PayPal Data Lake journey

Deepak Mohanakumar Chandramouli, Big Data Platform Eng

Speaker Bio

Teradata

- Over 12 years of experience building Data Warehouse Solutions.
- 9+ years of experience working with Teradata based Warehouse Applications

2016

- Engineering - Foundational Big Data Applications for PayPal's Data Lake
- Apache Spark Based Ingestion Framework, Data Quality Framework, Dedup & Snapshot processes for Transactional Datasets.

2017

- Big Data Platform Engineering: to build PayPal's Gimel - Data Platform
- Gimel powers Data Lake vision & many other Big Data Engineering Use-cases in PayPal Inc.

Agenda

TERADATA PARTNERS CONFERENCE

- PayPal at Scale
  - Business
  - Technology
- PayPal - Datalake
- Compute
  - Challenges
  - Solution
- Data Access
  - Challenges
  - Solution

THE EDGE OF NEXT

- **PayPal Gimel** - Bigger Picture
- **PayPal Gimel** – Next Steps

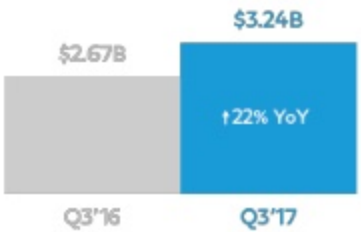
PayPal at Scale

Business &
Technology

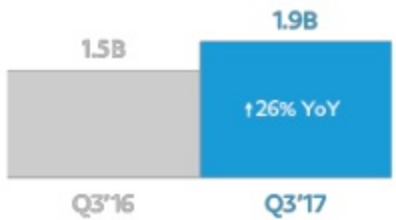
PayPal Scale | Business

- One of the world's largest internet payment companies
- PayPal platform includes **Braintree**, **Venmo**, **Paydiant**, **PP Credit** and **Xoom**

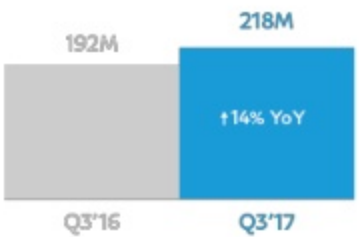
Q3 2017 Results



$3.24B
Revenue up 22% YoY

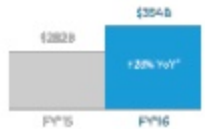


1.9B
Payment Transactions
up 26% YoY

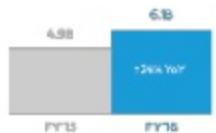


218M
Active Customer Accounts
up 14% YoY

2016 Full Year Results

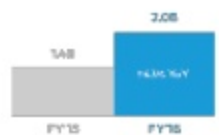


$354B
Total Payment
up 28% YoY



6.1B
Total Transactions
up 24% YoY

$102B
Mobile Payment
up 55% YoY



2.0B
Mobile Transactions
up 43% YoY

PayPal Scale | Technology

15+
Hadoop Clusters

95+ PB
Data

40,000+
Yarn Jobs Per Day

6+ compute
Supported - Hive,Pig, MR, Spark, Beam, Presto
Others - Flink

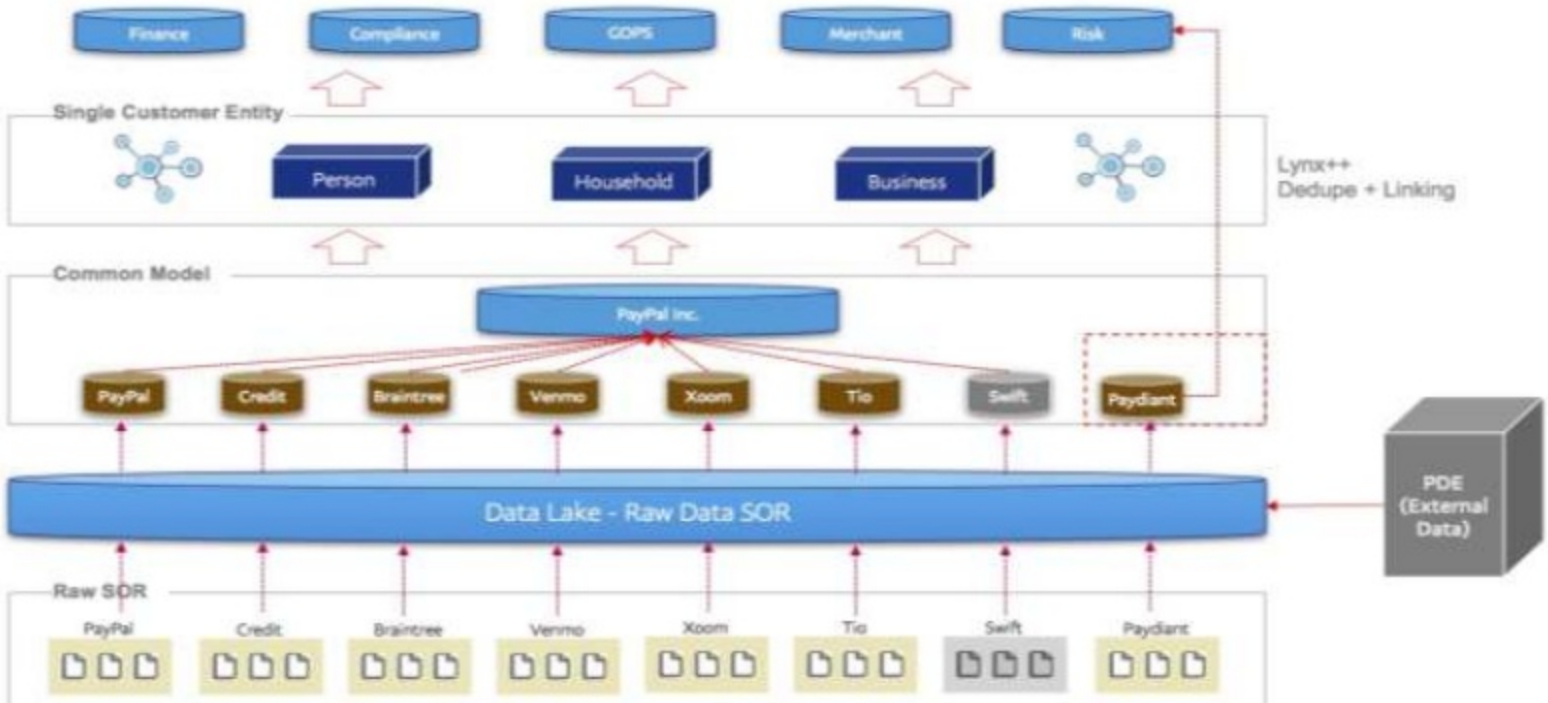
PayPal - Datalake

Bigger Picture
Complexities

Data Lake | Bigger Picture

Cross Property Data Lake – Conceptual View

Single source for all of PayPal and its subsidiaries data at the most granular level to serve the various needs across PayPal as well as the subsidiaries.



Data Lake | Recap | Complexities

Cross Property Integration

- **PayPal, Braintree, Venmo, Xoom, Credit, Tio, Paydient, More** to Come in future
- Each Property – Unique set of infrastructure & Storages
- Integration & Data Pipelines - involves **Multiple Storages** such as Oracle, MySQL, Kafka, Cloud Bases systems.

Compute / Processing Capabilities

- Multiple Clusters – Each has a purpose
- Large Scale / **Complex Data Systems**
- **Variety of Use Cases** - Stream, Batch, Interactive, ML
- **Multiple Compute Options** – Hive, Spark, Flink, Beam, Presto, Map-Reduce
- Data & Operational **Security** !

Data Lake | Recap | Complexities

Data Engineering

- **Multiple Storages** – Cloud, No-SQL, RDBMS, Messaging, In-Memory
- **100's of APIs** to access different storages – SQL & Programmatic (Python, Scala, Java…)
- **Complexity** – Many-fold if one considers different compute engines – spark, hive, flink
- Time to develop and Time to Market – goes through "**learning curves**"

Data Catalog

- **Puzzle**
- Awareness - **in what form & where data is available?**
- "Storage-specific" dataset creation
- **Dataset onboarding** – various ways
- Data assets – Lacking "**Centralized**" monitoring capabilities

2016

Beginning of the Data Lake Services

Data Lake Services |2016

Simple, Governed, Unified.

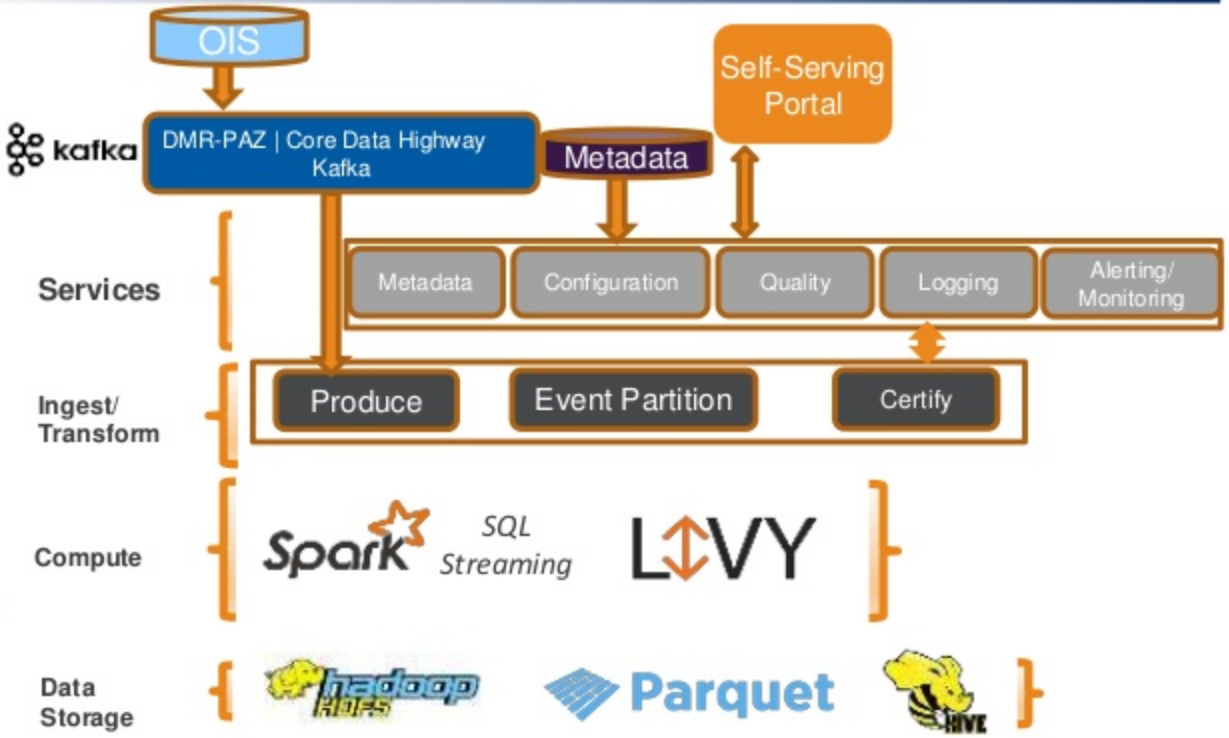
Quick Recap - Architecture

Key Features

- Configuration/Metadata driven apps
- Spark based framework
- Self-Service Portal
- Data Quality Monitoring
- Metrics Collection
- Operational Dashboards

Benefits

- Scalability
- User Experience
- High Speed Data Ingestion
- Converged Data Sources (PayPal, Braintree etc.)
- Secure and Governed Data



Data Lake Services |2016 | Recap Challenges

TERADATA PARTNERS CONFERENCE

Data Governance
- Security
- Metadata
- Data Quality & Metrics

Use cases
- Rapid growth rate of new business use cases
- Onboarding

Data Lineage
- Most common Challenge !

Data Models & Best Practices
- Data Layout
- Data Models

Data across Property - Integration
- PayPal, Credit, Braintree, Venmo, Paydiant, TIO, Xoom

Technology & Skillset
- Emerging/New technologies. Example - Spark, Flink, Beam, Kylo, …
- Talent & Learning Curve

Best Practices & Common Frameworks
- Logging
- Monitoring
- Alerting
- Auditing
- Operational Dashboards

Administration
- Hosts Management
- Software Management

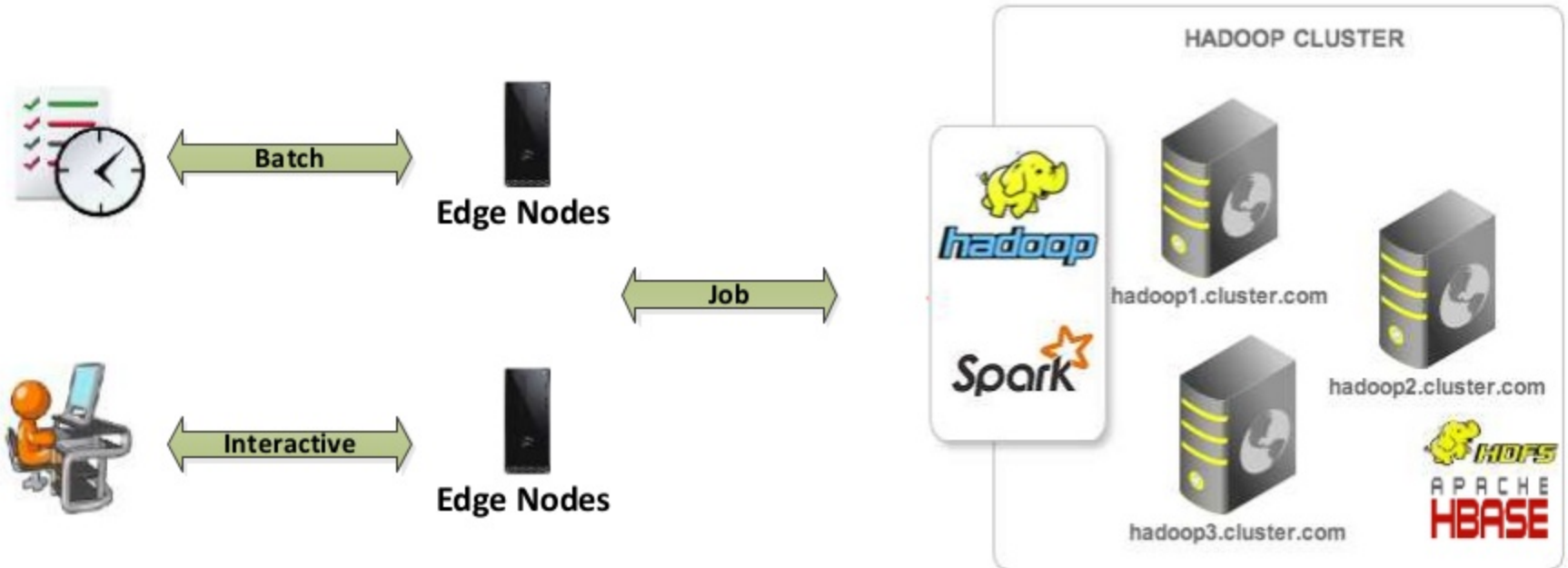
Users Experience
- Data Engineers
- Analysts
- Scientists

Challenges Faced with…

Compute

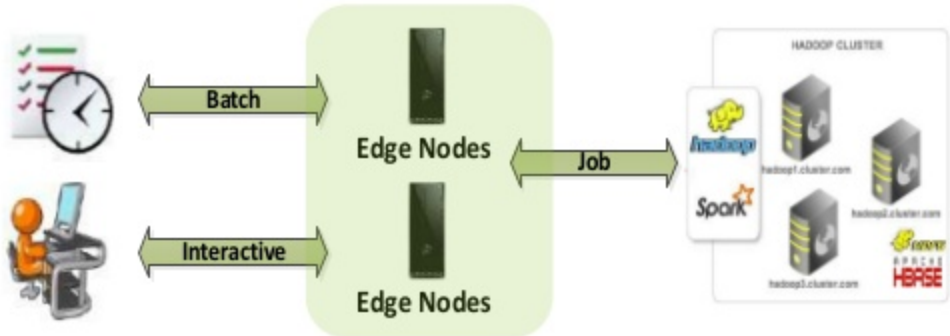
Spark on Yarn| Typically

Deployment



Data Lake | Challenges | Administrators

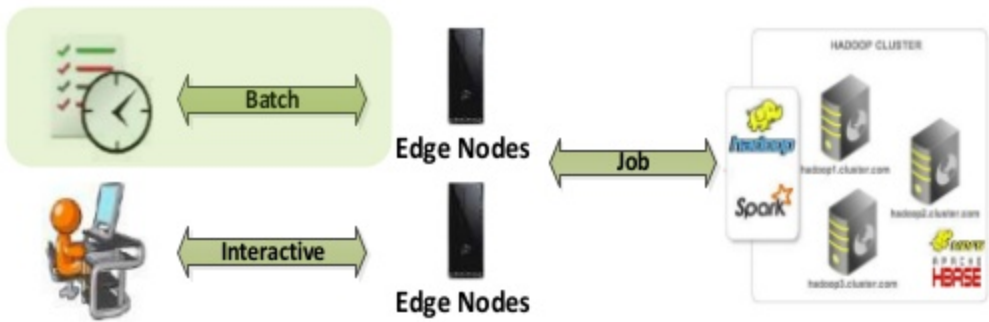
Administrators



- Need extensive support and maintenance for CLI
- Need to deploy entire stack of software
- Need to sync configurations across systems
- Need extensive testing of jobs before any upgrade

Data Lake | Challenges | Developers

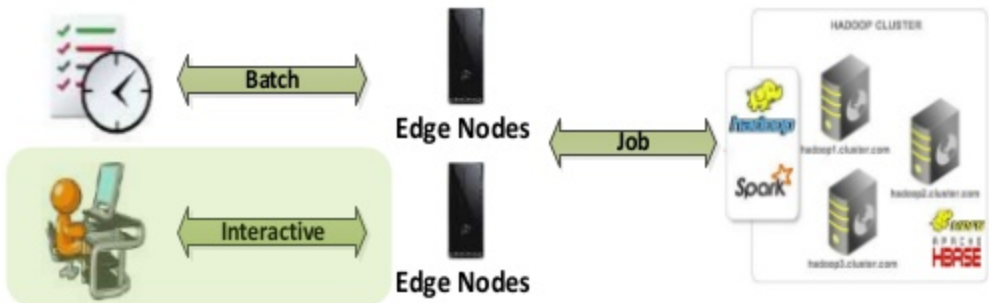
Developers



- No REST-friendly API
- No low-latency/sub-seconds execution
- No cache sharing across jobs
- No modularity and easy-restartability

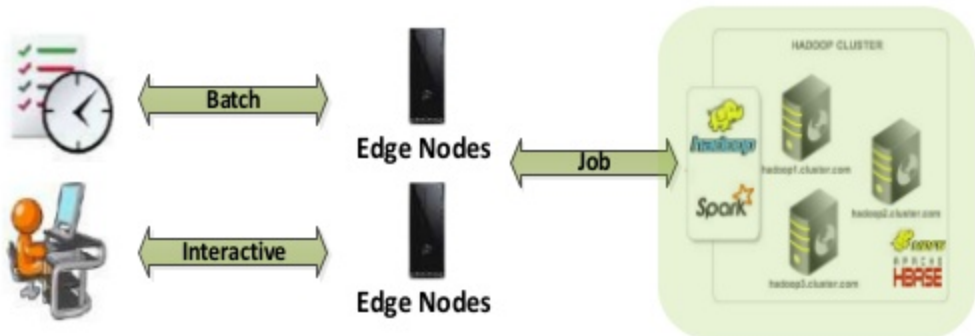
Data Lake | Challenges | Interactive Usecases

Analysts/Scientists



- No easy way of interactive applications
- No multi-tenancy support and private workspace
- No direct spark sql execution
- No Kerberos integration

Operations/Security

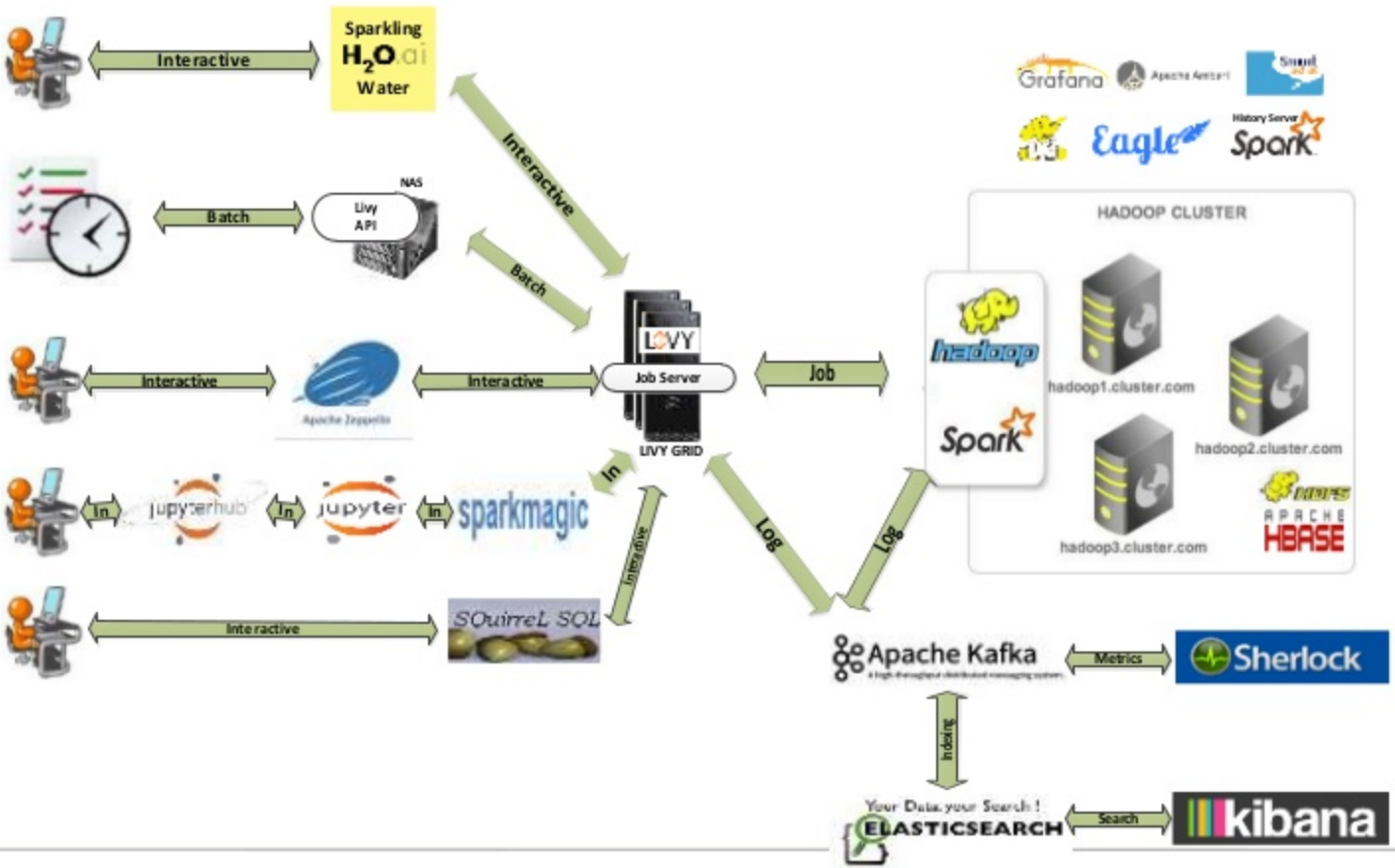


- Different ways of jobs execution and coding standards
- No uniform logging, monitoring and alerting
- Limited audit and control
- No statement level history or metrics

Solution !

Gimel
Compute Platform

Data Lake | Compute Platform



Administrators

- ✓ Less maintenance on CLI
- ✓ Deploy software stack only on Job Server
- ✓ Configurations at one place
- ✓ Easy platform/software upgrade

Developers

- ✓ REST-friendly and Docker-friendly
- ✓ Low-latency/sub-seconds execution
- ✓ **Sharing** cache across jobs
- ✓ Modularity and easy restartability

Analysts/Scientists

- ✓ User friendly interactive applications
- ✓ Multi-tenancy and Private workspace
- ✓ Direct spark sql execution
- ✓ Kerberos Support

Operations/Security

- ✓ Standardized coding and unified execution
- ✓ Uniformed logging, monitoring and alerting
- ✓ Fine-grained audit
- ✓ Complete statement level history and metric

Challenges Faced with…

Data Access

Data Platform | Data Access | Current State

Spark Read From Hbase

```
import org.apache.hadoop.hbase.client.{HBaseAdmin, Result}
import org.apache.hadoop.hbase.{ HBaseConfiguration, HTableDescriptor }
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.hadoop.hbase.io.ImmutableBytesWritable

import org.apache.spark._

object HBaseRead {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseRead").setMaster("local[2]")
    val sc = new SparkContext(sparkConf)
    val conf = HBaseConfiguration.create()
    val tableName = "table1"

    System.setProperty("user.name", "hdfs")
    System.setProperty("HADOOP_USER_NAME", "hdfs")
    conf.set("hbase.master", "localhost:60000")
    conf.setInt("timeout", 120000)
    conf.set("hbase.zookeeper.quorum", "localhost")
    conf.set("zookeeper.znode.parent", "/hbase-unsecure")
    conf.set(TableInputFormat.INPUT_TABLE, tableName)

    val admin = new HBaseAdmin(conf)
    if (!admin.isTableAvailable(tableName)) {
      val tableDesc = new HTableDescriptor(tableName)
      admin.createTable(tableDesc)
    }

    val hBaseRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat], classOf[ImmutableBytesWritable], classOf[Result])
    println("Number of Records found : " + hBaseRDD.count())
    sc.stop()
  }
}
```

Data Platform | Data Access | Current State

Spark Read From Elastic Search

```
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.spark.SparkConf
import org.elasticsearch.spark.sql._

object ElasticSearchRead {
  def main(args: Array (link is external)[String (link is external)]) {

        val conf = new SparkConf().setAppName("ReadFromES")

        conf.set("spark.es.nodes", "10.111.222.333")
        conf.set("es.index.auto.create", "true")
        conf.set("spark.es.nodes.client.only", "true")
        conf.set("es.query", "?q=*")
        conf.set("es.resource", "index/type")

        val sc = new SparkContext(conf)

        val sqlContext = new org.apache.spark.sql.SQLContext(sc)

        val es_df=sqlContext.read.format("org.elasticsearch.spark.sql").load("index/type")

        es_df.registerTempTable("things")

        val filtered_things = sqlContext.sql("select * from things where col = '1000'")

  }
}
```

Data Platform | Data Access | Current State

Spark Read From Hbase

```
import org.apache.hadoop.hbase.client.{HBaseAdmin, Result}
import org.apache.hadoop.hbase.{ HBaseConfiguration, HTableDescriptor }
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.hadoop.hbase.io.ImmutableBytesWritable

import org.apache.spark._

object HBaseRead {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseRead").setMaster("local[2]")
    val sc = new SparkContext(sparkConf)
    val conf = HBaseConfiguration.create()
    val tableName = "table1"

    System.setProperty("user.name", "hdfs")
    System.setProperty("HADOOP_USER_NAME", "hdfs")
    conf.set("hbase.master", "localhost:60000")
    conf.setInt("timeout", 120000)
    conf.set("hbase.zookeeper.quorum", "localhost")
    conf.set("zookeeper.znode.parent", "/hbase-unsecure")
    conf.set(TableInputFormat.INPUT_TABLE, tableName)

    val admin = new HBaseAdmin(conf)
    if (!admin.isTableAvailable(tableName)) {
      val tableDesc = new HTableDescriptor(tableName)
      admin.createTable(tableDesc)
    }

    val hBaseRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat], classOf[ImmutableBytesWritable], classOf[Result])
    println("Number of Records found : " + hBaseRDD.count())
    sc.stop()
  }
}
```

Spark Read From Elastic Search

```
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.spark.SparkConf
import org.elasticsearch.spark.sql._

object ElasticSearchRead {
  def main(args: Array (link is external)[String (link is external)]) {

        val conf = new SparkConf().setAppName("ReadFromES")

        conf.set("spark.es.nodes", "10.111.222.333")
        conf.set("es.index.auto.create", "true")
        conf.set("spark.es.nodes.client.only", "true")
        conf.set("es.query", "?q=*")
        conf.set("es.resource", "index/type")

        val sc = new SparkContext(conf)

        val sqlContext = new org.apache.spark.sql.SQLContext(sc)

        val es_df=sqlContext.read.format("org.elasticsearch.spark.sql").load("index/type")

        es_df.registerTempTable("things")

        val filtered_things = sqlContext.sql("select * from things where col = '1000'")

  }
}
```

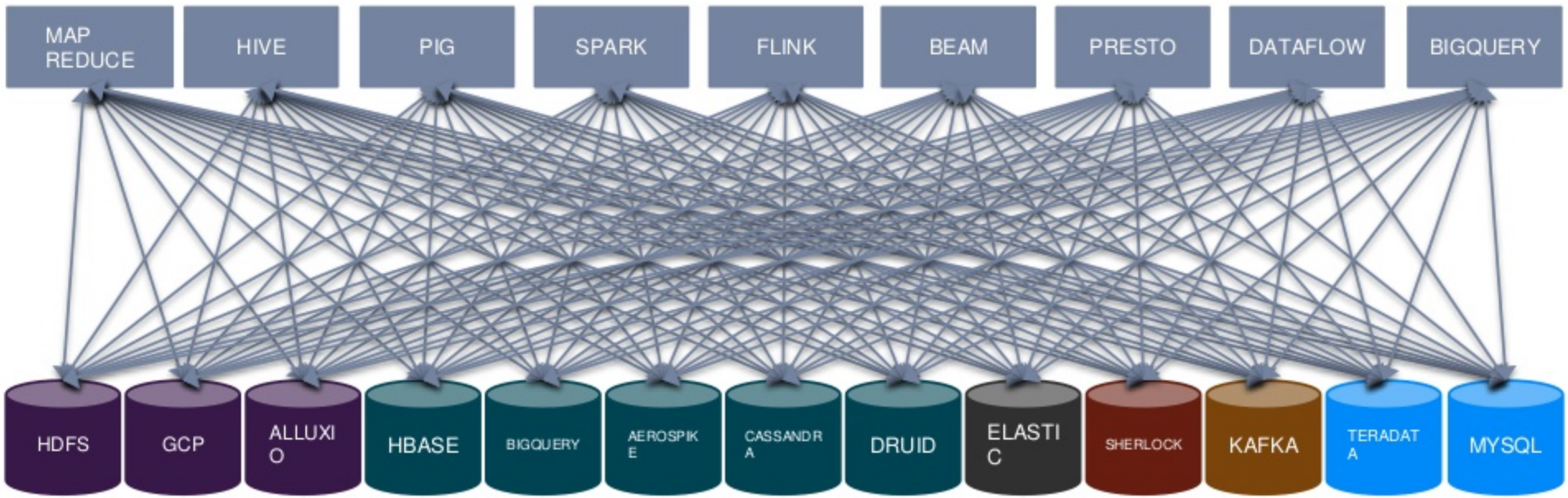
Spark Read From AeroSpike

```
import org.apache.spark._
import com.aerospike.spark.sql._
import com.aerospike.client.AerospikeClient
import com.aerospike.client.Bin
import com.aerospike.client.Key
import com.aerospike.client.Value

object AerospikeRead {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseRead").setMaster("local[2]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new SqlContext(sc)

    val thingsDF = sqlContext.read.
      format("com.aerospike.spark.sql").
      option("aerospike.seedhost", "127.0.0.1").
      option("aerospike.port", "3000").
      option("aerospike.namespace", namespace).
      option("aerospike.set", "vdk-test").
      load

    thingsDF.registerTempTable("things")

    val filteredThings = sqlContext.sql("select * from things where uno = 10")

    println("Number of Records found : " + filteredThings.count())
    sc.stop()
  }
}
```

Spark Read From Druid

```
import org.apache.spark._
import ...

object DruidRead {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseRead").setMaster("local[2]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new SqlContext(sc)

    sqlContext("""CREATE TEMPORARY TABLE orderLineItemPartSupplier
      USING org.apache.spark.sql.druid
      OPTIONS (sourceDataFrame "orderLineItemPartSupplierBase",
      timeDimensionColumn "l_shipdate",
      druidDatasource "tpch",
      druidHost "localhost",
      druidPort "8082",
      columnMapping '{ "l_quantity" : "sum_l_quantity", "pw_avgloyalty" : "sum_pw_avgloyalty" }',
      functionalDependencies '[  {"col1" : "c_name", "col2" : "c_address", "type" : "1:1"},
      {"col1" : "c_phone", "col2" : "c_address", "type" : "1:1"},
      {"col1" : "c_name", "col2" : "c_mktsegment", "type" : "n:1"},
      {"col1" : "c_name", "col2" : "c_comment", "type" : "1:1"},
      {"col1" : "c_name", "col2" : "c_nationk", "type" : "n:1"},
      {"col1" : "c_nation", "col2" : "c_region", "type" : "n:1"} ]'
    """)

    val filteredThings = sqlContext.sql("select * from orderLineItemPartSupplier")

    println("Number of Records found : " + filteredThings.count())
    sc.stop()
  }
}
```

Data Access| Bigger Picture | Today



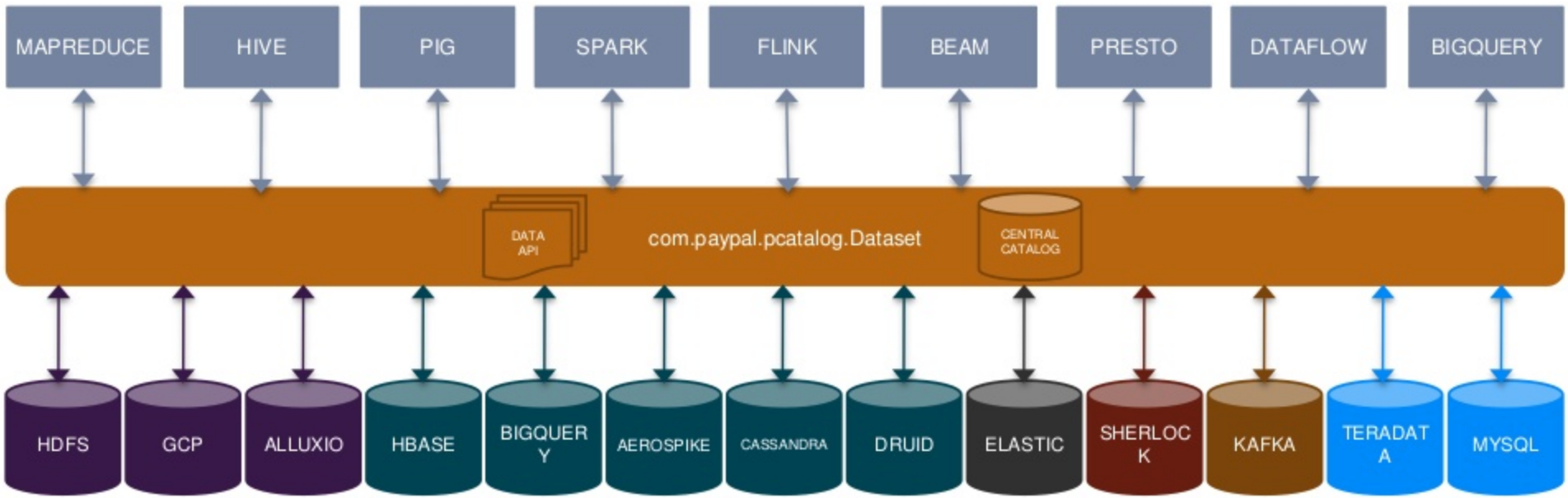
Data Platform | Today

Data Application Lifecycle - Current State

Solution !

Gimel
Data Platform

Data Platform | Data Access | With Data API



Data Platform | Objective

- **Data API**
  - Unified Data Access API
  - Access from any Compute Engine
  - Access any Storage

- **Data Catalog**
  - Global Catalog - Explore at one place
  - Onboard at one place
  - Dataset Management- Track/Approve Onboard Request
  - Dataset Stats

Data Platform | Data Access | Data API

Spark Read From Hbase

```
import org.apache.hadoop.hbase.client.{HBaseAdmin, Result}
import org.apache.hadoop.hbase.{ HBaseConfiguration, HTableDescriptor }
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.hadoop.hbase.io.ImmutableBytesWritable

import org.apache.spark._

object HBaseRead {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseRead").setMaster("local[2]")
    val sc = new SparkContext(sparkConf)
    val conf = HBaseConfiguration.create()
    val tableName = "table1"

    System.setProperty("user.name", "hdfs")
    System.setProperty("HADOOP_USER_NAME", "hdfs")
    conf.set("hbase.master", "localhost:60000")
    conf.setInt("timeout", 120000)
    conf.set("hbase.zookeeper.quorum", "localhost")
    conf.set("zookeeper.znode.parent", "/hbase-unsecure")
    conf.set(TableInputFormat.INPUT_TABLE, tableName)

    val admin = new HBaseAdmin(conf)
    if (!admin.isTableAvailable(tableName)) {
      val tableDesc = new HTableDescriptor(tableName)
      admin.createTable(tableDesc)
    }

    val hBaseRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat], classOf[ImmutableBytesWritable], classOf[
    println("Number of Records found : " + hBaseRDD.count())
    sc.stop()
  }
}
```

Spark Read From Elastic Search

```
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.spark.SparkConf
import org.elasticsearch.spark.sql._

object ElasticSearchRead {
  def main(args: Array (link is external)[String (link is external)]) {

        val conf = new SparkConf().setAppName("ReadFromES")

        conf.set("spark.es.nodes", "10.111.222.333")
        conf.set("es.index.auto.create", "true")
        conf.set("spark.es.nodes.client.only", "true")
        conf.set("es.query", "?q=*")
        conf.set("es.resource", "index/type")

        val sc = new SparkContext(conf)

        val sqlContext = new org.apache.spark.sql.SQLContext(sc)

        val es_df=sqlContext.read.format("org.elasticsearch.spark.sql").load("index/type")

        es_df.registerTempTable("things")

        val filtered_things = sqlContext.sql("select * from things where col = '1000'")

  }
}
```

Spark Read From AeroSpike

```
import org.apache.spark._
import com.aerospike.spark.sql._
import com.aerospike.client.AerospikeClient
import com.aerospike.client.Bin
import com.aerospike.client.Key
import com.aerospike.client.Value

object AerospikeRead {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseRead").setMaster("local[2]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new SQLContext(sc)

    val things2F = sqlContext.read.
      format("com.aerospike.spark.sql").
      option("aerospike.seedhost", "127.0.0.1").
      option("aerospike.port", "3000").
      option("aerospike.namespace", namespace).
      option("aerospike.set", "old-test").
      load

    things2F.registerTempTable("things")

    val filteredThings = sqlContext.sql("select * from things where one = 88")

    println("Number of Records found : " + filteredThings.count())
    sc.stop()
  }
}
```

Spark Read From Druid

```
import org.apache.spark._
import io.druid.related_imports

object DruidRead {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseRead").setMaster("local[2]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new SQLContext(sc)

    sqlContext("""CREATE TEMPORARY TABLE orderLineItemPartSupplier
      USING org.apache.sparkdruid
      OPTIONS (sourceDataFrame "orderLineItemPartSupplierBase",
      timeDimensionColumn "l_shipdate",
      zkHostAddresses "host",
      zkHostBase "/sparkdruid",
      zkPort "8082",
      columnMapping '{ "l_quantity" : "sum_l_quantity", "ps_availqty" : "sum_ps_availqty" }',
      functionalDependencies '[  {"col1" : "c_name", "col2" : "c_address", "type" : "1:1"},
        {"col1" : "c_phone", "col2" : "c_address", "type" : "1:1"},
        {"col1" : "c_name", "col2" : "c_mktsegment", "type" : "n:1"},
        {"col1" : "c_name", "col2" : "c_comment", "type" : "1:1"},
        {"col1" : "c_name", "col2" : "c_nation", "type" : "n:1"},
        {"col1" : "c_nation", "col2" : "c_region", "type" : "n:1"} ]'
    )""")

    val filteredThings = sqlContext.sql("select * from orderLineItemPartSupplier")

    println("Number of Records found : " + filteredThings.count())
    sc.stop()
  }
}
```

✓

With Data API

```
val df = Dataset.Read("dataset_from_hbase")
val df = Dataset.Read("dataset_from_elastic")
val df = Dataset.Read("dataset_from_aerospike")
val df = Dataset.Read("dataset_from_druid")
val df = Dataset.Read("dataset_from_hdfs")
val df = Dataset.Read("dataset_from_kafka")
val df = Dataset.Read("dataset_from_sherlock")
val df = Dataset.Read("dataset_from_cassandra")
val df = Dataset.Read("dataset_from_solr")
val df = Dataset.Read("dataset_from_mysql")
val df = Dataset.Read("dataset_from_teradata")
val df = Dataset.Read("dataset_from_oracle")
val df = Dataset.Read("dataset_from_bigquery")
val df = Dataset.Read("dataset_from_bigtable")
val df = Dataset.Read("dataset_from_alluxio")
```

Data Platform | Data Access | Gimel SQL

Spark Read From Hbase

```
import org.apache.hadoop.hbase.client.{HBaseAdmin, Result}
import org.apache.hadoop.hbase.{ HBaseConfiguration, HTableDescriptor }
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.hadoop.hbase.io.ImmutableBytesWritable

import org.apache.spark._

object HBaseRead {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseRead").setMaster("local[2]")
    val sc = new SparkContext(sparkConf)
    val conf = HBaseConfiguration.create()
    val tableName = "table1"

    System.setProperty("user.name", "hdfs")
    System.setProperty("HADOOP_USER_NAME", "hdfs")
    conf.set("hbase.master", "localhost:60000")
    conf.setInt("timeout", 120000)
    conf.set("hbase.zookeeper.quorum", "localhost")
    conf.set("zookeeper.znode.parent", "/hbase-unsecure")
    conf.set(TableInputFormat.INPUT_TABLE, tableName)

    val admin = new HBaseAdmin(conf)
    if (!admin.isTableAvailable(tableName)) {
      val tableDesc = new HTableDescriptor(tableName)
      admin.createTable(tableDesc)
    }

    val hBaseRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat], classOf[ImmutableBytesWritable], classOf[
    println("Number of Records found : " + hBaseRDD.count())
    sc.stop()
  }
}
```

Spark Read From Elastic Search

```
import org.apache.spark.SparkContext
import org.apache.spark.sql.SQLContext
import org.apache.spark.SparkConf
import org.elasticsearch.spark.sql._

object ElasticSearchRead {
  def main(args: Array (link is external)[String (link is external)]) {

        val conf = new SparkConf().setAppName("ReadFromES")

        conf.set("spark.es.nodes", "10.111.222.333")
        conf.set("es.index.auto.create", "true")
        conf.set("spark.es.nodes.client.only", "true")
        conf.set("es.query", "?q=*")
        conf.set("es.resource", "index/type")

        val sc = new SparkContext(conf)

        val sqlContext = new org.apache.spark.sql.SQLContext(sc)

        val es_df=sqlContext.read.format("org.elasticsearch.spark.sql").load("index/type")

        es_df.registerTempTable("things")

        val filtered_things = sqlContext.sql("select * from things where col = '1000'")

  }
}
```

Spark Read From AeroSpike

```
import org.apache.spark._
import com.aerospike.spark.sql._
import com.aerospike.client.AerospikeClient
import com.aerospike.client.BCo
import com.aerospike.client.Key
import com.aerospike.client.Value

object AerospikeRead {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseRead").setMaster("local[2]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new SQLContext(sc)

    val thingsDF = sqlContext.read.
      format("com.aerospike.spark.sql").
      option("aerospike.seedhost", "127.0.0.1").
      option("aerospike.port", "3000").
      option("aerospike.namespace", namespace).
      option("aerospike.set", "old-test").
      load

    thingsDF.registerTempTable("things")

    val filteredThings = sqlContext.sql("select * from things where one = 10")

    println("Number of Records found : " + filteredThings.count())
    sc.stop()
  }
}
```

Spark Read From Druid

```
import org.apache.spark._
import io.druid.related_imports

object DruidRead {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("HBaseRead").setMaster("local[2]")
    val sc = new SparkContext(sparkConf)
    val sqlContext = new SQLContext(sc)

    sqlContext("""CREATE TEMPORARY TABLE orderLineItemPartSupplier
      USING org.apache.sparkline.druid
      OPTIONS (sourceDataFrame "orderLineItemPartSupplierBase",
      timeDimensionColumn "l_shipdate",
      druidHost "localhost",
      druidPort "8082",
      columnMapping '{ "l_quantity" : "sum_l_quantity", "ps_availqty" : "sum_ps_availqty" }',
      functionalDependencies '[  {"col1" : "c_name", "col2" : "c_address", "type" : "1:1"},
        {"col1" : "c_phone", "col2" : "c_address", "type" : "1:1"},
        {"col1" : "c_name", "col2" : "c_mktsegment", "type" : "n:1"},
        {"col1" : "c_name", "col2" : "c_comment", "type" : "1:1"},
        {"col1" : "c_name", "col2" : "c_nation", "type" : "n:1"},
        {"col1" : "c_nation", "col2" : "c_region", "type" : "n:1"} ]'
    )""")

    val filteredThings = sqlContext.sql("select * from orderLineItemPartSupplier")

    println("Number of Records found : " + filteredThings.count())
    sc.stop()
  }
}
```

✓

With Gimel SQL

```
SELECT * FROM dataset_from_hbase
SELECT * FROM dataset_from_elastic
SELECT * FROM dataset_from_aerospike
SELECT * FROM dataset_from_druid
SELECT * FROM dataset_from_hdfs
SELECT * FROM dataset_from_kafka
SELECT * FROM dataset_from_sherlock
SELECT * FROM dataset_from_cassandra
SELECT * FROM dataset_from_solr
SELECT * FROM dataset_from_mysql
SELECT * FROM dataset_from_teradata
SELECT * FROM dataset_from_oracle
SELECT * FROM dataset_from_bigquery
SELECT * FROM dataset_from_bigtable
SELECT * FROM dataset_from_alluxio
```

Data Platform | Express everything as Gimel SQL

All Kind of Hive SQLs Supported !

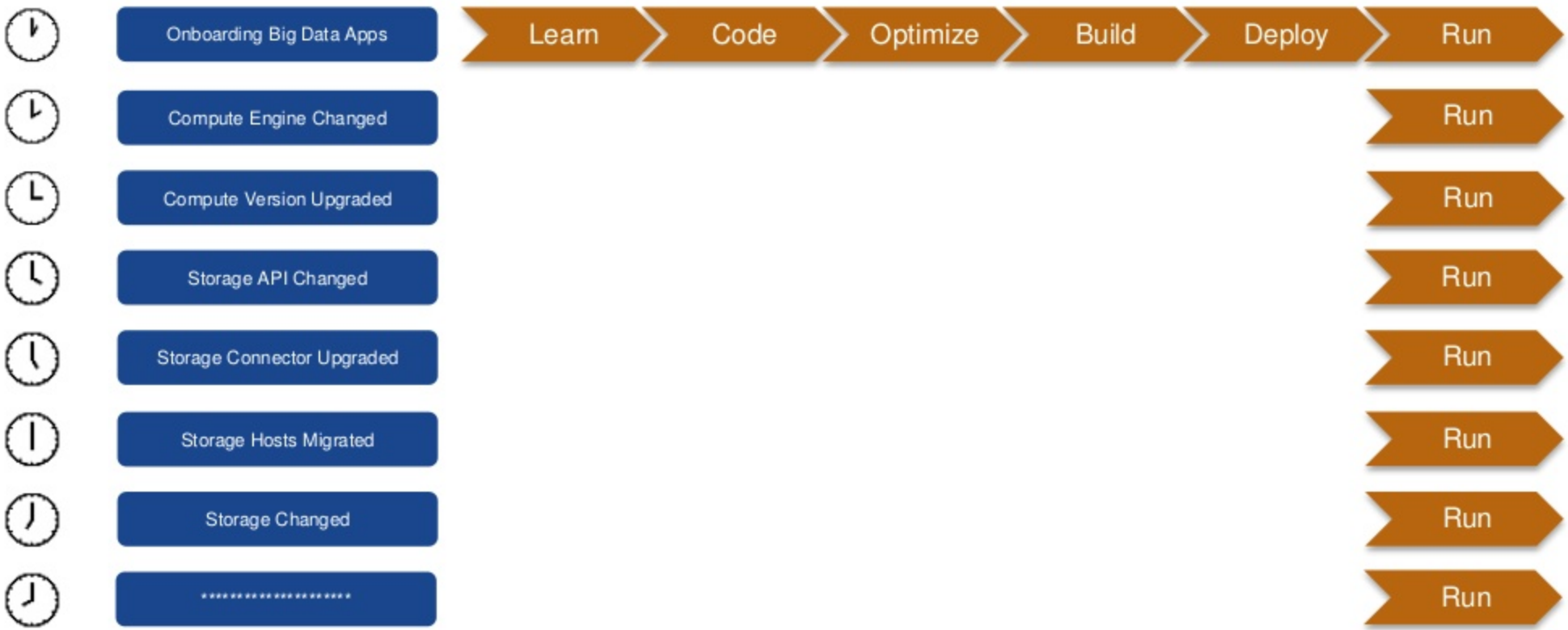
Batch SQL

```
%%gimel-batch
insert into pcatalog.hdfs_dataset
 partition(yyyy,mm,dd,hh,mi)
 select kafka_ds.*,gimel_load_id
,substr(commit_timestamp,1,4) as yyyy
,substr(commit_timestamp,6,2) as mm
,substr(commit_timestamp,9,2) as dd
,substr(commit_timestamp,12,2) as hh
,case when cast(substr(commit_timestamp,15,2) as INT) <= 30 then "00" else
"30" end as mi
from pcatalog.kafka_dataset kafka_ds;
```

Streaming SQL

```
%%gimel-stream
insert into pcatalog.hdfs_dataset
 partition(yyyy,mm,dd,hh,mi)
 select kafka_ds.*,gimel_load_id
,substr(commit_timestamp,1,4) as yyyy
,substr(commit_timestamp,6,2) as mm
,substr(commit_timestamp,9,2) as dd
,substr(commit_timestamp,12,2) as hh
,case when cast(substr(commit_timestamp,15,2) as INT) <= 30 then "00" else
"30" end as mi
from pcatalog.kafka_dataset kafka_ds;
```

Data Platform | Data Application Lifecycle | With Data API



Data Platform | Data API | Unlocking Additional Benefits

- ***Explorer***
  - View sample data
  - Add capacity reservations
  - Approval and Tracking
- ***Discovery***
  - Auto-discovery of datasets and importing into PCatalog
- ***Dashboard***
  - Operation dashboard: Metrics, statistics, trends, refresh status
  - Admin dashboard: Capacity/growth/space statistics and action
- ***Alert***
  - User alerts to notify user datasets refresh delay and quality issue
  - Admin alerts to notify datasets capacity and access violation
- ***Query/BI Integration***
  - Integrate catalog with Jupyter, Zeppelin and query any dataset

- ***Out-of-the-box Benefits***
  - Connector dependencies maintained by BDPE
  - Change component version at one place. E.g. Hbase version
  - Easy cluster upgrade
  - Abstract storage from user. Just Prod, QA and Dev
  - Add new compute engine to stack easily
  - Add new storage system to stack easily
  - Time to Develop
  - Time to Market

Presenting ….

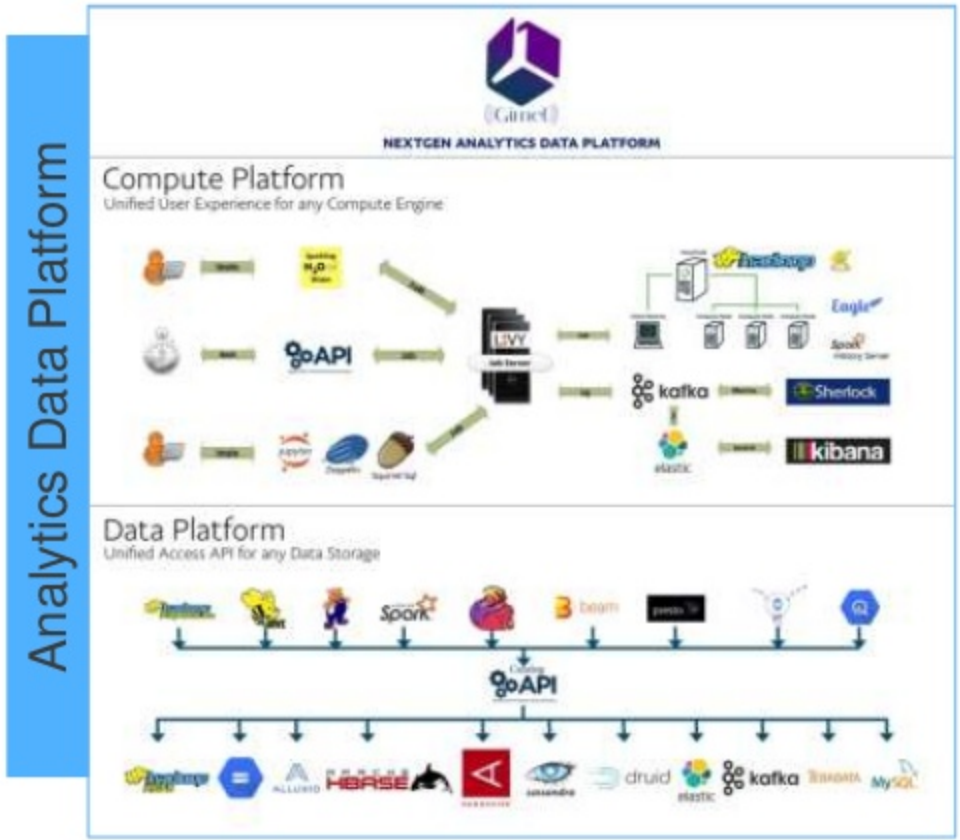
PayPal Gimel

Data Platform

+

Compute Platform

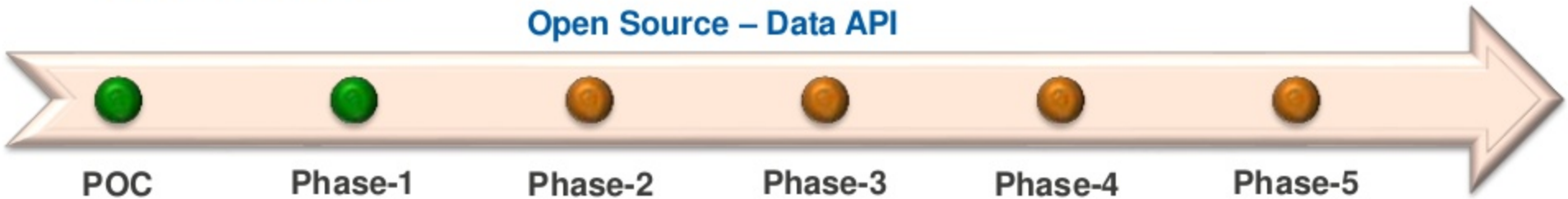
PayPal GIMEL



Data Platform: Plan

2017 ==========➔ 2018 ==========➔

Open Source – Data API



POC

- ✓ Analysis
- ✓ Evaluation
- ✓ Show & Tell

Phase-1

Compute Engine:
- ✓ Spark
- ✓ Hive
- ✓ Presto

Storage:
- ✓ TERADATA
- ✓ HDFS
- ✓ Hbase
- ✓ Elastic Search
- ✓ Kafka
- ✓ Aerospike
- ✓ Sherlock

Catalog:
- ✓ Web UI
- ✓ 1Time Onboard
- ✓ Catalog API

Phase-2

Compute Engine:
- ✓ Spark
- ✓ Hive
- ✓ Presto

Storage:
- ✓ Druid
- ✓ Cassandra

Catalog:
- ✓ Web UI
- ✓ Auto-Onboard
- ✓ Catalog API

Phase-3

Compute Engine:
- ✓ Pig/MR
- ✓ Beam
- ✓ Flink

Storage:
- ✓ Alluxio
- ✓ Oracle
- ✓ Mysql

Catalog:
- ✓ Metrics/Trend/Status
- ✓ Discovery/Alerts
- ✓ Onboard Approval
- ✓ Jira Integration
- ✓ Rally Integration
- ✓ Storage Integration

Phase-4

Compute Engine:
- ✓ Big Query

Storage:
- ✓ GSFS
- ✓ Big Query
- ✓ Big Table

Catalog:
- ✓ Enhancements
- ✓ Improvements
- ✓ Deployment

Phase-5

- ✓ Cont. Onboarding
- ✓ Post-Prod Support
- ✓ Maintenance

Thank You!

Rate This Session # 0348

with the PARTNERS Mobile App

Questions/Comments

Email: dmohanakumarchan@paypal.com

LinkedIn: https://www.linkedin.com/in/deepakmc/

Appendix

References

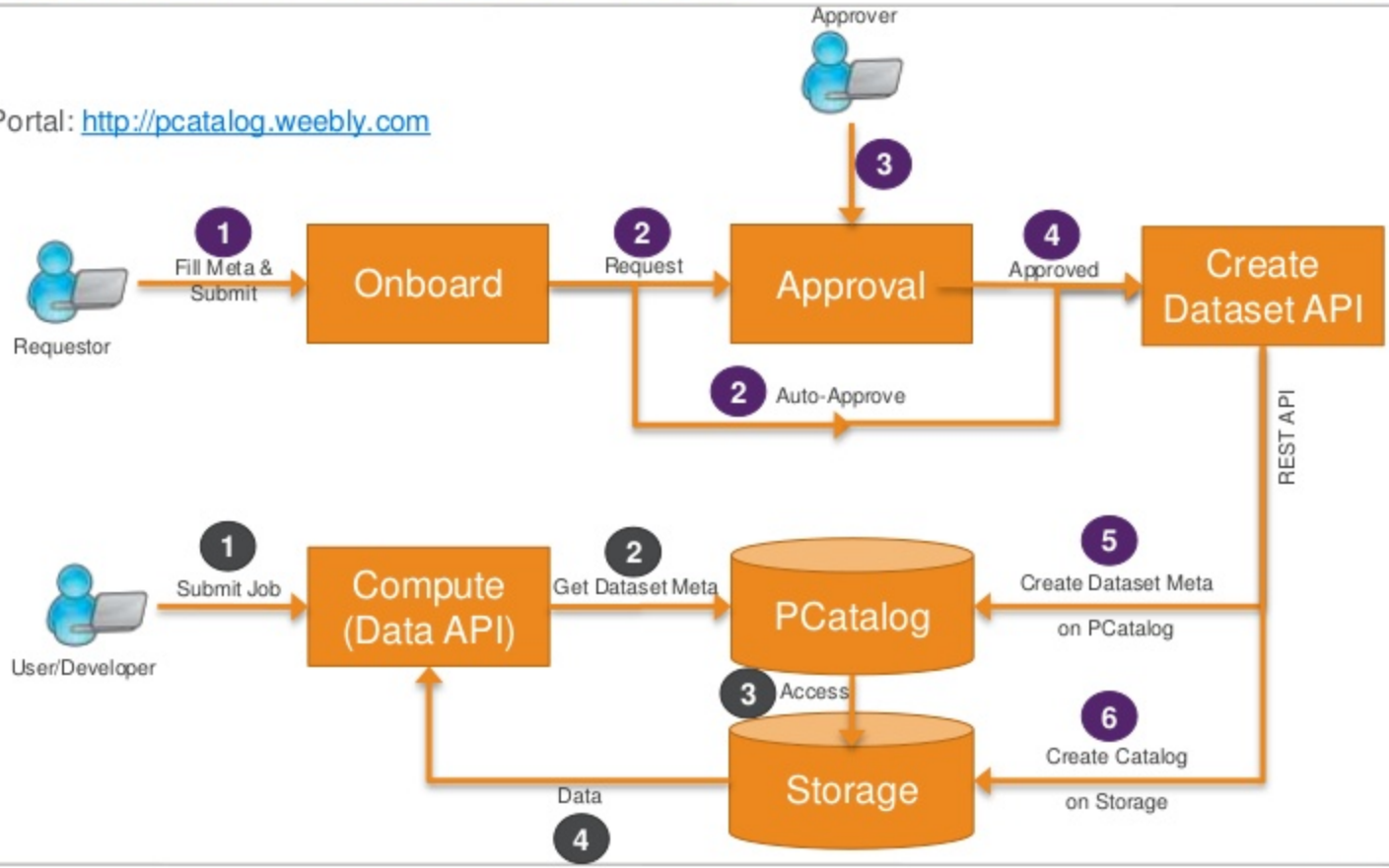
Approver

Google Search Links Used for Logos of open source technologies :
https://www.google.com/search?q=big+data+stack+icons&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiwnevLztzWAhVHsFQKHZp7CukQ_AUICigB&biw=1369&bih=739

PayPal Q3 2017 & 2016 Full Year Results :
https://www.paypal.com/us/webapps/mpp/about

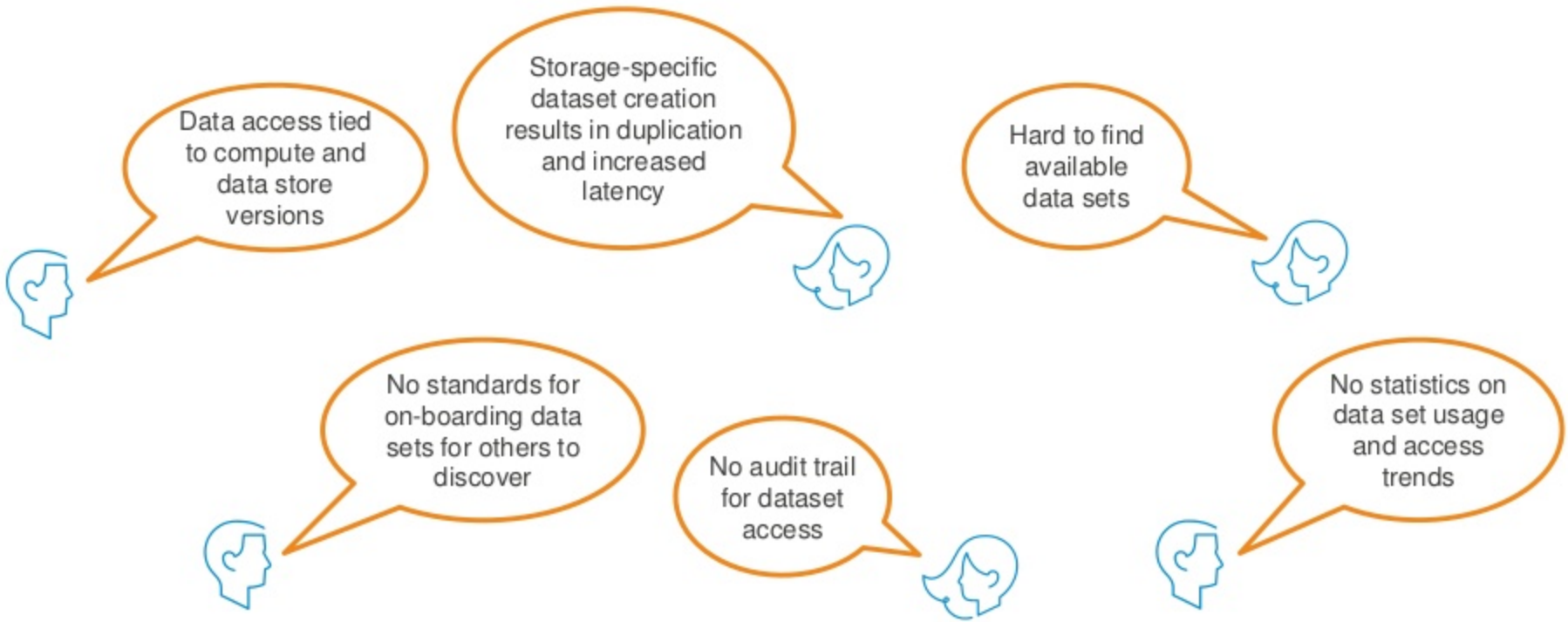
Data Platform: Process Flow

- Sample Portal: http://pcatalog.weebly.com

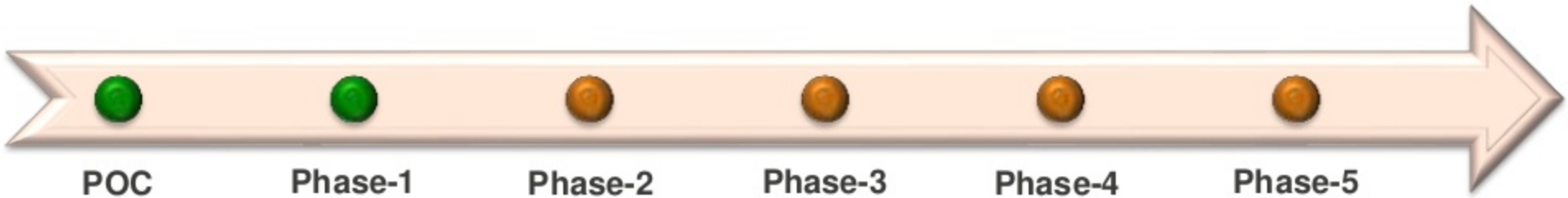


Data Platform | Today

- Data Access – Other challenges



Data Platform: Plan Proposal



POC

- ✓ Analysis
- ✓ Evaluation
- ✓ Show & Tell

Phase-1

Compute Engine:
- ✓ Spark
- ✓ Hive
- ✓ Presto

Storage:
- ✓ HDFS
- ✓ Hbase
- ✓ Elastic Search
- ✓ Kafka

Catalog:
- ✓ Web UI
- ✓ 1Time Onboard
- ✓ Catalog API

Phase-2

Compute Engine:
- ✓ Spark
- ✓ Hive
- ✓ Presto

Storage:
- ✓ Aerospike
- ✓ Druid
- ✓ Sherlock
- ✓ Cassandra

Catalog:
- ✓ Web UI
- ✓ Auto-Onboard
- ✓ Catalog API

Phase-3

Compute Engine:
- ✓ Pig/MR
- ✓ Beam
- ✓ Flink

Storage:
- ✓ Alluxio
- ✓ Teradata
- ✓ Oracle
- ✓ Mysql

Catalog:
- ✓ Metrics/Trend/Status
- ✓ Discovery/Alerts
- ✓ Onboard Approval
- ✓ Jira Integration
- ✓ Rally Integration
- ✓ Storage Integration

Phase-4

Compute Engine:
- ✓ Big Query

Storage:
- ✓ GSFS
- ✓ Big Query
- ✓ Big Table

Catalog:
- ✓ Enhancements
- ✓ Improvements
- ✓ Deployment

Phase-5

- ✓ Cont. Onboarding
- ✓ Post-Prod Support
- ✓ Maintenance