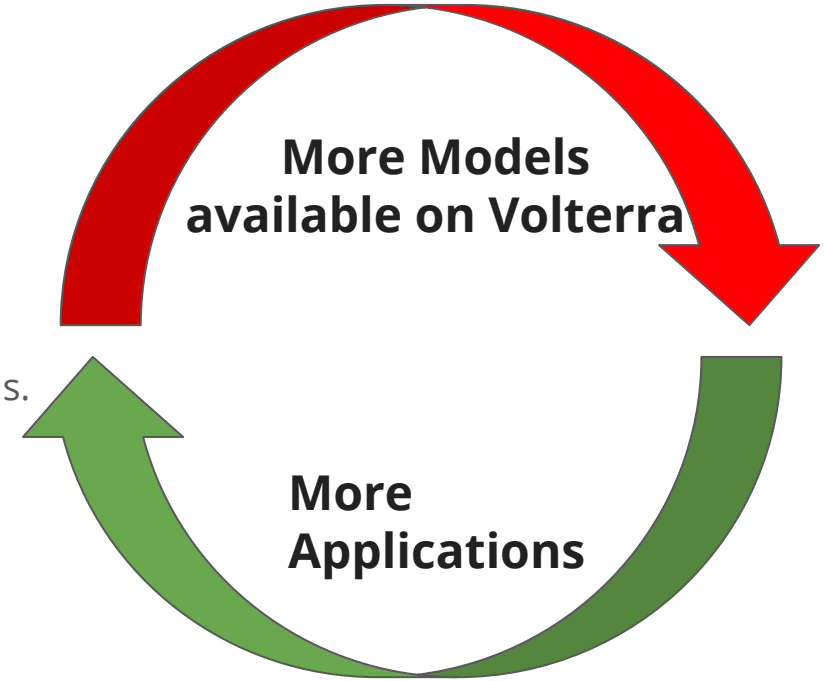
Grouped Map Pandas UDF

```
# define schema for what the pandas udf will return
schema = StructType([StructField('group_id', IntegerType()),
StructField('model_str', StringType())])

@pandas_udf(schema, functionType=PandasUDFType.GROUPED_MAP)
def train_model(df_pandas):
    # get the value of this group id
    group_id = df_pandas['group_id'].iloc[0]
    # get features and label for all training instances in this group
    X = df_pandas[['my_feature_1', 'my_feature_2']]
    Y = df_pandas['my_label']
    # train this model
    model = RandomForestRegressor()
    model.fit(X,Y)
    # get a string representation of our trained model to store
    model_str = pickle.dumps(model)
    # build the DataFrame to return
    df_to_return = pd.DataFrame([group_id, model_str],
    columns = ['group_id', 'model_str'])
    return df_to_return

spark.conf.set("spark.sql.execution.arrow.enabled", "true")
df_sp = spark.createDataFrame(df_pd)
results = df_sp.groupBy('group_id').apply(train_model)
res_pd = results.toPandas()
```

Volterra

Conclusion

- **Goal: Add new models and scale with growing customers/applications**
  - End to End Automation, Security and CI/CD
- **Features**
  - Microservice architecture
    - Embedded Spark
  - Provide platform to seamlessly add new models.
  - Minimize resource costs
  - Extend to Spark Streaming for near real-time applications



Volterra

Contacts

https://medium.com/volterra-io

@Volterra_

Volterra

https://gitlab.com/volterra.io