NOSQL UNIT-3

Dr. SELVA KUMAR S

B.M.S COLLEGE OF ENGINEERING

AGENDA

- NOSQL in CLOUD
  - Exploring ready-to-use NoSQL databases in the cloud
  - Leveraging Google AppEngine and its scalable data store
  - Using Amazon SimpleDB
- Parallel Processing with Map Reduce
- BigData with Hive

EXPLORING READY-TO-USE NOSQL DATABASES IN THE CLOUD

- Google and Amazon, have achieved
  - High availability
  - Ability to concurrently service millions of users
  - Scaling out horizontally among multiple machines
  - Spread across multiple data centers.
- Success stories of large-scale web applications like those from Google and Amazon have proven that
  - Horizontally scaled environments
  - NoSQL solutions
  - Available on-demand
- Provisioned as required have been christened as the “cloud.”
- If scalability and availability is your priority, NoSQL in the cloud is possibly the ideal setup.

NOSQL OPTIONS IN THE CLOUD

- Google's Bigtable data store
- Amazon SimpleDB

GOOGLE APP ENGINE DATA STORE

- The Google App Engine (GAE) provides a sandboxed deployment environment for applications.
- It is written using:
  - Python programming
  - Java Virtual Machine (JVM)
- Google provides developers with a set of rich APIs and an SDK to build applications for the app engine.

OVERVIEW

- **Google App Engine** (**GAE**) is a Platform as a Service (PaaS) cloud computing platform for developing and hosting web applications in Google-managed data centers.
- Google's Platform to build web applications on Cloud.

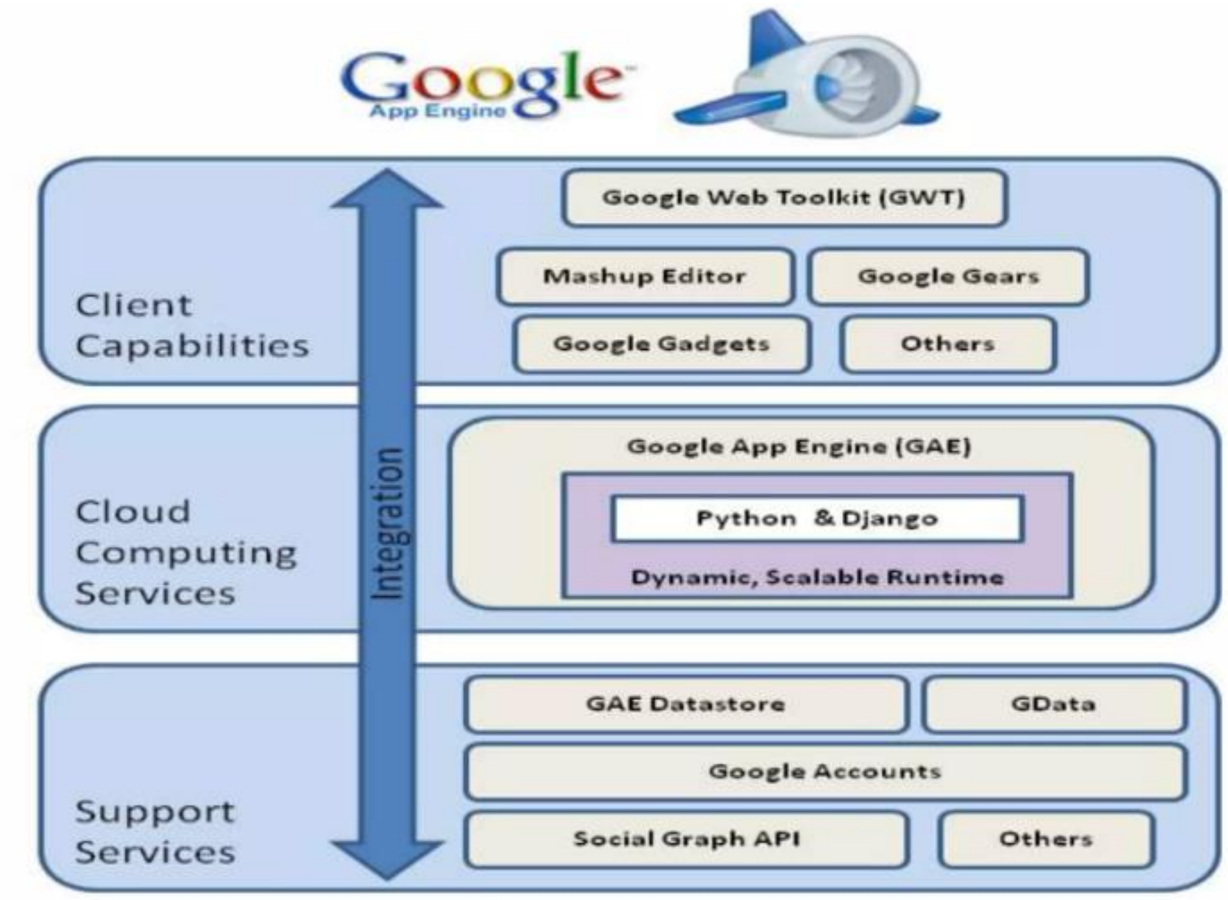
- Easy to build.
- Easy to maintain.
- Easy to scale as the traffic and storage needs grow.
- Automatic scaling and load balancing.
- Transactional data store model.

- Free for up to 1 GB of storage and enough CPU and bandwidth to support 5 million page views a month. 10 Applications per Google account.

WHY APP ENGINE?

- Lower total cost of ownership
- Rich set of APIs
- Fully featured SDK for local development
- Ease of Deployment

ARCHITECTURE OF APP ENGINE



PROGRAMMING LANGUAGES SUPPORTED

- **Java:**
- App Engine runs JAVA apps on a JAVA 7 virtual machine (currently
- supports JAVA 6 as well).
- Uses JAVA Servlet standard for web applications:
  - WAR (Web Applications ARchive) directory structure.
  - Servlet classes
  - Java Server Pages (JSP)
  - Static and data files
  - Deployment descriptor (web.xml)
  - Other configuration files

PROGRAMMING LANGUAGES SUPPORTED

- **Python:**
  - Uses WSGI (Web Server Gateway Interface) standard.
  - Python applications can be written using:
    - Webapp2 framework
    - Django framework
    - Any python code that uses the CGI (Common Gateway Interface) standard.

PROGRAMMING LANGUAGES SUPPORTED

- **PHP (Experimental support):**
  - Local development servers are available to anyone for developing and testing local applications.

- **Google's Go:**
  - Go is an Google's open source programming environment.
  - Tightly coupled with Google App Engine.
  - Applications can be written using App Engine's Go SDK.

DATA STORAGE

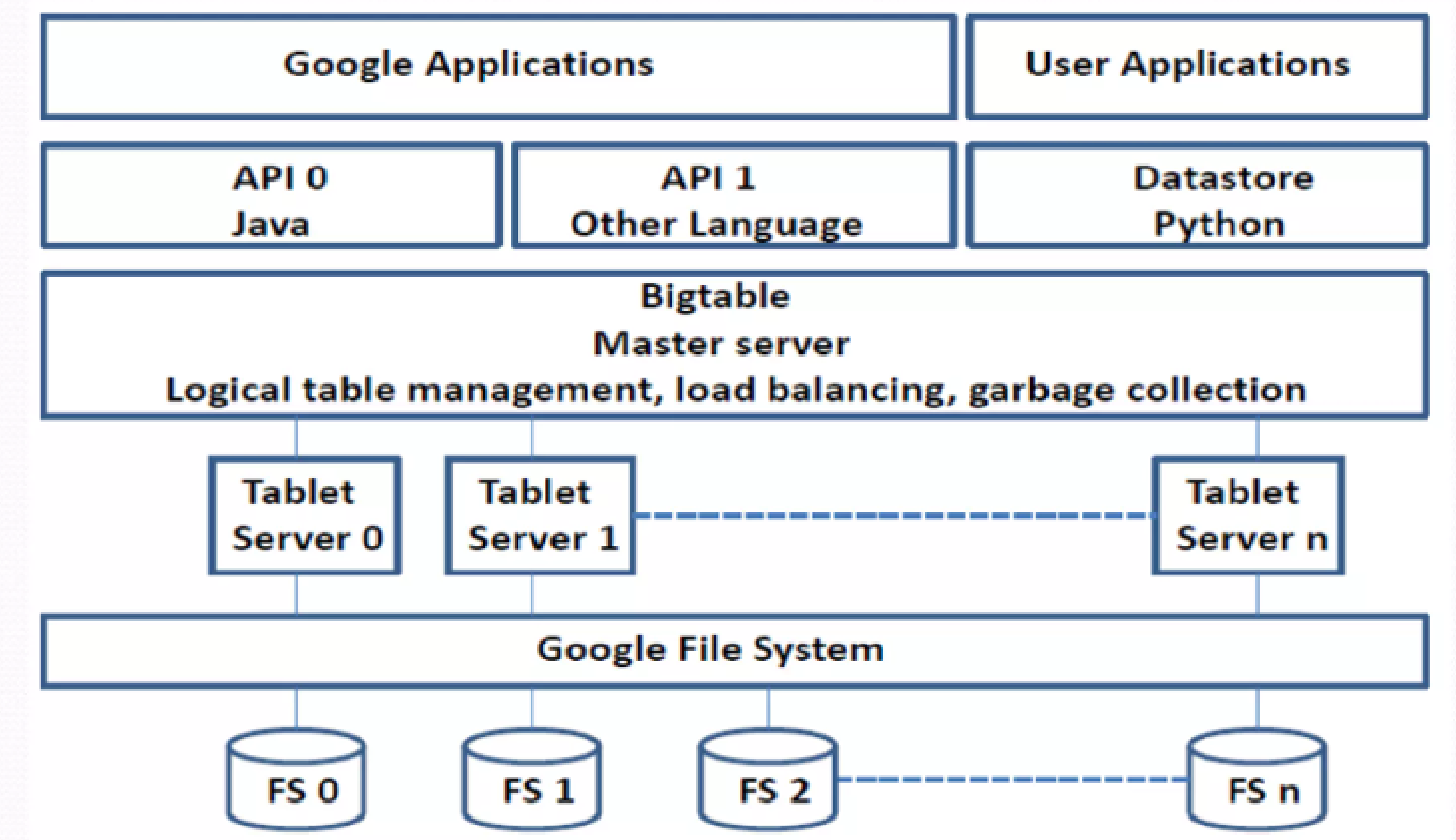
- **App Engine Datastore:**
  - NoSQL schema-less object based data storage, with a query engine and
  - atomic transactions.
  - Data object is called a "Entity" that has a kind (~ table name) and a set of
  - properties (~ column names).
  - JAVA JDO/ JPA interfaces and Python datastore interfaces.

- **Google cloud SQL:**
  - Provides a relational SQL database service.
  - Similar to MySQL RDBMS.

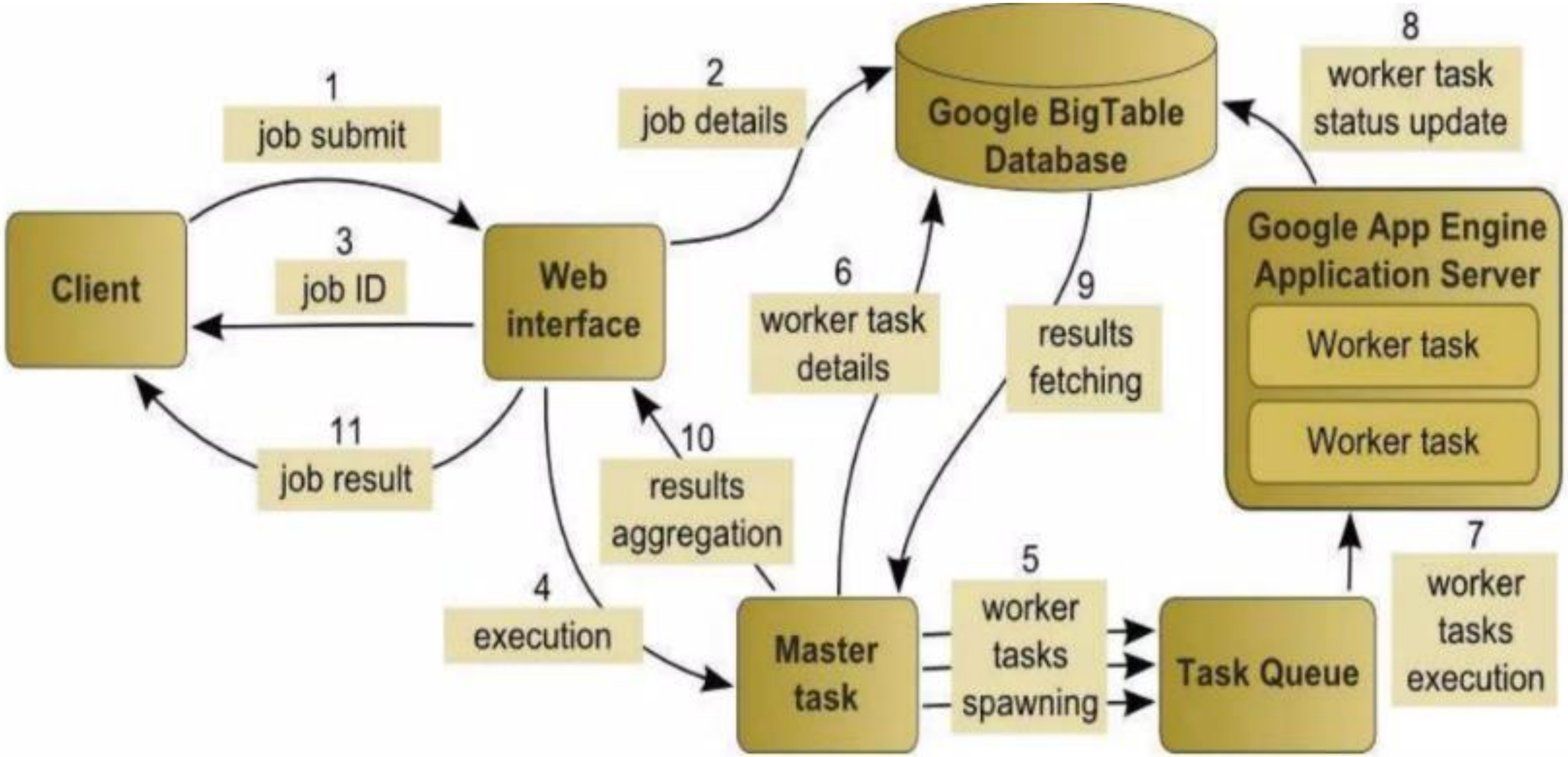
DATA STORAGE

- **Google cloud store:**
- RESTful service for storing and querying data.
- Fast, scalable and highly available solution.
- Provides Multiple layers of redundancy. All data is replicated to multiple
- data centers.
- Provides different levels of access control.
- HTTP based APIs.

GOOGLE DATA STORE ARCHITECTURE



GOOGLE DATA STORE ARCHITECTURE

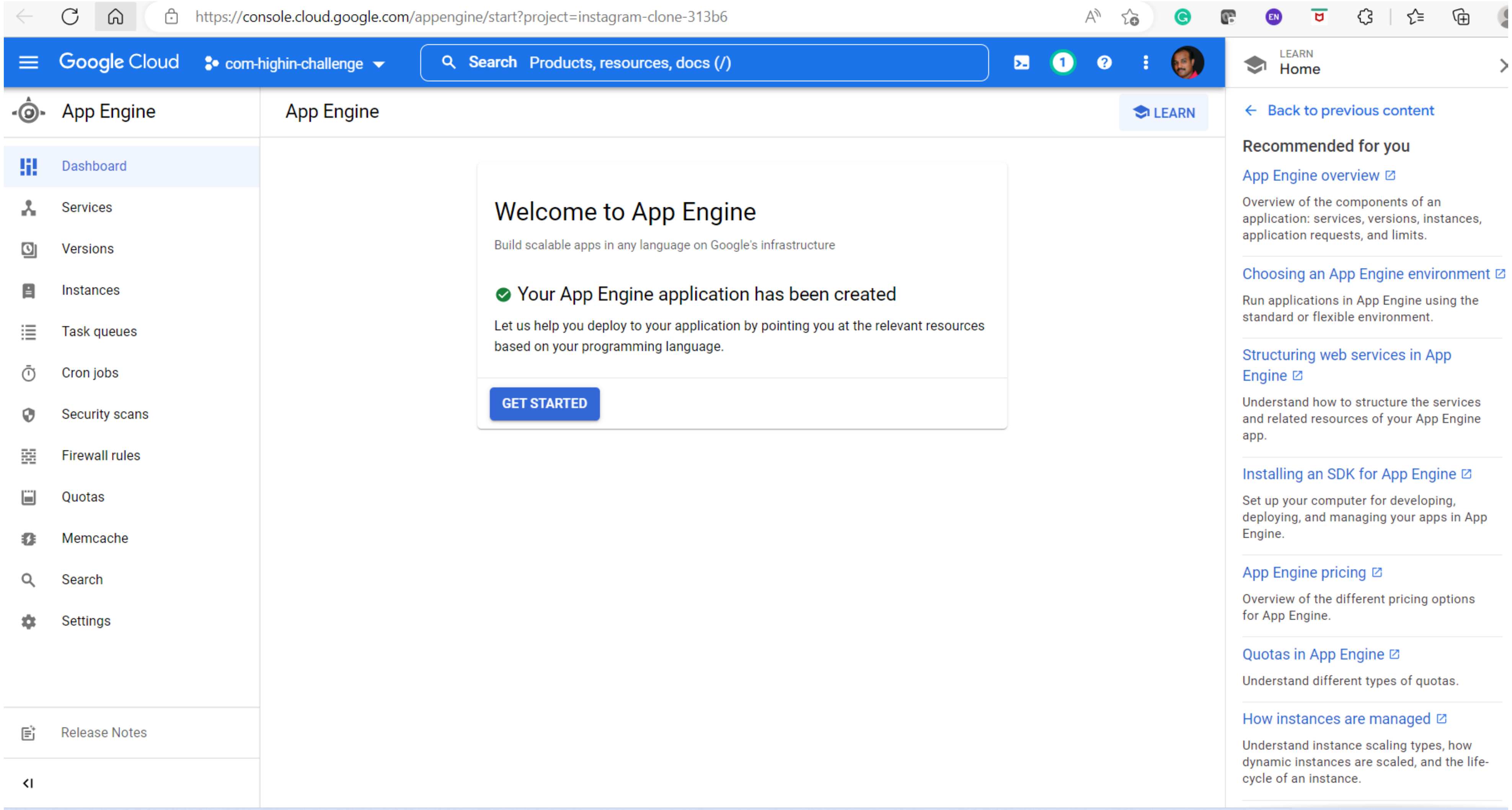


WHEN TO USE GOOGLE APP ENGINE

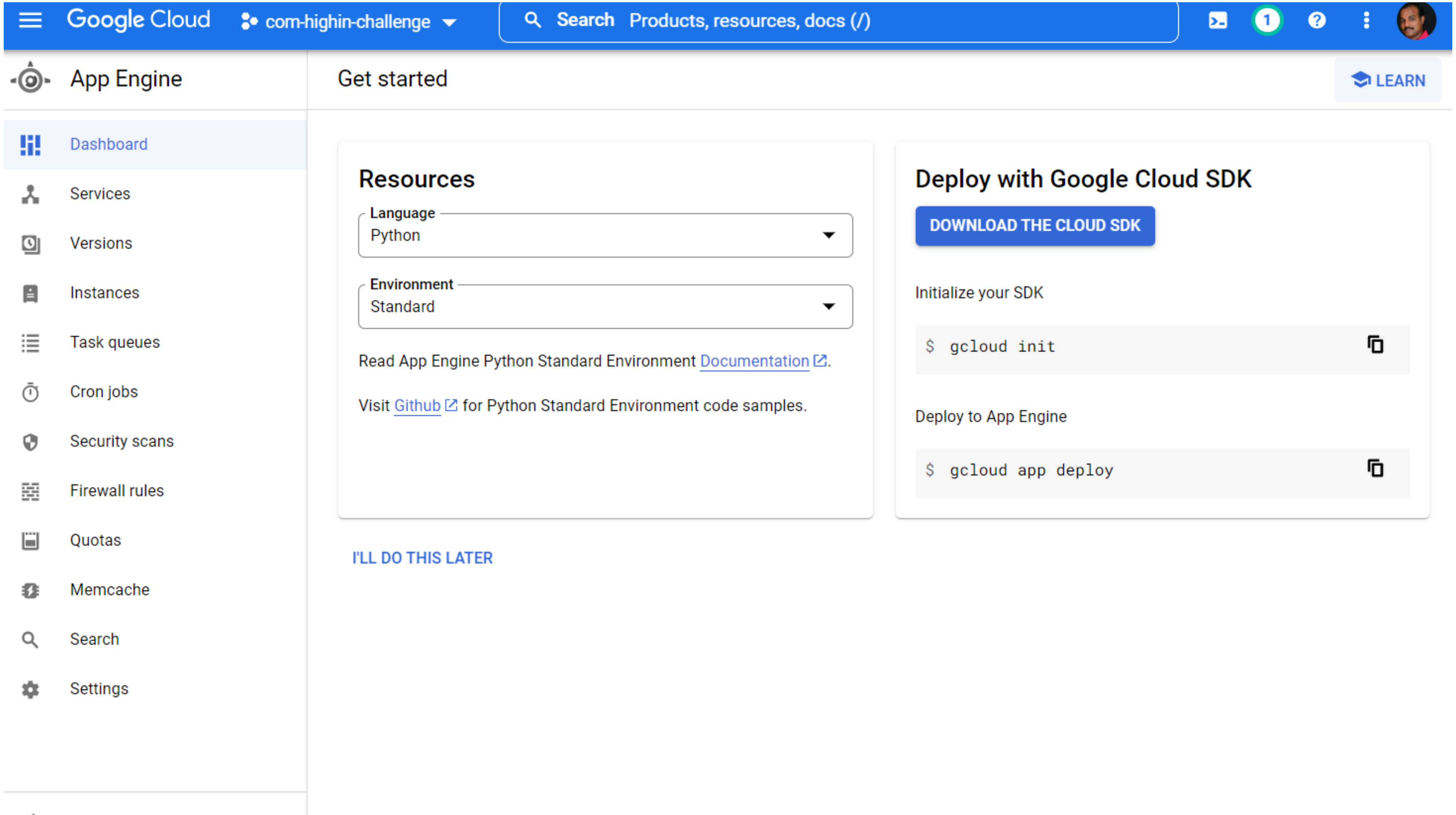
- **Use App Engine when**:

- You don’t want to get troubled for setting up a server.
- You want instant for-free nearly infinite scalability support.
- Your application’s traffic is spiky and rather unpredictable.
- You don't feel like taking care of your own server monitoring tools.
- You need pricing that fits your actual usage and isn't time-slot based (App engine provides pay-per-drink cost model).
- You are able to chunk long tasks into 60 second pieces.
- You are able to work without direct access to local file system.

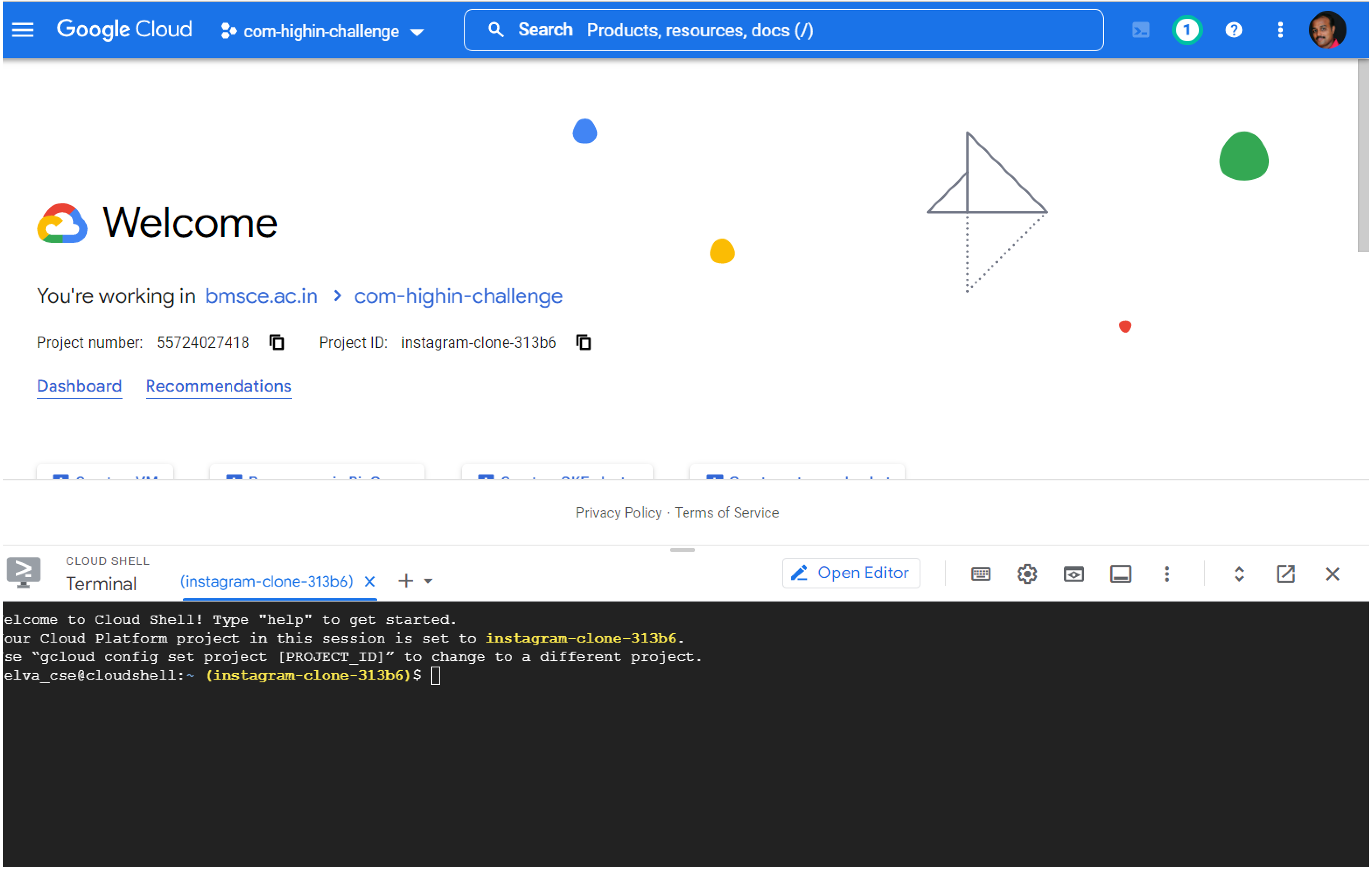
GOOGLE CLOUD PLATFORM

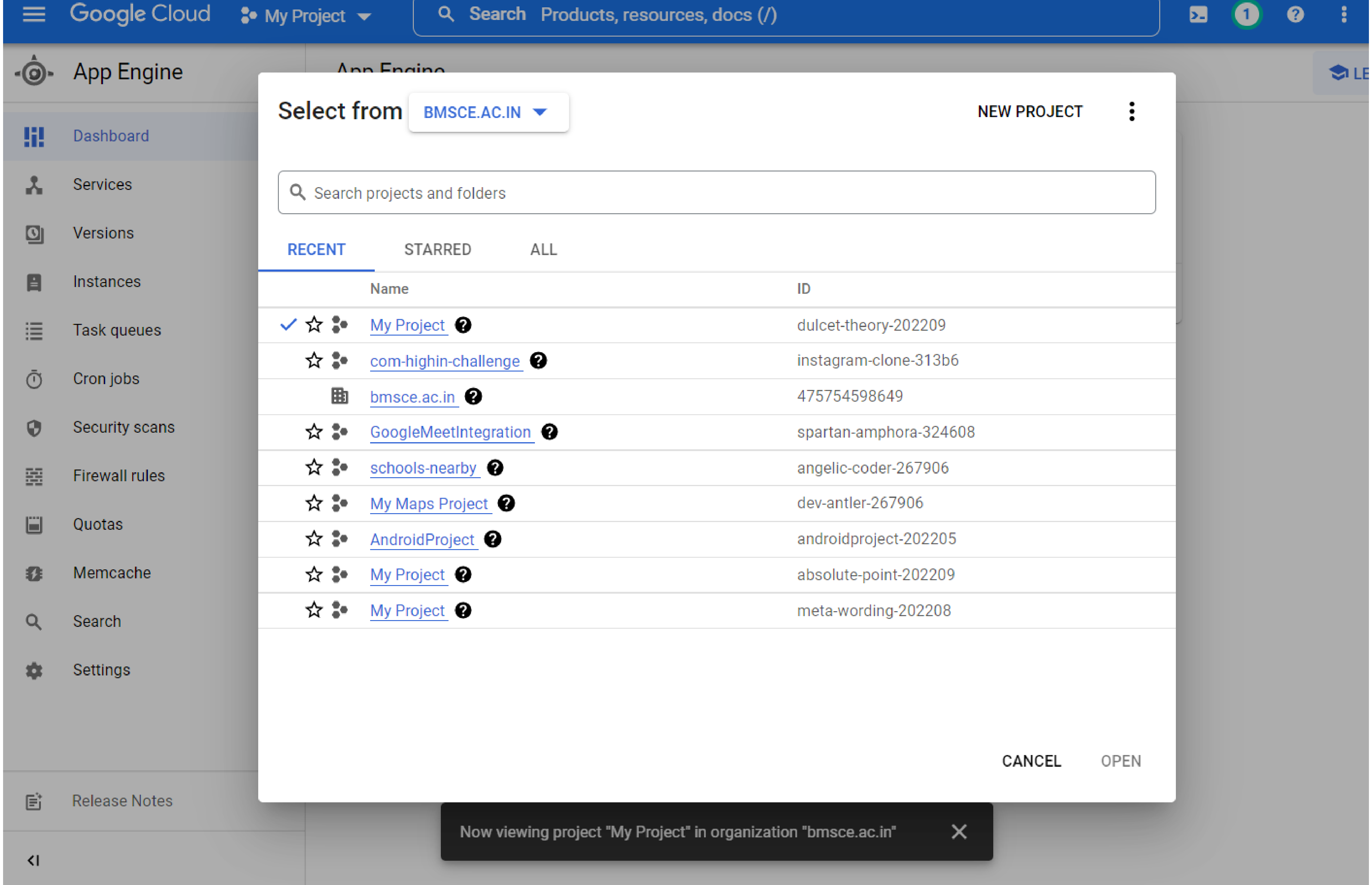


GET STARTED



Google Cloud
com-highin-challenge
Search Products, resources, docs (/)
App Engine
Get started
LEARN
Dashboard
Services
Versions
Instances
Task queues
Cron jobs
Security scans
Firewall rules
Quotas
Memcache
Search
Settings
Resources
Language
Python
Environment
Standard
Read App Engine Python Standard Environment Documentation.
Visit Github for Python Standard Environment code samples.
Deploy with Google Cloud SDK
DOWNLOAD THE CLOUD SDK
Initialize your SDK
$ gcloud init
Deploy to App Engine
$ gcloud app deploy
I'LL DO THIS LATER


Google Cloud
com-highin-challenge
Search Products, resources, docs (/)
Welcome
You're working in bmsce.ac.in > com-highin-challenge
Project number: 55724027418
Project ID: instagram-clone-313b6
Dashboard
Recommendations
Privacy Policy · Terms of Service
CLOUD SHELL
Terminal
(instagram-clone-313b6)
Open Editor
elcome to Cloud Shell! Type "help" to get started.
our Cloud Platform project in this session is set to instagram-clone-313b6.
se "gcloud config set project [PROJECT_ID]" to change to a different project.
elva_cse@cloudshell:~ (instagram-clone-313b6)$


Google Cloud
My Project
Search Products, resources, docs (/)
App Engine
Dashboard
Services
Versions
Instances
Task queues
Cron jobs
Security scans
Firewall rules
Quotas
Memcache
Search
Settings
Release Notes
Select from BMSCE.AC.IN
NEW PROJECT
Search projects and folders
RECENT
STARRED
ALL
Name
ID
My Project dulcet-theory-202209
com-highin-challenge instagram-clone-313b6
bmsce.ac.in 475754598649
GoogleMeetIntegration spartan-amphora-324608
schools-nearby angelic-coder-267906
My Maps Project dev-antler-267906
AndroidProject androidproject-202205
My Project absolute-point-202209
My Project meta-wording-202208
CANCEL
OPEN
Now viewing project "My Project" in organization "bmsce.ac.in"

Create app

1 **Select a location** — 2 Get started

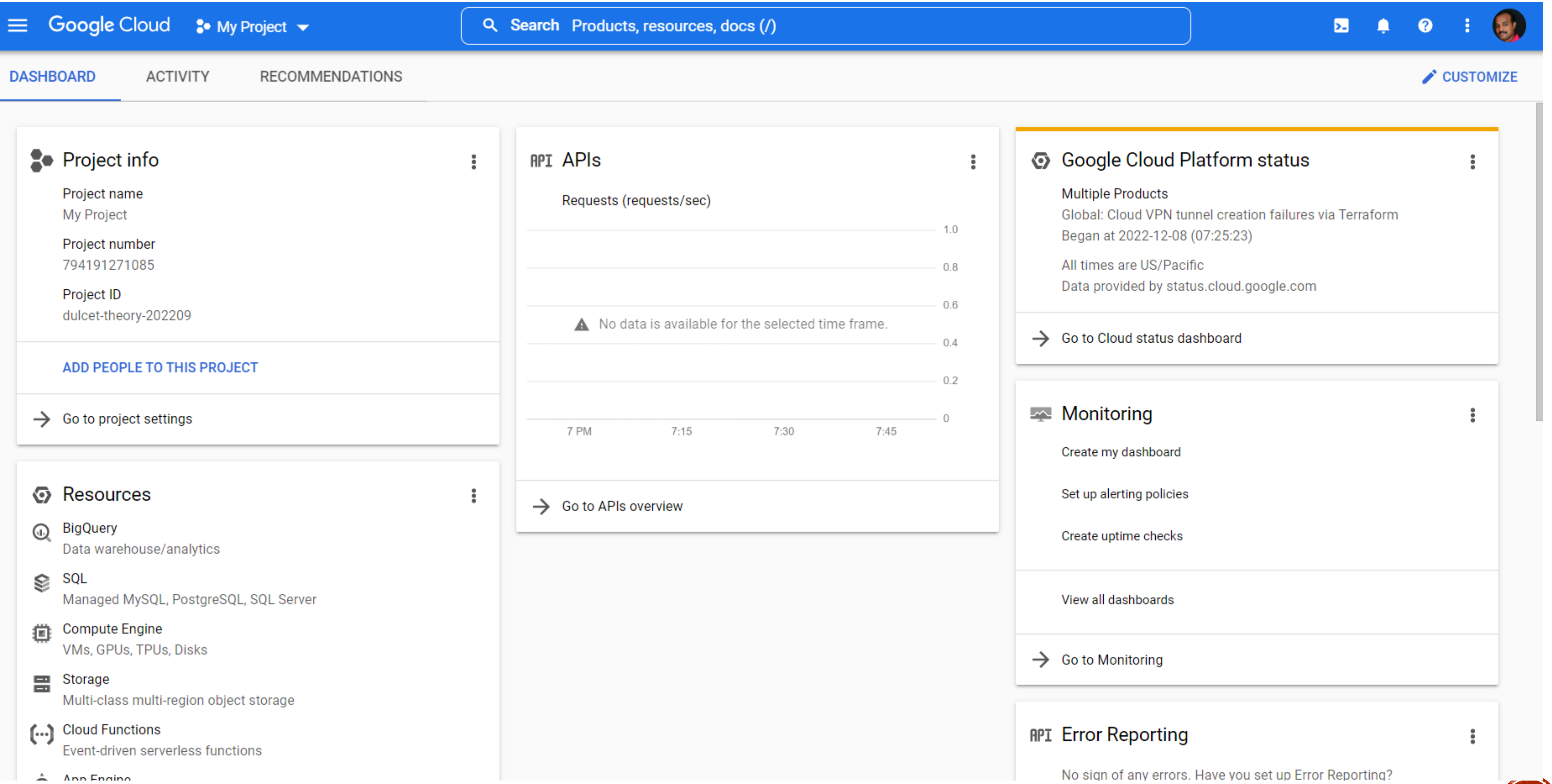
Region

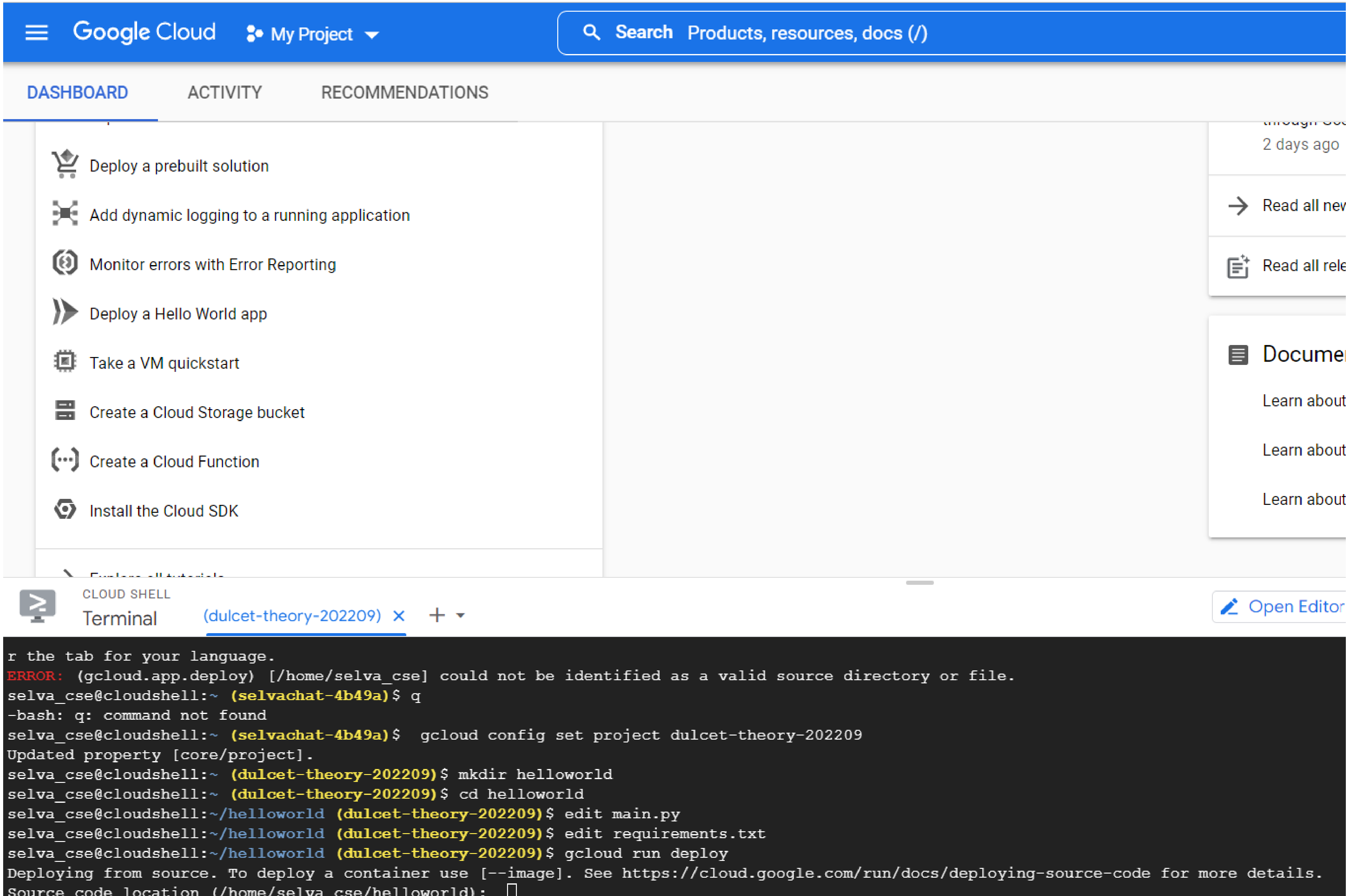
Region is permanent.

Select a region *

asia-south1

NEXT


Google Cloud
My Project
Search Products, resources, docs (/)
DASHBOARD
ACTIVITY
RECOMMENDATIONS
CUSTOMIZE
Project info
Project name
My Project
Project number
794191271085
Project ID
dulcet-theory-202209
ADD PEOPLE TO THIS PROJECT
Go to project settings
APIs
Requests (requests/sec)
1.0
0.8
0.6
0.4
0.2
0
No data is available for the selected time frame.
7 PM
7:15
7:30
7:45
Go to APIs overview
Google Cloud Platform status
Multiple Products
Global: Cloud VPN tunnel creation failures via Terraform
Began at 2022-12-08 (07:25:23)
All times are US/Pacific
Data provided by status.cloud.google.com
Go to Cloud status dashboard
Resources
BigQuery
Data warehouse/analytics
SQL
Managed MySQL, PostgreSQL, SQL Server
Compute Engine
VMs, GPUs, TPUs, Disks
Storage
Multi-class multi-region object storage
Cloud Functions
Event-driven serverless functions
Monitoring
Create my dashboard
Set up alerting policies
Create uptime checks
View all dashboards
Go to Monitoring
Error Reporting
No sign of any errors. Have you set up Error Reporting?


Google Cloud
My Project
Search Products, resources, docs (/)
DASHBOARD
ACTIVITY
RECOMMENDATIONS
Deploy a prebuilt solution
Add dynamic logging to a running application
Monitor errors with Error Reporting
Deploy a Hello World app
Take a VM quickstart
Create a Cloud Storage bucket
Create a Cloud Function
Install the Cloud SDK
2 days ago
Read all nev
Read all rele
Docume
Learn about
Learn about
Learn about
CLOUD SHELL
Terminal
(dulcet-theory-202209)
Open Editor
r the tab for your language.
ERROR: (gcloud.app.deploy) [/home/selva_cse] could not be identified as a valid source directory or file.
selva_cse@cloudshell:~ (selvachat-4b49a)$ q
-bash: q: command not found
selva_cse@cloudshell:~ (selvachat-4b49a)$ gcloud config set project dulcet-theory-202209
Updated property [core/project].
selva_cse@cloudshell:~ (dulcet-theory-202209)$ mkdir helloworld
selva_cse@cloudshell:~ (dulcet-theory-202209)$ cd helloworld
selva_cse@cloudshell:~/helloworld (dulcet-theory-202209)$ edit main.py
selva_cse@cloudshell:~/helloworld (dulcet-theory-202209)$ edit requirements.txt
selva_cse@cloudshell:~/helloworld (dulcet-theory-202209)$ gcloud run deploy
Deploying from source. To deploy a container use [--image]. See https://cloud.google.com/run/docs/deploying-source-code for more details.
Source code location (/home/selva_cse/helloworld):

GAE QUERIES

- The app engine provides a SQL-like query language called GQL.
- GQL queries on entities and their properties.
- Entities manifest as objects in the GAE Python and the Java SDK.
- GQL is quite similar to object-oriented query languages that are used:
  - query, filter, and get model instances and their properties.

EXAMPLE: PYTHON DATASTORE API

```
from google.appengine.ext import db

class Person(db.Model):
  name = db.StringProperty()
  age = db.IntegerProperty()

# We use a unique username for the Entity's key.
amy = Person(key_name='amym', name='Amy', age=48)
amy.put()
Person(key_name='bettyd', name='Betty', age=42).put()
Person(key_name='charliec', name='Charlie', age=32).put()
Person(key_name='charliek', name='Charlie', age=29).put()
Person(key_name='eedna', name='Edna', age=20).put()
Person(key_name='fredm', name='Fred', age=16, parent=amy).put()
Person(key_name='georgemichael', name='George').put()
```

EXAMPLE

- SELECT * FROM Person WHERE age >= 18 AND age <= 35
- SELECT * FROM Person ORDER BY age DESC LIMIT 3
- SELECT * FROM Person WHERE name IN ('Betty', 'Charlie')
- SELECT name FROM Person
- SELECT __key__ FROM Person WHERE age = NULL

EXAMPLE CONTD..

```
import datetime
from google.appengine.ext import db


class Employee(db.Model):
  first_name = db.StringProperty()
  last_name = db.StringProperty()
  hire_date = db.DateProperty()
  attended_hr_training = db.BooleanProperty()


employee = Employee(first_name='Antonio',
                    last_name='Salieri')

employee.hire_date = datetime.datetime.now().date()
employee.attended_hr_training = True

employee.put()
```

KEY_NAME ARGUMENT

```
employee = Employee(key_name='asalieri',
                    first_name='Antonio',
                    last_name='Salieri')

employee.hire_date = datetime.datetime.now().date()
employee.attended_hr_training = True

employee.put()
```

```
employee = Employee(first_name='Antonio',
                    last_name='Salieri')

employee.hire_date = datetime.datetime.now().date()
employee.attended_hr_training = True

employee.put()
```

PARENT ENTITY

```
# Create Employee entity
employee = Employee()
employee.put()

# Set Employee as Address entity's parent directly...
address = Address(parent=employee)

# ...or using its key
e_key = employee.key()
address = Address(parent=e_key)

# Save Address entity to datastore
address.put()
```

RETRIEVING AN ENTITY

- address_k = db.Key.from_path('Employee', 'asalieri', 'Address', 1)
- address = db.get(address_k)

UPDATING AN ENTITY

- **To update an existing entity:**
- Modify the attributes of the object
- Call its put() method.
- The object data overwrites the existing entity.
- The entire object is sent to Datastore with every call to put().

DELETING AN ENTITY

- employee_k = db.Key.from_path('Employee', 'asalieri')
- employee = db.get(employee_k)
- # ...
- employee.delete()

- address_k = db.Key.from_path('Employee', 'asalieri', 'Address', 1)
- db.delete(address_k)

QUERY

```
class Person(db.Model):
  first_name = db.StringProperty()
  last_name = db.StringProperty()
  city = db.StringProperty()
  birth_year = db.IntegerProperty()
  height = db.IntegerProperty()


# Query interface constructs a query using instance methods
q = Person.all()
q.filter("last_name =", "Smith")
q.filter("height <=", max_height)
q.order("-height")


# GqlQuery interface constructs a query using a GQL query string
q = db.GqlQuery("SELECT * FROM Person " +
                "WHERE last_name = :1 AND height <= :2 " +
                "ORDER BY height DESC",
                "Smith", max_height)


# Query is not executed until results are accessed
for p in q.run(limit=5):
  print "%s %s, %d inches tall" % (p.first_name, p.last_name, p.height)
```

EXAMPLE: JAVA DATASTORE API

```
import java.io.IOException;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;

// ...
        DatastoreService ds = DatastoreServiceFactory.getDatastoreService();

        Entity book = new Entity("Book");

        book.setProperty("title", "The Grapes of Wrath");
        book.setProperty("author", "John Steinbeck");
        book.setProperty("copyrightYear", 1939);
        Date authorBirthdate =
            new GregorianCalendar(1902, Calendar.FEBRUARY, 27).getTime();
        book.setProperty("authorBirthdate", authorBirthdate);

        ds.put(book);
```

AMAZON SIMPLEDB

- Amazon SimpleDB is a ready-to-run database alternative to the app engine data store.
- Amazon SimpleDB is a web service for running queries on structured data in real time.
- Amazon SimpleDB requires no schema, automatically indexes your data and provides a simple API for storage and access.
- This eliminates the administrative burden of data modeling, index maintenance, and performance tuning.
- This service works in close conjunction with Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2), collectively providing the ability to store, process and query data sets in the cloud.

SIMPLEDB DATA MODEL

- Domain
- Attributes
- Item

DOMAINS, ATTRIBUTES AND ITEMS

- A domain is like a table.
- An attribute is analogous to a field or column.
- An item is similar to a database row.
- We can change the structure of a domain easily, since it has no schema.
- In addition, attributes are of string type and can contain multiple values.

DATA LOADING

- SimpleDB can be queried in one of the following ways:
- Making RESTful get and post requests over HTTP or HTTPS.
- Making SQL like query using a programming language.

USING REST TO LOAD DATA

- This shows a REST request that puts three attributes and values for an item named Item123 into the domain named MyDomain.

```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Host: sdb.amazonaws.com
Action=PutAttributes
&DomainName=MyDomain
&ItemName=Item123
&Attribute.1.Name=Color&Attribute.1.Value=Blue
&Attribute.2.Name=Size&Attribute.2.Value=Med
&Attribute.3.Name=Price&Attribute.3.Value=0014.99
&AWSAccessKeyId=access_key
&Version=2009-04-15
&Signature=valid_signature
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A01%3A28-07%3A00
```

RESPONSE TO THE REQUEST

```
<PutAttributesResponse>
  <ResponseMetadata>
    <StatusCode>Success</StatusCode>
    <RequestId>f6820318-9658-4a9d-89f8-b067c90904fc</RequestId>
    <BoxUsage>0.0000219907</BoxUsage>
  </ResponseMetadata>
</PutAttributesResponse>
```

TYPES OF QUERIES

- Simple Queries:
- These are the usual queries we perform like in any database:
- Examples: select * from mydomain where Title = 'The Right Stuff'

  select * from mydomain where Year > '1985'
- Range Queries:
- Amazon SimpleDB enables us to execute more than one comparison against attribute values within the same predicate.
- This is most commonly used to specify a range of values.
- select * from mydomain where Year between '1975' and '2008'
- select * from mydomain where (Year > '1950' and Year < '1960') or Year like '193%' or Year = '2007'

QUERIES ON ATTRIBUTES WITH MULTIPLE VALUES

- Amazon SimpleDB allows you to associate multiple values with a single attribute.
- Each attribute is considered individually against the comparison conditions defined in the predicate.
- Example: select * from mydomain where Keyword = 'Book' and Keyword = 'Hardcover'
- Retrieve all items that have the Keyword attribute as both "Book" and "Hardcover."
- Each value is evaluated individually against the predicate expression.

MULTIPLE ATTRIBUTE QUERIES

- Multiple attribute queries work by producing a set of item names from each predicate and applying the intersection operator.
- The intersection operator only returns item names that appear in both result sets.
- select * from mydomain where Keyword = 'Book' intersection Keyword = 'Hardcover’
- The first predicate produces 100, 200, and 50. The second produces 50.
- The result returns 50 counts. The intersection operator returns results that appear in both queries.

QUERY OPTIMIZATION

- Amazon does the query optimization on its own and lets the users to just store the data and query it.
- The 10gb domain limit was created with optimization in mind.
- The user can optimize it themselves by splitting data to multiple domains.
- In order to improve the performance, we can partition our dataset among multiple domains to parallelize queries and have them operate on smaller individual datasets.

PARTITIONING THE DATA

- Applications to parallelize queries:
- Natural Partitions— The data set naturally partitions along some dimension. For example, a University catalog might be partitioned in the "Grad", "UnderGrad" and "Staff" domains. Although we can store all the product data in a single domain, partitioning can improve overall performance.
- High Performance Application— This can be useful when the application requires higher throughput than a single domain can provide.
- Large Data Set—This can be useful when timeout limits are reached because of the data size or query complexity.

AGGREGATION AND JOINS

- If we need aggregation, SimpleDB is not the right solution.
- It is built around the school of thought that the DB is just a key value store, and aggregation should be handled by an aggregation process that writes the results back to the key value store.
- The count() function is recently introduced to the set of functions.
- Since only 2500 data records will be displayed per query we should make sure that the count function does not exceed this range.
- We cannot perform joins in SimpleDB as we can execute a query against a single domain only and this is one of the limitations present in it.

DATA INDEXING

- Amazon does not provide enough information about how indexes are created or managed on SimpleDB, except for the fact that they are automatically created and managed.
- SimpleDB users do not have any control over it.
- Following are some of the salient features of indexes:
- Domain keys are indexed.
- Data are indexed when we enter or modify them in the database.
- SimpleDB takes all data as input and indexes all the attributes.

REPLICATION

- Asynchronous replication is supported.
- Amazon SimpleDB creates and manages multiple geographically distributed replicas of the data automatically.
- Every time we store a data item, multiple replicas are created in different data centers within the region we select.

EXAMPLE

```
from __future__ import print_function
import boto3


def quote(string):
    return string.replace("'", "''").replace('"', '""').replace('`', '``')


def put_attributes(sdb, domain, id, color):
    response = sdb.put_attributes(
        DomainName=domain,
        ItemName=id,
        Attributes=[
            {
                'Name': 'color',
                'Value': color,
                'Replace': True
            },
        ],
    )
    print(response)
```

```
if __name__ == "__main__":
    domain = "TEST_DOMAIN"
    sdb = boto3.client('sdb')
    response = sdb.create_domain(
        DomainName=domain
    )
    print(response)
    response = sdb.list_domains(
    )
    print("Current domains: %s" % response['DomainNames'])
    put_attributes(sdb, domain, "id1", "red")
    put_attributes(sdb, domain, "id2", "redblue")
    put_attributes(sdb, domain, "id3", "blue")
    response = sdb.select(
        SelectExpression='select * from %s where color like "%%%s%%"' %
(domain, quote('blue')),
    )
    print(response)
    response = sdb.delete_domain(
        DomainName=domain
    )
    print(response)
```

https://aws.amazon.com/simpledb/

Amazon SimpleDB

Create Free Account »

Amazon SimpleDB is a highly available NoSQL data store that offloads the work of database administration. Developers simply store and query data items via web services requests and Amazon SimpleDB does the rest.

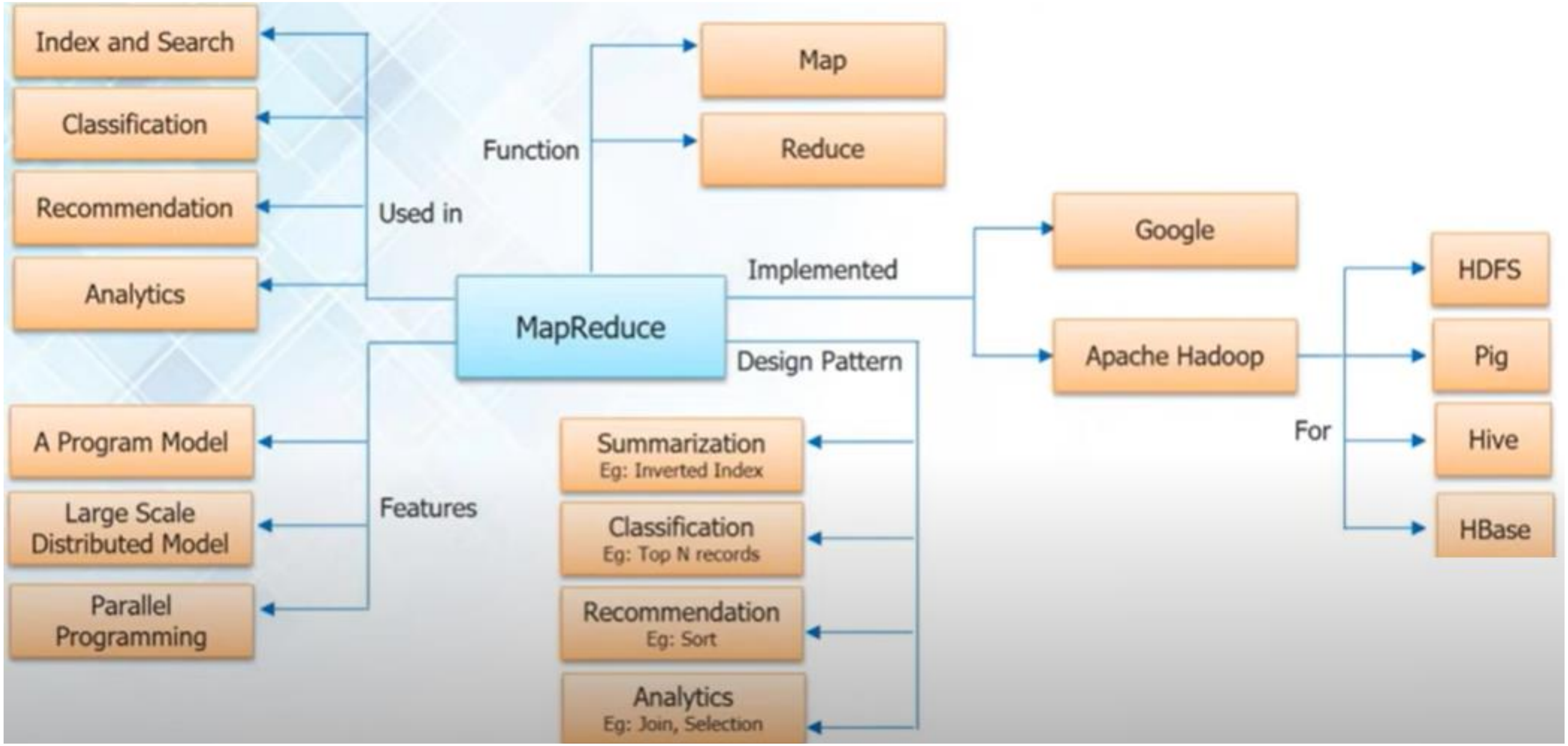
Unbound by the strict requirements of a relational database, Amazon SimpleDB is optimized to provide high availability and flexibility, with little or no administrative burden. Behind the scenes, Amazon SimpleDB creates and manages multiple geographically distributed replicas of your data automatically to enable high availability and data durability. The service charges you only for the resources actually consumed in storing your data and serving your requests. You can change your data model on the fly, and data is automatically indexed for you. With Amazon SimpleDB, you can focus on application development without worrying about infrastructure provisioning, high availability, software maintenance, schema and index management, or performance tuning.

Benefits

Low touch

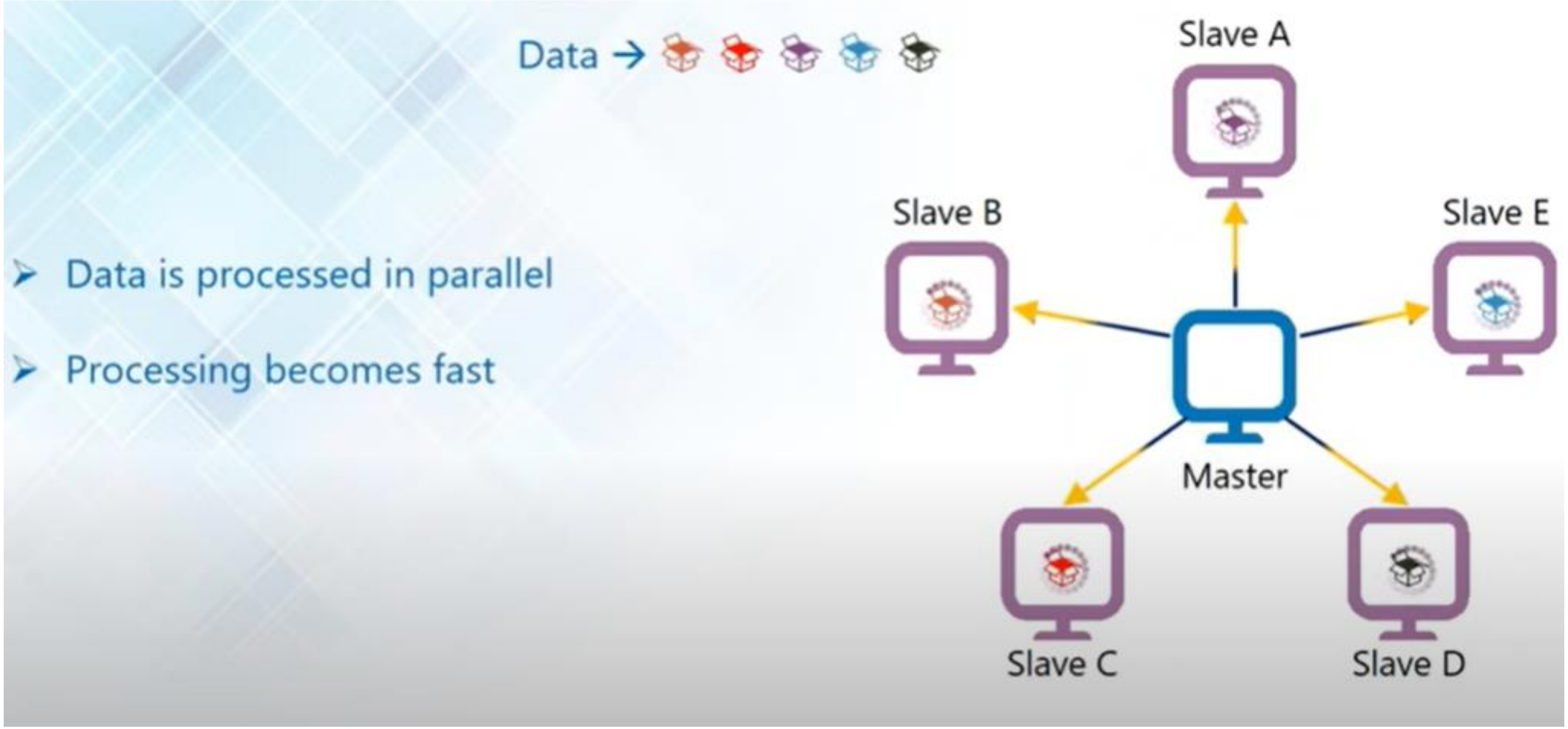
The service allows you to focus fully on value-added application development, rather than arduous and time-consuming database

MAP REDUCE

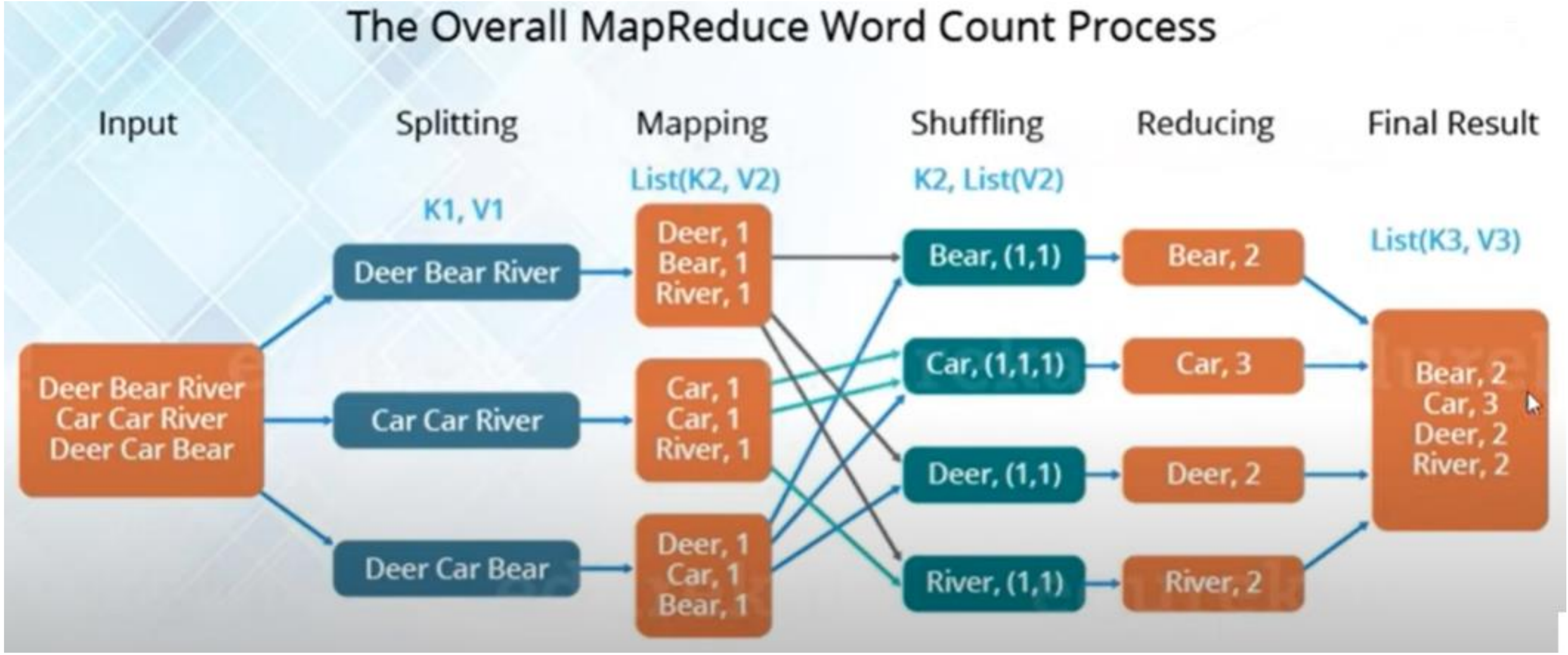


Index and Search
Classification
Recommendation
Analytics
Used in
Function
Map
Reduce
MapReduce
Implemented
Google
Apache Hadoop
For
HDFS
Pig
Hive
HBase
Design Pattern
Features
A Program Model
Large Scale Distributed Model
Parallel Programming
Summarization
Eg: Inverted Index
Classification
Eg: Top N records
Recommendation
Eg: Sort
Analytics
Eg: Join, Selection

ADVANTAGE


Data →
Data is processed in parallel
Processing becomes fast
Slave A
Slave B
Slave E
Master
Slave C
Slave D

EXAMPLE



The Overall MapReduce Word Count Process
Input
Splitting
Mapping
Shuffling
Reducing
Final Result
List(K2, V2)
K2, List(V2)
K1, V1
List(K3, V3)
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, (1,1)
Car, (1,1,1)
Deer, (1,1)
River, (1,1)
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

INPUT/OUTPUT OF MAPREDUCE JOB

Input : a set of (key values) stored in files

key: document ID

value: a list of words as content of each document

Output: a set of (key values) stored in files

key: wordID

value: word frequency appeared in all documents

MapReduce function specification:

map(String input_key, String input_value):

reduce(String output_key, Iterator intermediate_values):

PSEUDO CODE

```
map(String input_key, String input_value):
// input_key: document name
// input_value: document contents
    for each word w in input_value:
      EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
// output_key: a word
// output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
      result  = result + ParseInt(v);
    Emit(output_key, AsString(result));
```

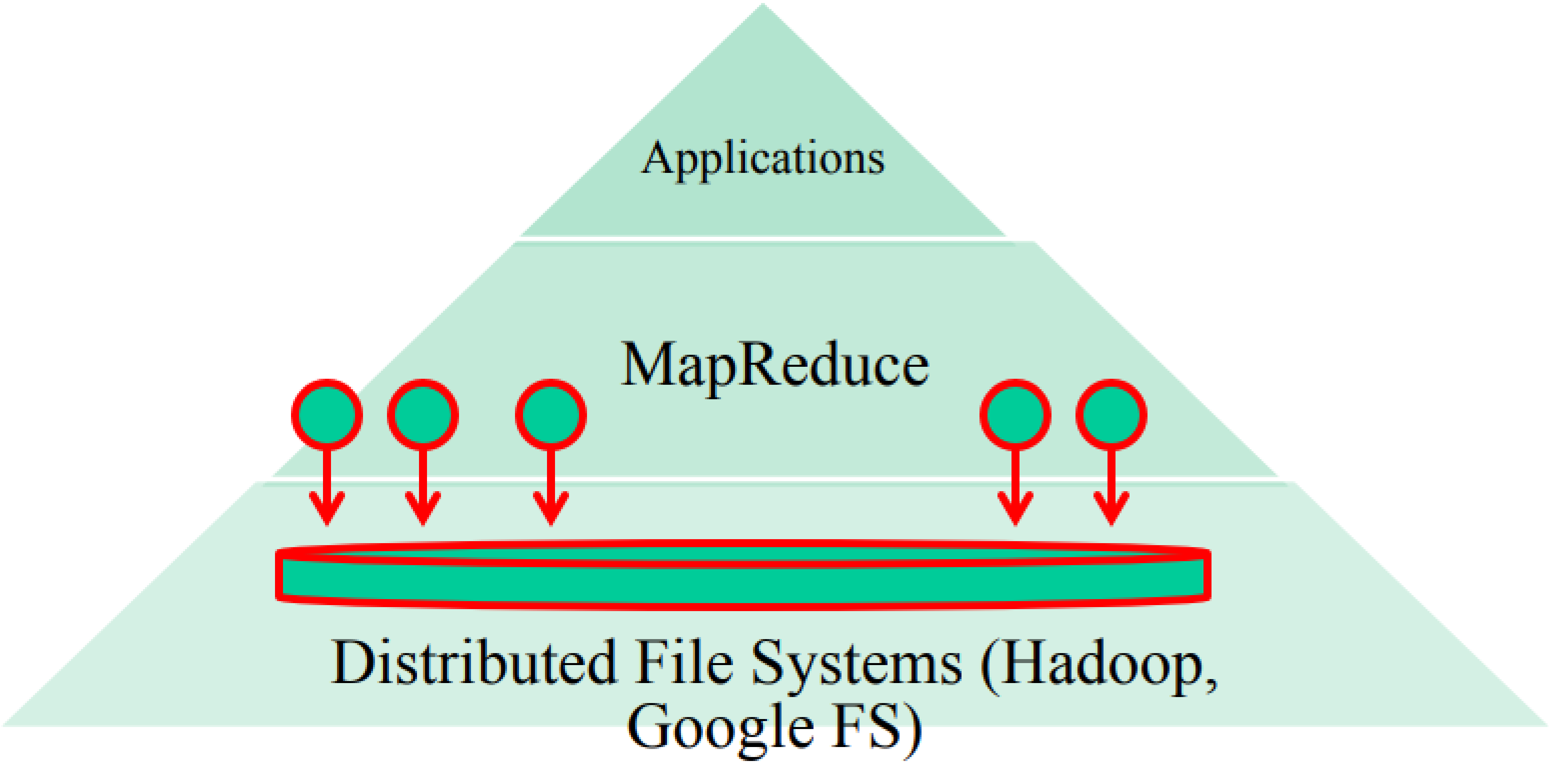
JAVA CODE

```
public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1); // a mapreduce int class
  private Text word = new Text(); //a mapreduce String  class

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException { // key is the offset of
   current record in a file
   StringTokenizer itr = new StringTokenizer(value.toString());
   while (itr.hasMoreTokens()) { // loop for each token
          word.set(itr.nextToken());  //convert from string to token
          context.write(word, one);  // emit (key,value) pairs for reducer
```

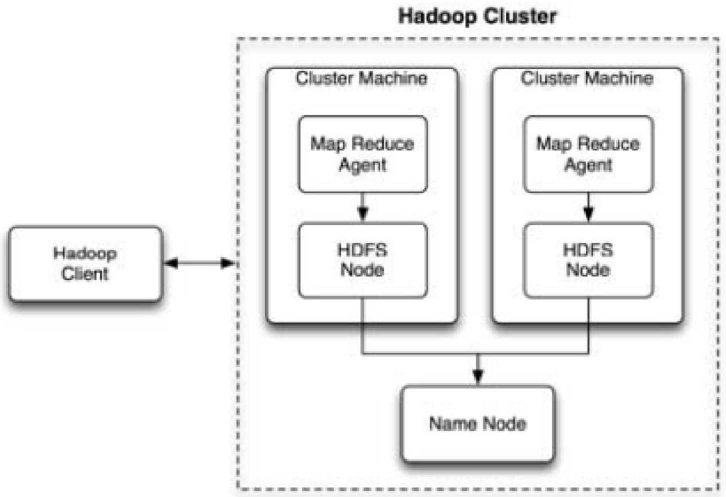
JAVA CODE CONTINUED..

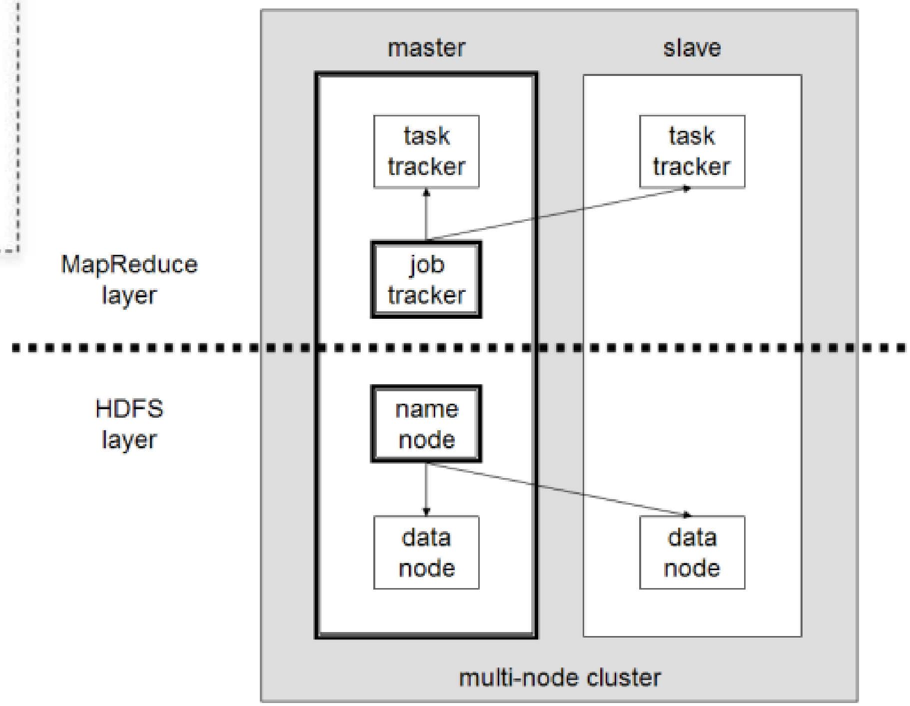
```
public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);  //convert  "int" to IntWritable
      context.write(key, result); //emit the final key-value result
    }
```

SYSTEMS SUPPORT FOR MAPREDUCE



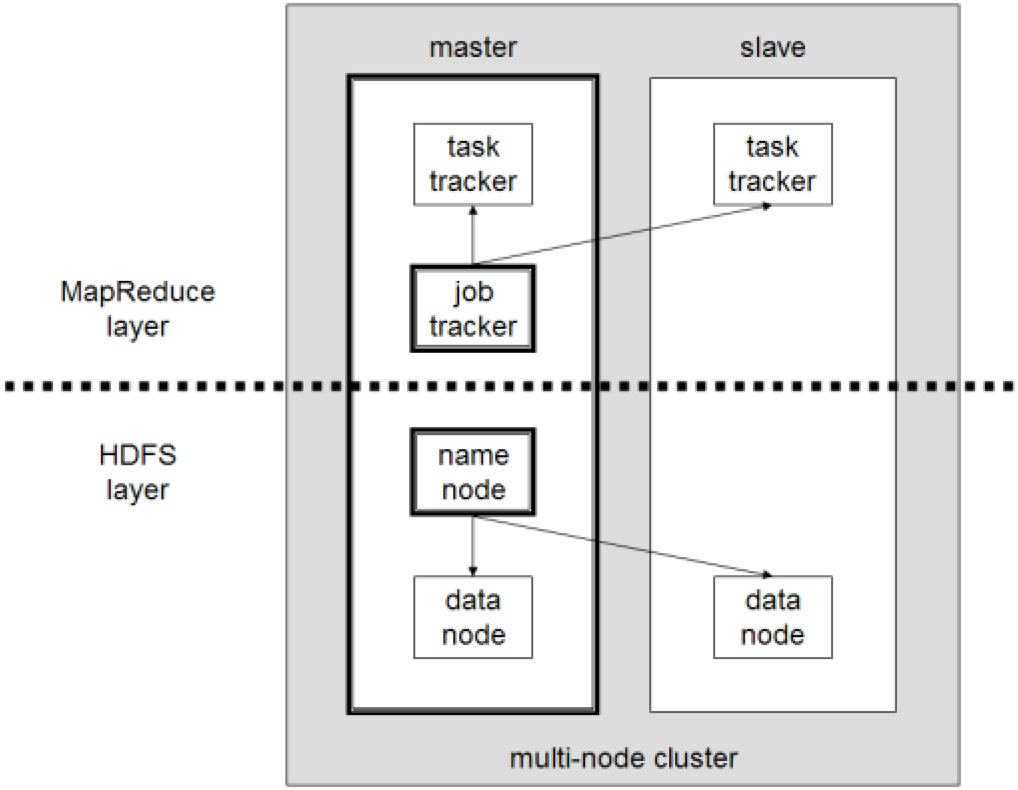
HADOOP DFS WITH MAPREDUCE



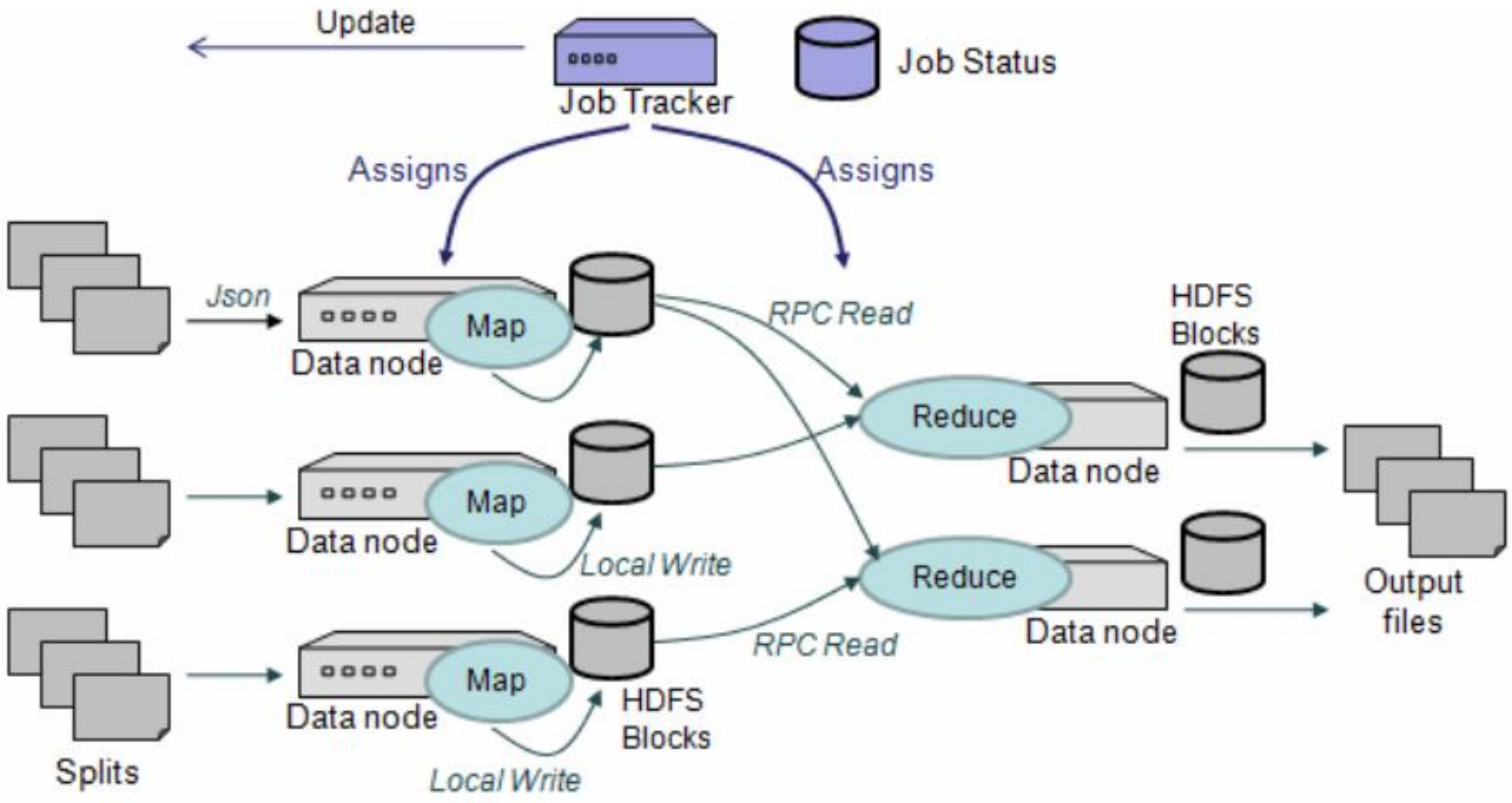


DEMONS FOR HADOOP/MAPREDUCE

- **Following demons must be running (use jps to show these Java processes)**
- **Hadoop**
  - Name node (master)
  - Secondary name node
  - data nodes
- **Mapreduce**
  - Task tracker
  - Job tracker



MAPREDUCE EXECUTION FLOW



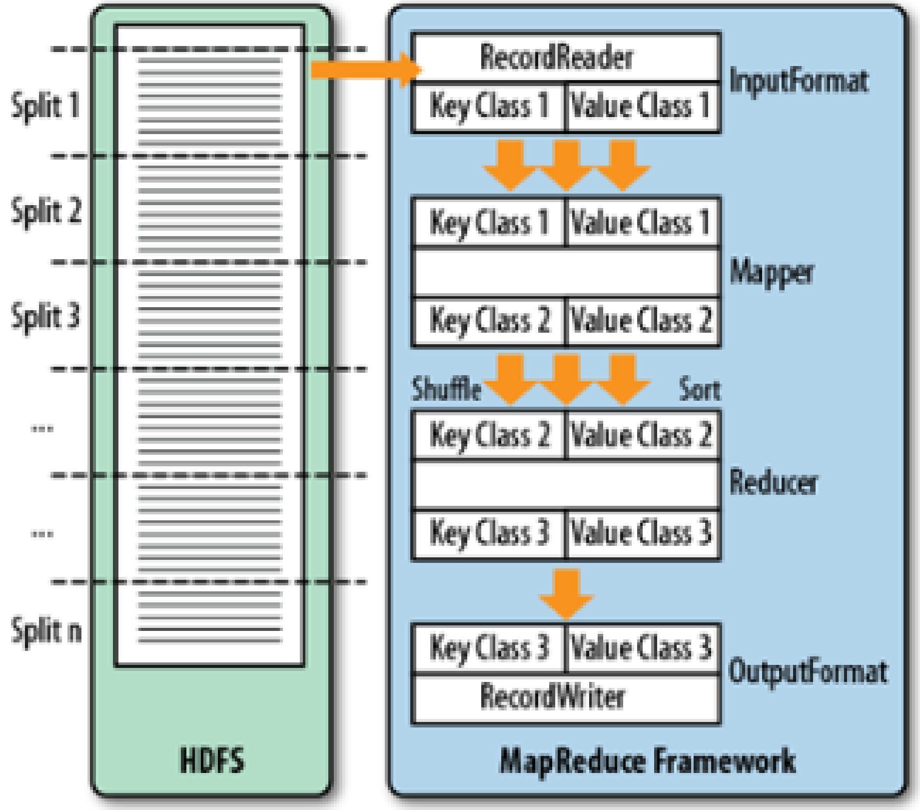
MAPREDUCE: EXECUTION DETAILS

- **Input reader**
  - Divide input into splits, assign each split to a Map task
- **Map task for data parallelism**
  - Apply the Map function to each record in the split
  - Each Map function returns a list of (key, value) pairs
- **Shuffle/Partition and Sort**
  - Shuffle distributes sorting & aggregation to many reducers
  - All records for key *k* are directed to the same reduce processor
  - Sort groups the same keys together, and prepares for aggregation
- **Reduce task for data parallelism**
  - Apply the Reduce function to each key
  - The result of the Reduce function is a list of (key, value) pairs

MAPREDUCE WITH HBASE

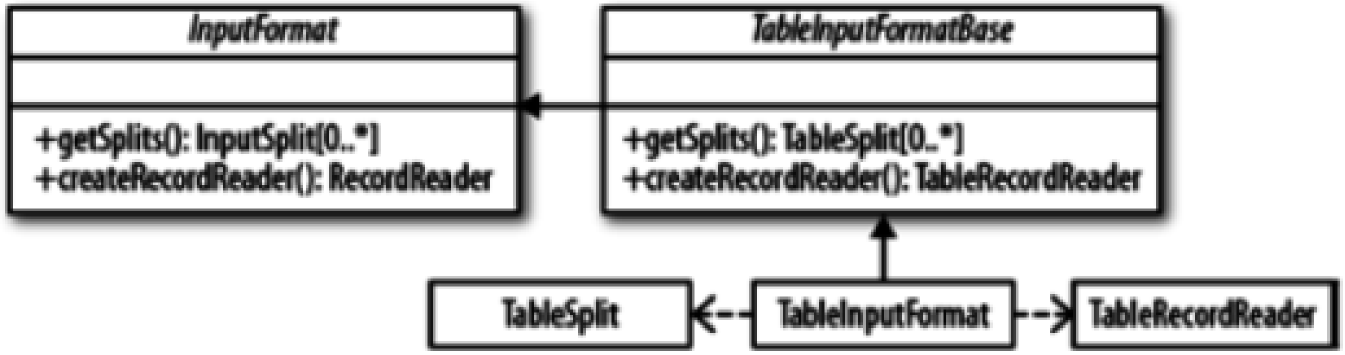
- HBase provides a TableInputFormat, to which you provided a table scan, that splits the rows resulting from the table scan into the regions in which those rows reside.
- The map process is passed an ImmutableBytesWritable that contains the row key for a row and a Result that contains the columns for that row.
- The map process outputs its key/value pair based on its business logic in whatever form makes sense to your application.
- The reduce process builds its results but emits the row key as an ImmutableBytesWritable and a Put command to store the results back to HBase.
- Finally, the results are stored in HBase by the HBase MapReduce infrastructure.

HBASE MAPREDUCE INTEGRATION



IMPLEMENTATION OF MAPREDUCE

- Input format
- First it splits the input data, and then it returns a RecordReader instance that defines the classes of the key and value objects, and provides a next() method that is used to iterate over each input record.



IMPLEMENTATION OF MAPREDUCE

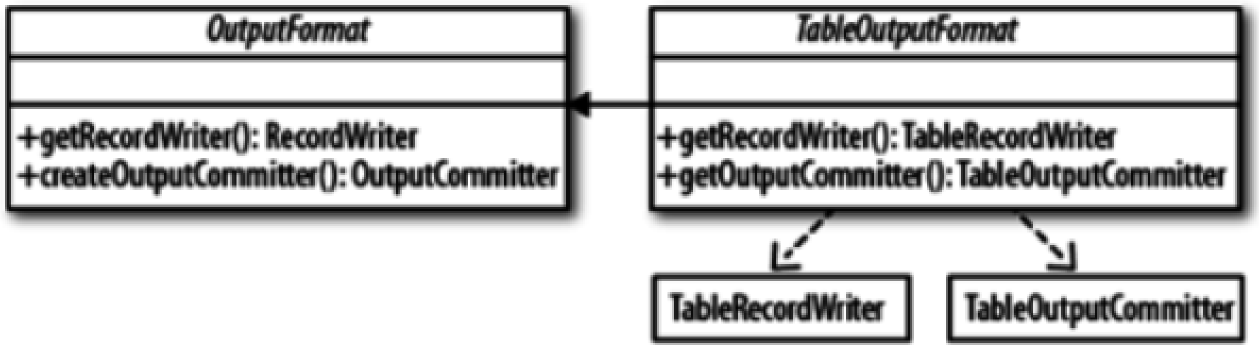
- Mapper
- In this step, each record read using the RecordReader is processed using the map() method.

IMPLEMENTATION OF MAPREDUCE

- **Reducer**
- The Reducer stage and class hierarchy is very similar to the Mapper stage. This time we get the output of a Mapper class and process it after the data has been shuffled and sorted.

IMPLEMENTATION OF MAPREDUCE

- OutputFormat
- The final stage is the OutputFormat class, and its job is to persist the data in various locations. There are specific implementations that allow output to files, or to HBase tables in the case of the TableOutputFormat class. It uses a TableRecord Writer to write the data into the specific HBase output table.



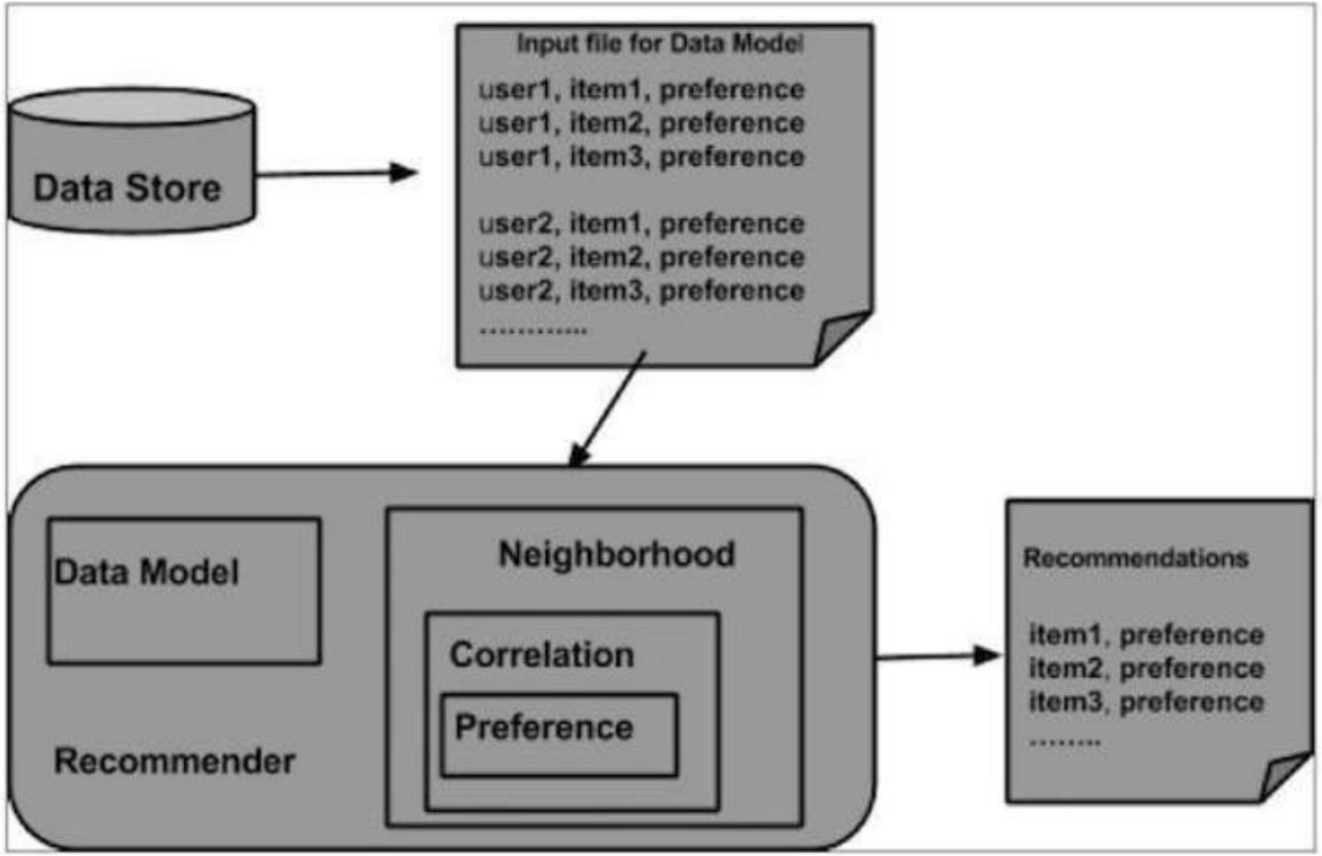
APACHE MAHOUT

- Apache Mahout is a project of the Apache Software Foundation which is implemented on top of Apache Hadoop and uses the MapReduce paradigm.
- It is also used to create implementations of scalable and distributed machine learning algorithms that are focused in the areas of
- Clustering,
- Collaborative filtering and
- Classification.
- Mahout contains Java libraries for common math algorithms and operations focused on statistics and linear algebra, as well as primitive Java collections.

COMPONENTS OF MAHOUT

- To build a recommender engine mahout provides the following components:
- DataModel
- UserSimilarity
- ItemSimilarity
- UserNeighborhood
- Recommender

ARCHITECTURE OF RECOMMENDER ENGINE



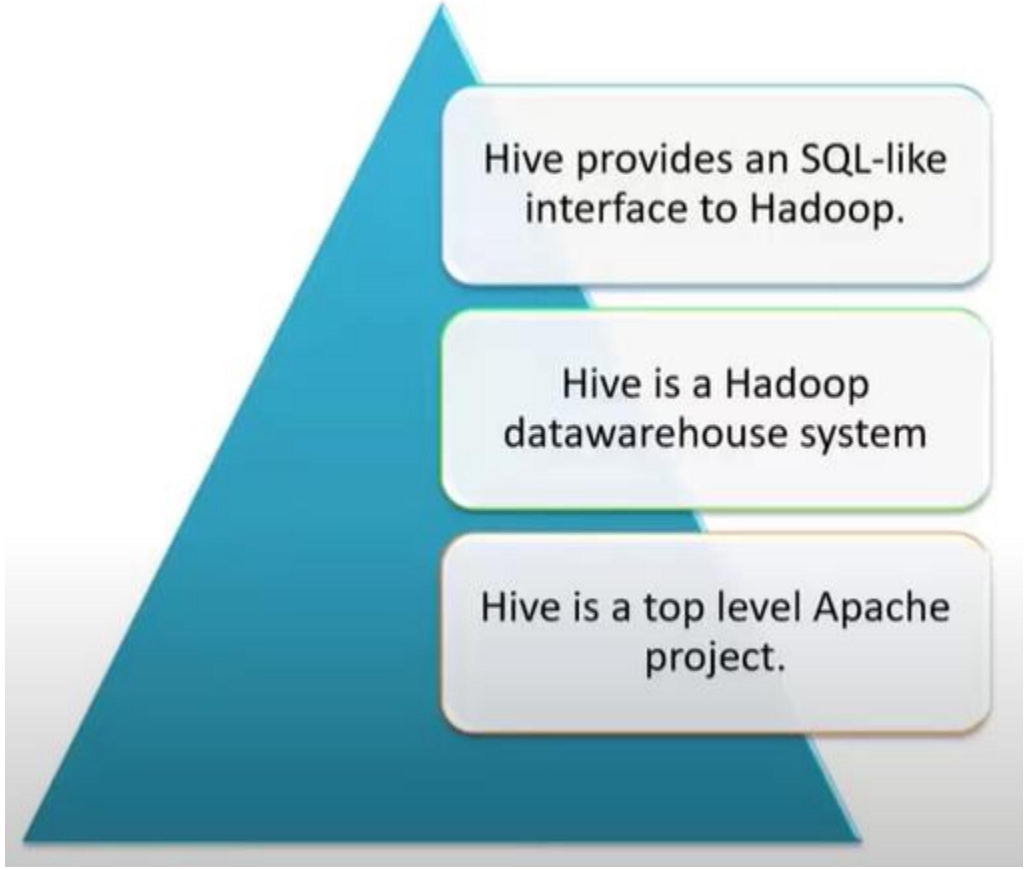
BUILDING A RECOMMENDER USING MAHOUT

- DataModel datamodel = new FileDataModel(new File("input file"));
- UserSimilarity similarity = new PearsonCorrelationSimilarity(datamodel);
- UserNeighborhood neighborhood = new ThresholdUserNeighborhood(3.0, similarity, model);
- UserBasedRecommender recommender = new GenericUserBasedRecommender(model, neighborhood, similarity);
- List<RecommendedItem> recommendations = recommender.recommend(2, 3);
- for (RecommendedItem recommendation : recommendations) {
-   System.out.println(recommendation);
- }

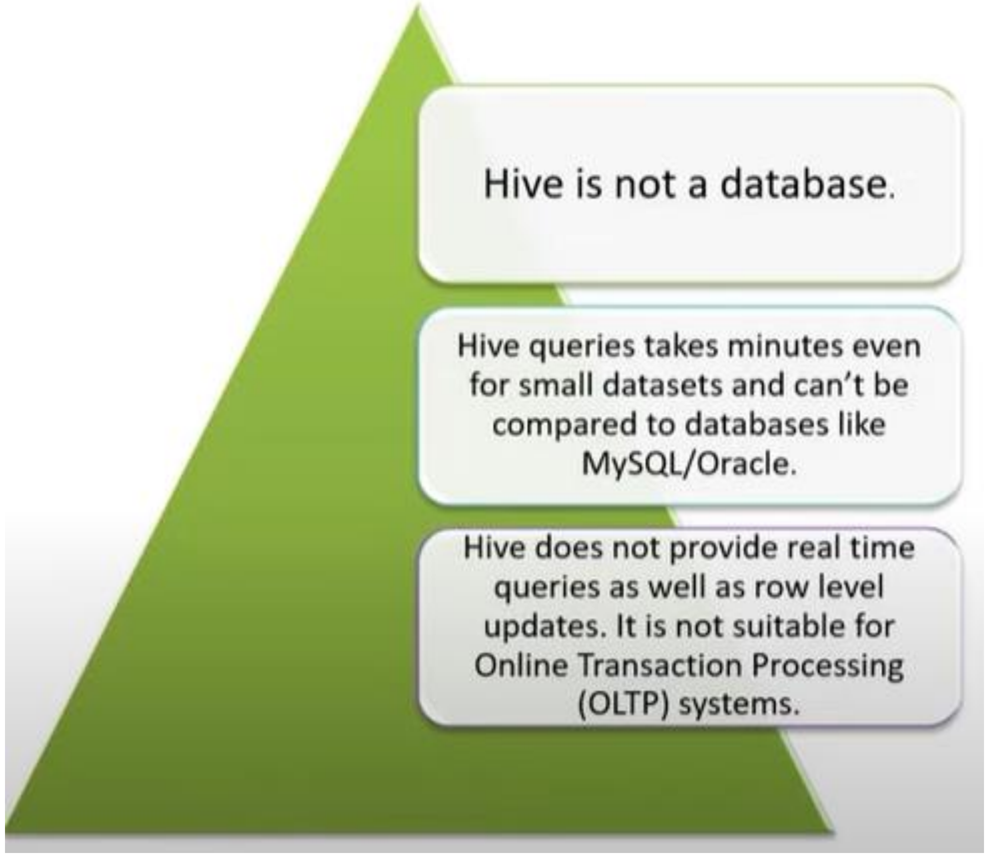
```
public class Recommender {
  public static void main(String args[]){
    try{
      //Creating data model
      DataModel datamodel = new FileDataModel(new File("data")); //data

      //Creating UserSimilarity object.
      UserSimilarity usersimilarity = new PearsonCorrelationSimilarity(datamodel);

      //Creating UserNeighbourHHood object.
      UserNeighborhood userneighborhood = new ThresholdUserNeighborhood(3.0, usersimilarity, datamodel);

      //Create UserRecomender
      UserBasedRecommender recommender = new GenericUserBasedRecommender(datamodel,
userneighborhood, usersimilarity);

      List<RecommendedItem> recommendations = recommender.recommend(2, 3);

      for (RecommendedItem recommendation : recommendations) {
        System.out.println(recommendation);
      }

    }catch(Exception e){}

  }
 }
```

BIGDATA WITH HIVE

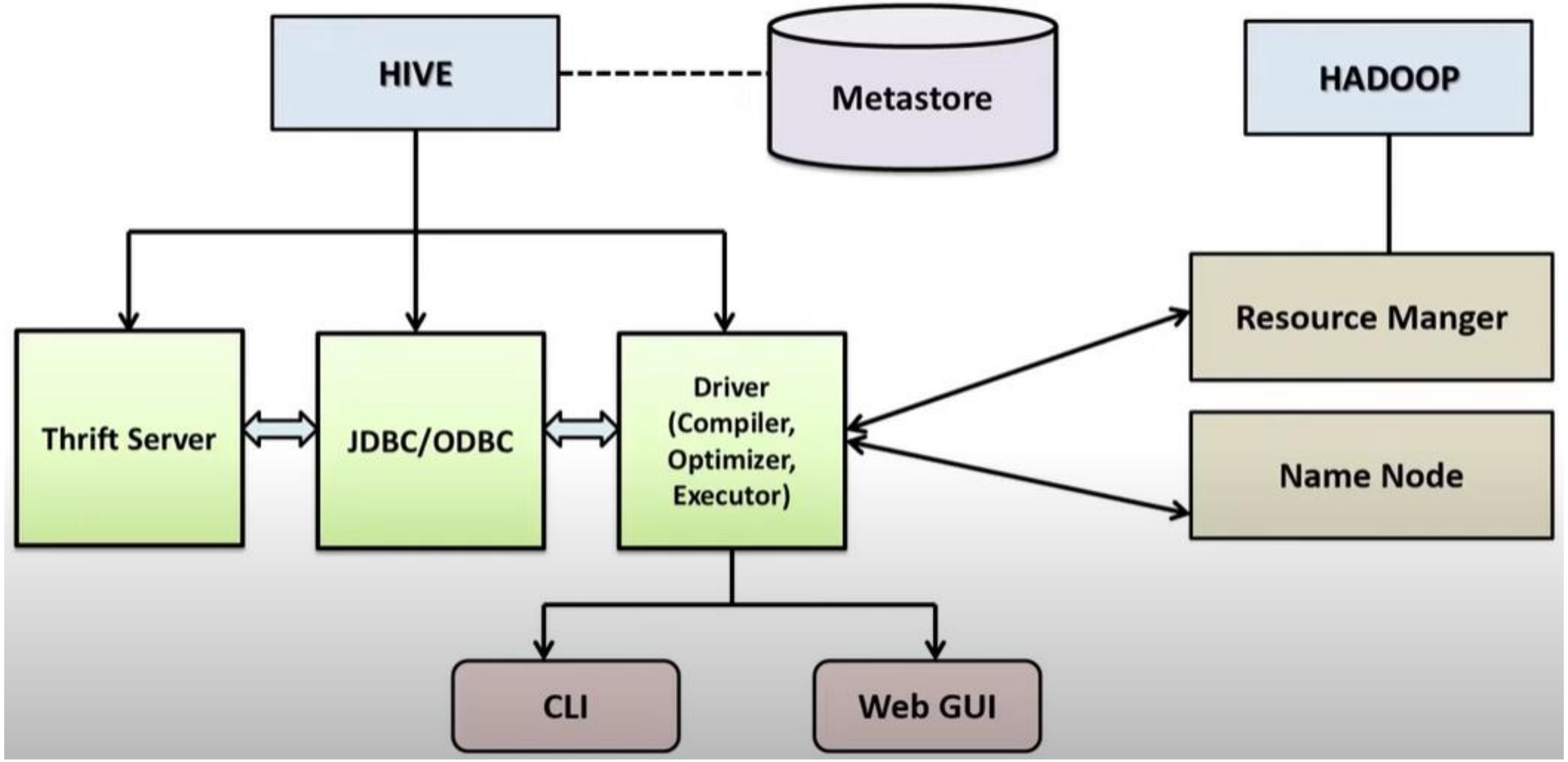
- What is HIVE?



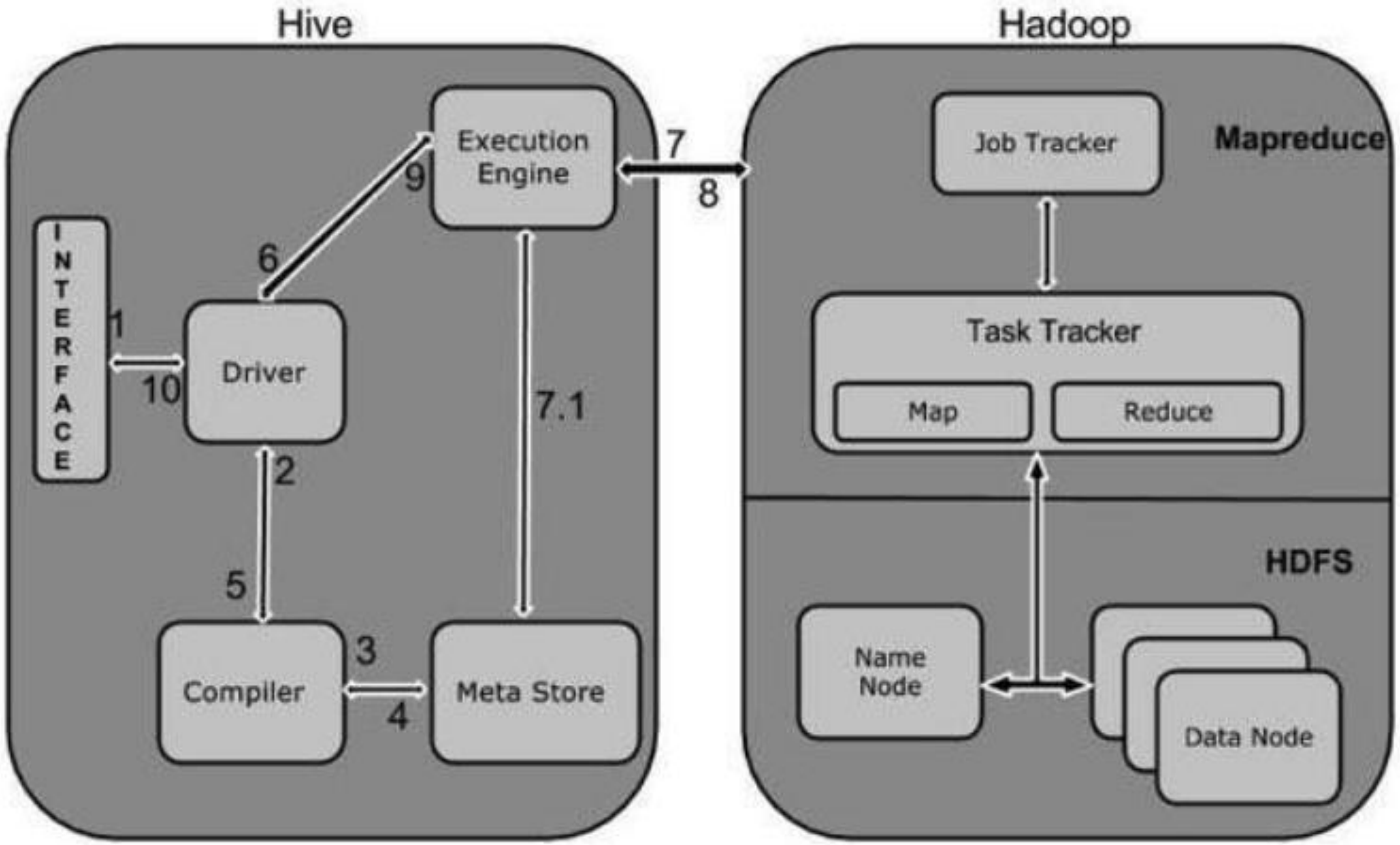
WHAT HIVE IS NOT?



HIVE ARCHITECTURE

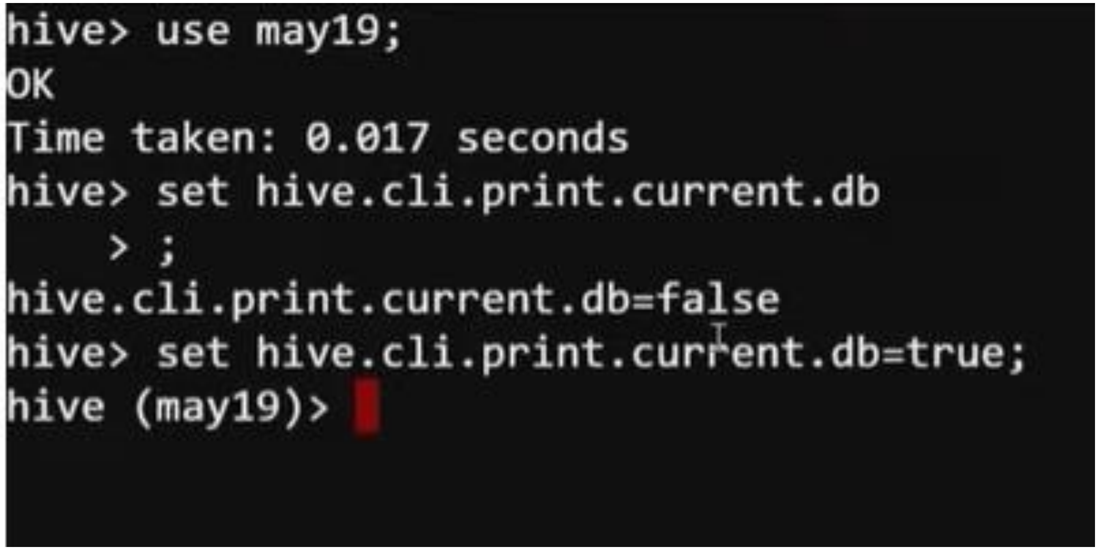


WORKING OF HIVE



HQL COMMANDS

- Create database mydb;
- Show databases;
- Use mydb;



HQL COMMANDS

- Create table customer(custId INT, custName String, mobile INT)

  row format delimited

  fields terminated by ‘,’;
- Load data local inpath ‘c:/temp/cust.txt’ into table customer;
- Select * from customer;
- Select count(*) from customer;

```
hive (may19)> select count(*) from txnrecords;
Query ID = gl_faculty_greatlearning_20180519121919_06bd96da-5fb9-47bb-adec-c809700a32d6
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1518113766572_8510, Tracking URL = http://ip-20-0-21-94.ap-south-1.compute.internal:8088/pro
13766572_8510/
Kill Command = /opt/cloudera/parcels/CDH-5.11.2-1.cdh5.11.2.p0.4/lib/hadoop/bin/hadoop job  -kill job_151811376
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-05-19 12:19:23,126 Stage-1 map = 0%,  reduce = 0%
2018-05-19 12:19:25,289 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.38 sec
MapReduce Total cumulative CPU time: 3 seconds 380 msec
Ended Job = job_1518113766572_8510
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.38 sec   HDFS Read: 8848463 HDFS Write: 541852 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 380 msec
OK
50000
Time taken: 12.612 seconds, Fetched: 1 row(s)
```

HQL COMMANDS

- Create table out(custId INT, custName String, amount INT, product String)
  row format delimited
  fields terminated by ‘,’;
- Insert overwrite table out
  select a.custId, a.custName, b.amount, b.product
  from customer a JOIN products b ON a.custId = b.custId;
- Select * from out limit 5;

HQL COMMANDS

- Insert overwrite table out1

  select *, case

  when age<30 then ‘young’

  when age>=30 and age<50 ‘middle’

  when age>=50 ‘old’

  else ‘others’

  end

  from out;

- Insert overwrite table out2

  select level, sum(amount) from out1 group by level;

HQL JOIN AND SUBQUERIES

- hive> SELECT ratings.userid, ratings.rating, ratings.tstamp, movies.title, users.gender
- > FROM ratings JOIN movies ON (ratings.movieid = movies.movieid)
- > JOIN users ON (ratings.userid = users.userid)
- > LIMIT 5;

- hive> SELECT user_id, rating_count
- > FROM (SELECT ratings.userid as user_id, COUNT(ratings.rating) as rating_count
- > FROM ratings
- > WHERE ratings.rating = 5
- > GROUP BY ratings.userid) top_raters
- > WHERE rating_count > 15;

HQL EXPLAIN PLAN

- An explain plan in Hive reveals the MapReduce behind a query.
- hive> EXPLAIN SELECT COUNT(*) FROM ratings
- > WHERE movieid = 1 and rating = 5;
- OK
- ABSTRACT SYNTAX TREE:
- (TOK_QUERY (TOK_FROM (TOK_TABREF ratings))
- (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE))
- (TOK_SELECT (TOK_SELEXPR (TOK_FUNCTIONSTAR COUNT)))
- (TOK_WHERE (and (= (TOK_TABLE_OR_COL movieid) 1)
- (= (TOK_TABLE_OR_COL rating) 5)))))
- STAGE DEPENDENCIES:
- Stage-1 is a root stage

THANK YOU