

Real-Time Sentiment Analysis System

CSYE7200 Team 5

Ruifeng Cui 001029411

Anxi Liu 001029165

Ashish Roy 001044804

A decorative background at the top of the slide featuring a complex network of interconnected nodes and lines, resembling a data graph or neural network, in shades of blue and grey.

Outline

- Project Overview
- Project Web UI
- Project Implementation
 - Data Ingestion
 - Data Processing and Storage
 - AWS Deploy
- Project Summary

The background of the slide is a complex network of thin, light blue lines connecting numerous small, semi-transparent blue dots. The dots are scattered across the entire frame, creating a dense, web-like pattern that suggests a global or interconnected network.

Project Overview (same as the proposal)

The background of the slide features a complex network of thin, light blue lines connecting numerous small, semi-transparent blue dots of varying sizes. This pattern is most dense at the top and fades slightly towards the bottom, creating a sense of a global or digital network.

Project Overview

This real-time sentiment analysis system will achieve the sentiment analysis for the streaming tweets which are related to the input keyword by users and will show the sentiment analysis results in the visualization chart as the feedback to the user.

This system consists of data ingestion module, data processing and storage module, web UI and this system will be deployed on the cloud.

Project Overview

Use Case

Real users input a keyword which they want to know. This system will ingest the real-time tweets which is relative to this keyword on the Twitter. Then, system will process batches of data in the real-time and perform the sentiment analysis by using machine learning algorithms. Finally, this system will display the sentiment analysis result according to this keyword for the user.

If users want to know what people think and attitudes towards a certain university or a certain team on Twitter in real-time, they can just type the name of that university or that team. Then, in seconds, this system will do the processing and computing. Finally, system will return the analysis results in the charts for users. Users can intuitively see the attitude of Twitter users about this team or school.

A background image featuring a complex network graph with numerous blue and grey nodes connected by thin lines, creating a web-like pattern.

Project Overview

Methodology

data ingestion : Twitter API + Kafka

data processing: Spark Streaming + Spark MLlib

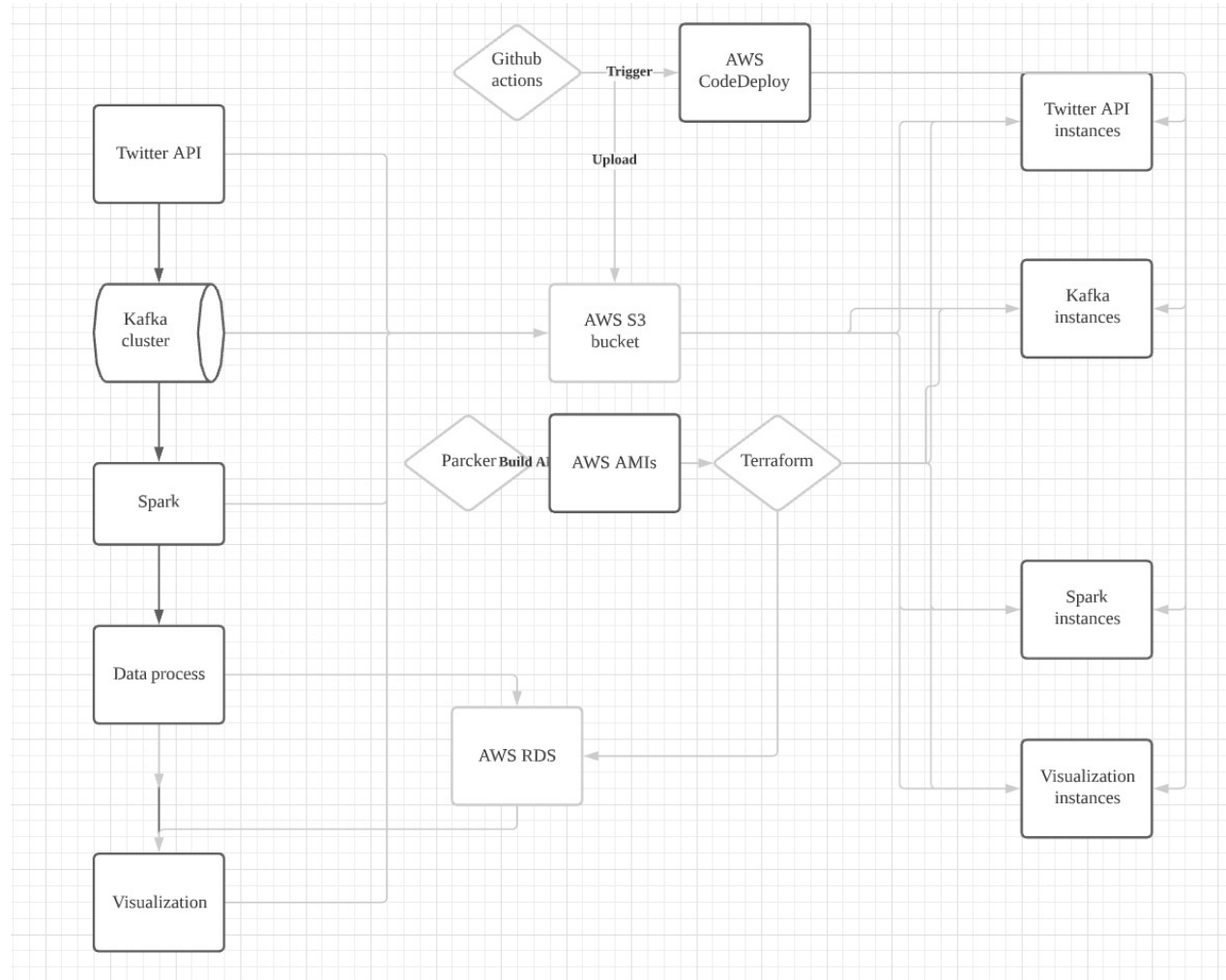
data storage: HDFS + MySQL (Maybe a little bit change when we implement)

data visualization: web or visualization tools

deploying and resource management: AWS

Project Overview

Architecture and Workflow





Project Web UI

Project Web UI

AWS Server

The screenshot shows the AWS Management Console interface for the 'Instances' page. The top navigation bar includes the AWS logo, a search bar, and the current region 'N. Virginia' with the account ID 'CSYE7200-TEAMS'. The left sidebar lists various services, with 'Instances' selected. The main content area shows a table of 6 running EC2 instances. Below the table is a 'Select an instance' modal.

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elast
<input type="checkbox"/>	zookeeper	i-0a4a558a4b0dc7f65	Running	t2.medium	2/2 checks passed	No alarms	us-east-1a	ec2-3-81-98-238.comp...	3.81.98.238	-
<input type="checkbox"/>	kafka	i-0e1b25d570e494b8d	Running	t2.medium	2/2 checks passed	No alarms	us-east-1a	ec2-54-164-187-104.co...	54.164.187.104	-
<input type="checkbox"/>	kafka	i-05dcafd6413e4ca28	Running	t2.medium	2/2 checks passed	No alarms	us-east-1a	ec2-18-234-206-192.co...	18.234.206.192	-
<input type="checkbox"/>	kafka	i-01a1fdc415ad6854e	Running	t2.medium	2/2 checks passed	No alarms	us-east-1a	ec2-107-22-63-116.co...	107.22.63.116	-
<input type="checkbox"/>	spark	i-0429c72c40c8c8c8f	Running	t2.medium	2/2 checks passed	No alarms	us-east-1a	ec2-54-224-22-247.co...	54.224.22.247	-
<input type="checkbox"/>	webapp	i-0fd83df623be997a9	Running	t2.medium	2/2 checks passed	No alarms	us-east-1a	ec2-34-201-137-152.co...	34.201.137.152	-

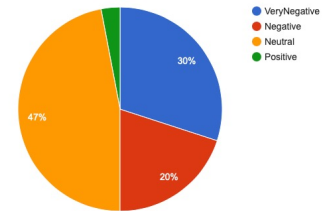
Select an instance

© 2021, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

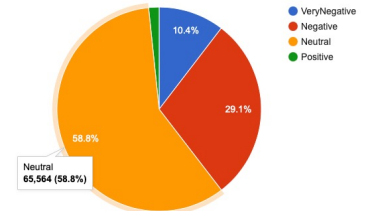
Project Web UI Screen Shot

Tweet Sentiment Analysis

Sentiment Result



Retweet Count



Project Web UI URL

AWS Services:

[https://console.aws.amazon.com/ec2/v2/home
?region=us-east-1](https://console.aws.amazon.com/ec2/v2/home?region=us-east-1) - Instances:

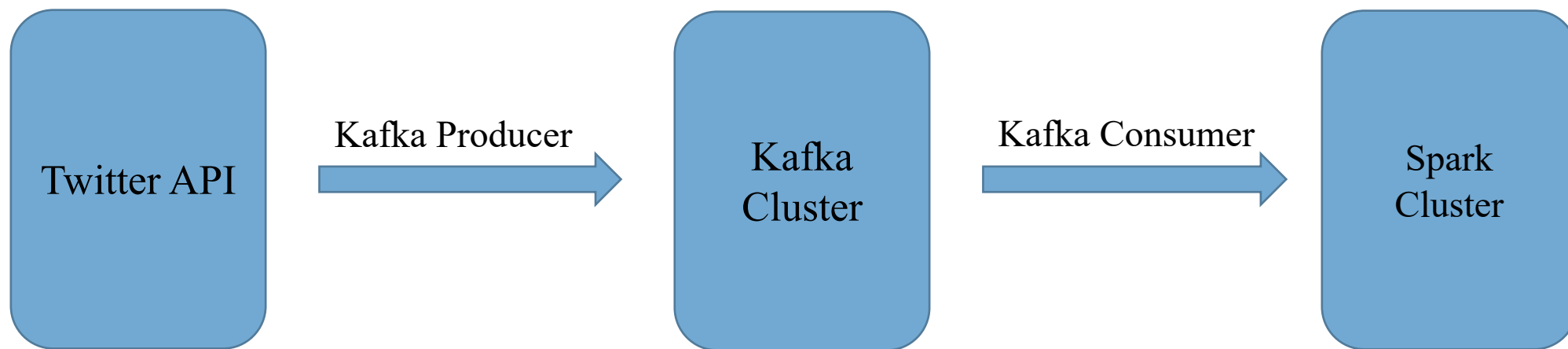
Web UI URL:

demo.csye7200.xyz



Implementation: Data Ingestion

Implementation: Data Ingestion workflow



Implementation: Data Ingestion

Kafka

```
ubuntu@ip-10-0-1-110:~/kafka_2.13-2.5.0$ bin/kafka-topics.sh --describe --zookeeper ec2-3-81-98-238.compute-1.amazonaws.com:2181 --topic twitterdata
Topic: twitterdata      PartitionCount: 3      ReplicationFactor: 3      Configs:
  Topic: twitterdata    Partition: 0          Leader: 0                Replicas: 0,1,2 Isr: 0,1,2
  Topic: twitterdata    Partition: 1          Leader: 1                Replicas: 1,2,0 Isr: 1,2,0
  Topic: twitterdata    Partition: 2          Leader: 2                Replicas: 2,0,1 Isr: 2,0,1
```

Kafka topic: twitterdata

Broker count: 3

Partition count: 3 higher parallelism, higher throughput

Replication count: 3

Implementation: Data Ingestion Kafka

Kafka consumer connects with Spark Streaming

```
// setup Kafka params
val kafkaParams = Map[String, Object](
  elems = "bootstrap.servers" -> ConnectUtils.getParams( paramName = "KAFKA_IP"),
  "key.deserializer" -> classOf[StringDeserializer],
  "value.deserializer" -> classOf[StringDeserializer],
  "group.id" -> "group_1",
  "auto.offset.reset" -> "latest",
  "enable.auto.commit" -> (true: java.lang.Boolean)
)

// setup consumer Kafka topics
val topics = Array("twitterdata")

// consume discretized streams(DStream) from Kafka
val kafkaDStream = KafkaUtils.createDirectStream[String, String](
  ssc,
  LocationStrategies.PreferConsistent,
  ConsumerStrategies.Subscribe[String, String](topics, kafkaParams)
)
```




Implementation: Data Processing

Implementation: Data Processing Spark MLlib

```
object MLibNaiveBayesPrediction {
```

```
def computeSentiment(text: String, stopWordsList: Broadcast[List[String]], model: NaiveBayesModel): Int = {  
  val tweets: Seq[String] = getClearTweetText(text, stopWordsList.value)  
  val polarity = model.predict(MLibNaiveBayesPrediction.transformFeatures(tweets))  
  normalizeSentiment(polarity)  
}
```

```
def normalizeSentiment(sentiment: Double): Int = {  
  sentiment match {  
    case x if x == 0 => -2 // very negative  
    case x if x == 1 => -1 // negative  
    case x if x == 2 => 0 // neutral  
    case x if x == 4 => 1 // positive  
    case _ => 0 // neutral  
  }  
}
```

```
def getClearTweetText(tweetText: String, stopWordsL
```

```
object MLibNaiveBayesModelCreator {
```

```
def main(args: Array[String]) {  
  val sc = createSparkContext()  
  // LogUtils.setLogLevels(sc)  
  val stopWordsList = sc.broadcast(StopWordsLoader.loadStopWords(PropertiesLoaderUtils.nltkStopWordsFileName))  
  createAndSaveModel(sc, stopWordsList)  
  computeAccuracyOfModel(sc, stopWordsList)  
}
```

```
def replaceNewLines(tweetText: String): String = {...}
```

```
def createSparkContext(): SparkContext = {...}
```

```
def createAndSaveModel(sc: SparkContext, stopWordsList: Broadcast[List[String]]): Unit = {...}
```

```
def computeAccuracyOfModel(sc: SparkContext, stopWordsList: Broadcast[List[String]]): Unit = {...}
```


Implementation: Data Processing Spark MLlib

```
21/12/07 18:15:54 INFO DAGScheduler: Job 14 is finished. Cancelling potential speculative or zombie
21/12/07 18:15:54 INFO TaskSchedulerImpl: Killing all running tasks in stage 17: Stage finished
21/12/07 18:15:54 INFO DAGScheduler: Job 14 finished: count at MLlibNaiveBayesModelCreator.scala:62,
21/12/07 18:15:54 INFO SparkContext: Invoking stop() from shutdown hook
```

```
[***** ML model prediction accuracy compared to actual: 77.91% *****]
```

```
21/12/07 18:15:54 INFO SparkUI: Stopped Spark web UI at http://luciens-mbp.fios-router.home:4040
21/12/07 18:15:54 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
21/12/07 18:15:54 INFO MemoryStore: MemoryStore cleared
21/12/07 18:15:54 INFO BlockManager: BlockManager stopped
```


Implementation: Data Processing Spark Streaming

```
kafkaDStream.foreachRDD(rdd => {  
  rdd.foreachPartition(partitionOfRecords => {  
    //both partition and record are located in local Worker  
    //get MySQL connection  
    val conn = MySQLManager.getMysqlPool.getConnection  
    if (conn == null) {  
      println("conn is null.") //// print in the executor  
    } else {  
      println("conn is not null.")  
      //create statement  
      val statement = conn.createStatement()  
      try {  
        conn.setAutoCommit(false) //commit manually  
        partitionOfRecords.foreach(record => {  
          val recordArray = record.value().split(regex = "  
          val keyword = recordArray(0)  
          val token = recordArray(1)  
          val tweet = recordArray(2)  
          val like_num = recordArray(3).toInt  
          val reply_num = recordArray(4).toInt  
          val retweet_num = recordArray(4).toInt
```

```
          val sentiment_res = MLlibNaiveBayesPrediction.computeSentiment(record.value(), stopWordslist, naiveBayesModel)  
          println(sentiment_res)  
          println("-----")  
          // create SQL query and add it into batch  
          val sql = "INSERT INTO tweetsInfo(keyword,token,sentiment_res,like_num,retweet_num) " +  
            "values ('"+keyword+"','"+token+"','"+sentiment_res+"','"+like_num+"','"+retweet_num+"');"  
          statement.addBatch(sql) // add into batch  
        })  
        statement.executeBatch() //execute query in batch  
        conn.commit() //transaction commit  
      } catch {  
        case e: Exception => e.printStackTrace()  
      } finally {  
        statement.close() //close statement  
        conn.close() //close connection  
      }  
    }  
  })  
})  
  
// start the computation  
ssc.start()  
// wait for the computation to terminate  
ssc.awaitTermination()  
}
```


Implementation: Data Processing Optimization

Kryo Serializer instead of default Java Serializer

```
// create StreamingContext
// val conf = new SparkConf().setMaster("local[2]").setAppName("twitter_stream_processing") // submit job in IDE
val conf = new SparkConf().setAppName("twitter_stream_processing").
set("spark.serializer", classOf[KryoSerializer].getCanonicalName)// submit job in Spark Standalone
val ssc = new StreamingContext(conf, Seconds(5))
```

DStream Cache and serialization

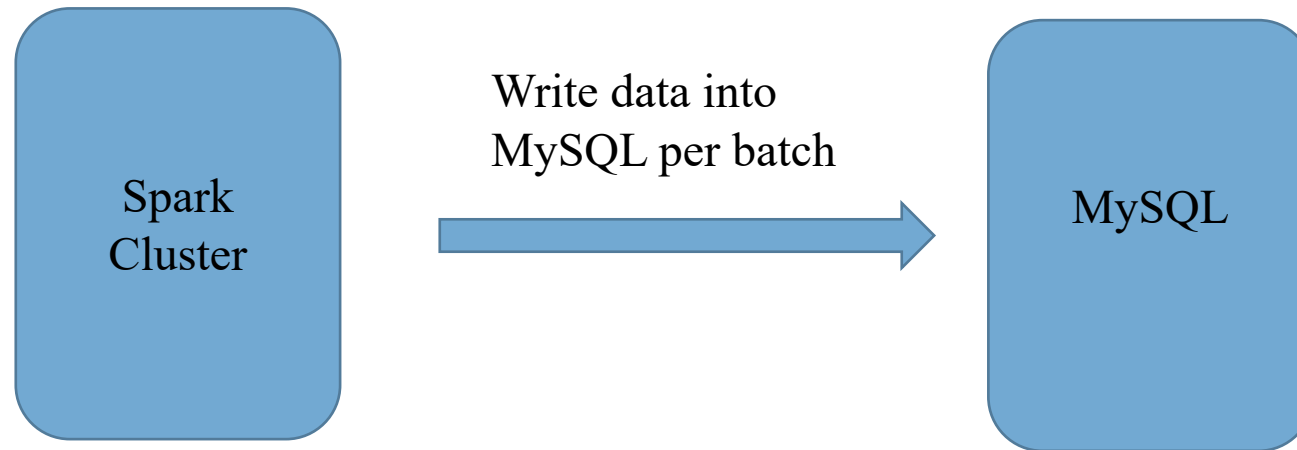
```
kafkaDStream.persist(StorageLevel.MEMORY_ONLY_SER)
// compute each RDD in discretized streams(DStream)
kafkaDStream.foreachRDD(rdd => {
  rdd.foreachPartition(partitionOfRecords => {
    //both partition and record are located in local Worker
```




Implementation: Data Storage

Implementation: Data Storage

MySQL



Implementation: Data Storage MySQL

```
object DBTableCreator {
  def main(args: Array[String]): Unit = {
    // get MySQL connection
    val conn = MySQLManager.getMysqlPool.getConnection
    if (conn == null) {
      println("conn is null.") // print in the executor of worker
    } else {
      println("conn is not null.")
      // create statement
      val statement = conn.createStatement()
      try {
        conn.setAutoCommit(false) //do not auto commit

        //create SQL query
        val sql = "CREATE TABLE tweetsInfo (" +
          "id INT PRIMARY KEY AUTO_INCREMENT, " +
          "keyword VARCHAR(25), " +
          "token VARCHAR(25), " +
          "sentiment_res INT, " +
          "like_num INT, " +
          "reply_num INT, " +
          "retweet_num INT);"
        statement.execute(sql)
        conn.commit() // transaction commit
      } catch {
        case e: Exception => e.printStackTrace()
      } finally {
        statement.close() // close statement
        conn.close() //connection close
      }
    }
  }
}
```

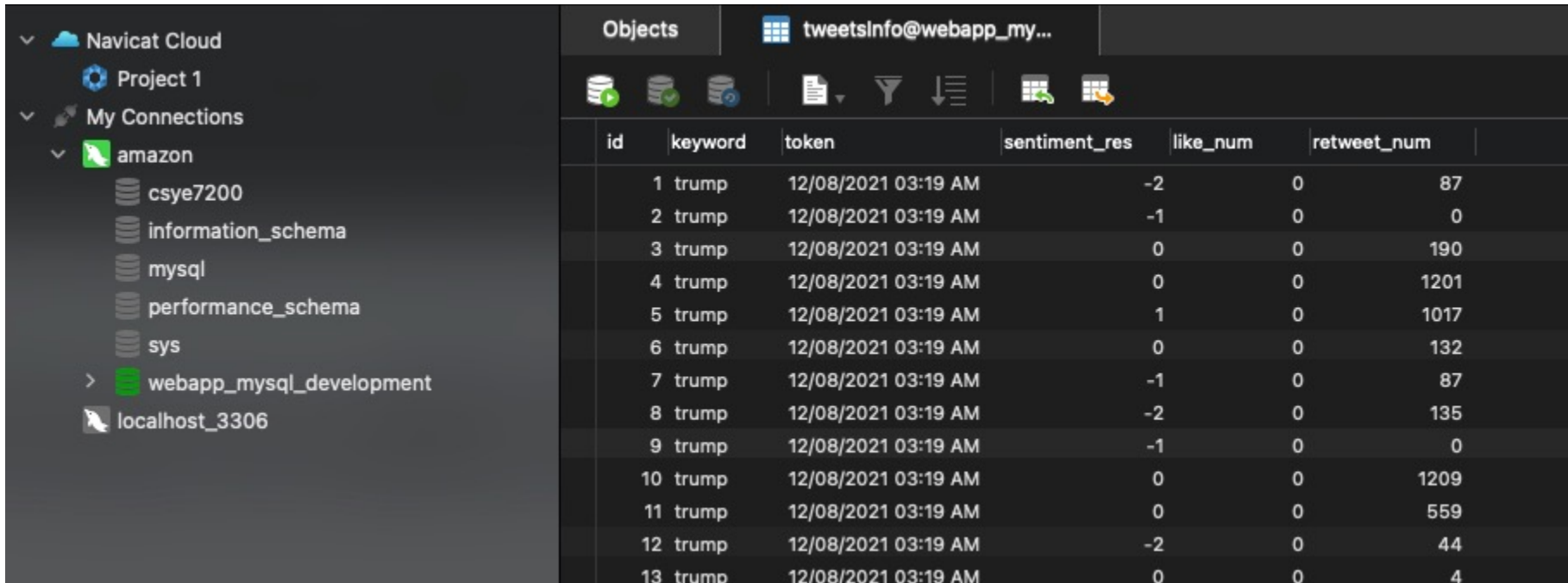
```
class MySQLConnectPool extends Serializable {
  private val cpds: ComboPooledDataSource = new ComboPooledDataSource(autoRegister = true) // auto registration
  try {
    //.../
    cpds.setJdbcUrl("jdbc:mysql://" + ConnectUtils.getParams(paramName = "MYSQL_URL") + ":3306/webapp_mysql_development")
    cpds.setDriverClass("com.mysql.cj.jdbc.Driver") //mysql-connector-java-8.0.16 driver
    cpds.setUser(ConnectUtils.getParams(paramName = "MYSQL_USER"))
    cpds.setPassword(ConnectUtils.getParams(paramName = "MYSQL_PW"))
    cpds.setMaxPoolSize(10)
    cpds.setMinPoolSize(2)
    cpds.setAcquireIncrement(2)
    cpds.setMaxStatements(180)
  } catch {
    case e: Exception => e.printStackTrace()
  }

  // get connection
  def getConnection: Connection = {...}
}

// lazy singleton, initialize as needed
object MySQLManager {
  @volatile private var mysqlPool: MySQLConnectPool = _
  def getMysqlPool: MySQLConnectPool = {
    if (mysqlPool == null) {
      synchronized {
        if (mysqlPool == null) {
          mysqlPool = new MySQLConnectPool
        }
      }
    }
    mysqlPool
  }
}
```


Implementation: Data Storage

MySQL



The screenshot shows the Navicat Cloud interface. On the left, the 'My Connections' tree is expanded, showing a connection to 'amazon' with a database named 'webapp_mysql_development' on 'localhost_3306'. The main pane displays a table with the following columns: id, keyword, token, sentiment_res, like_num, and retweet_num. The table contains 13 rows of data, all with the keyword 'trump' and a timestamp of '12/08/2021 03:19 AM'.

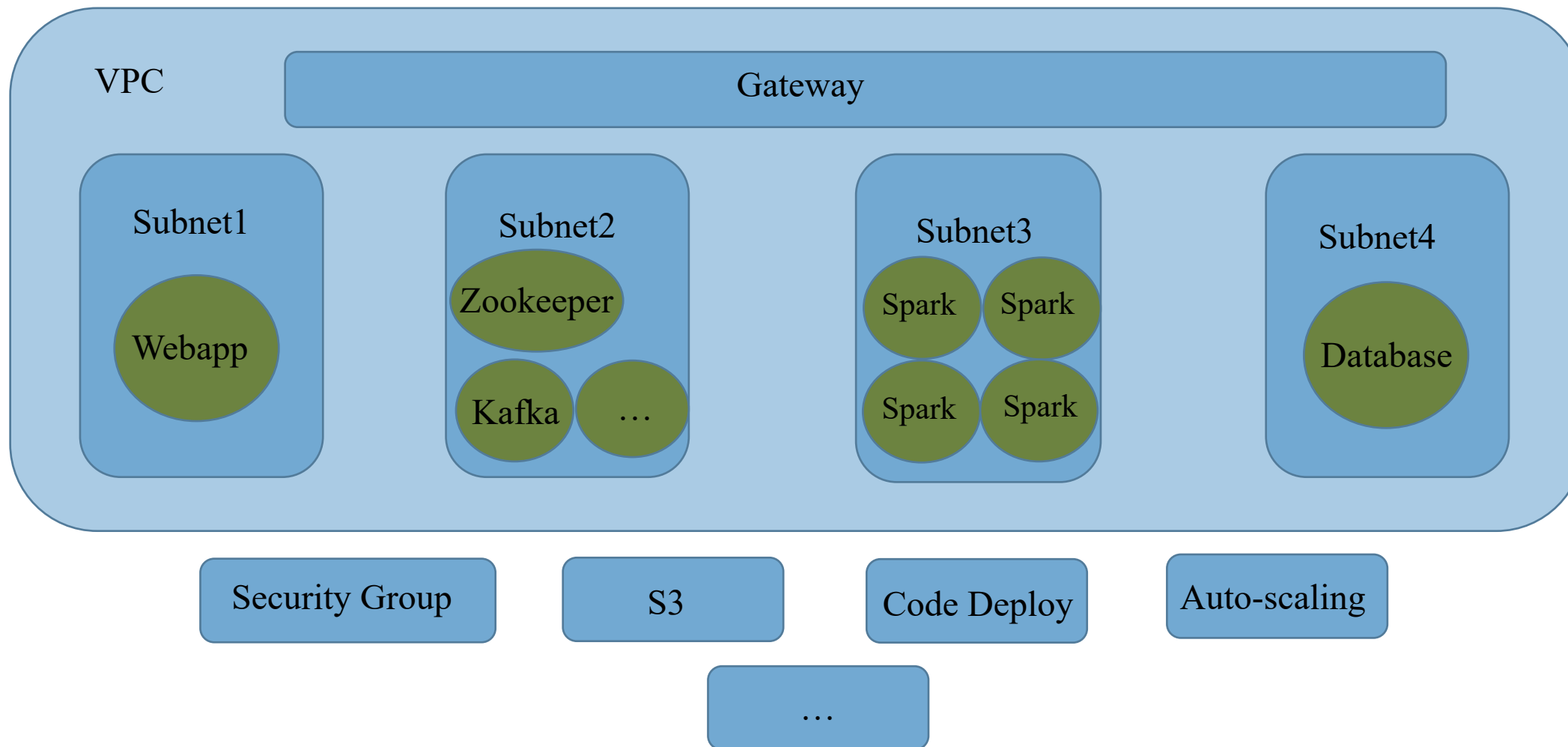
id	keyword	token	sentiment_res	like_num	retweet_num
1	trump	12/08/2021 03:19 AM	-2	0	87
2	trump	12/08/2021 03:19 AM	-1	0	0
3	trump	12/08/2021 03:19 AM	0	0	190
4	trump	12/08/2021 03:19 AM	0	0	1201
5	trump	12/08/2021 03:19 AM	1	0	1017
6	trump	12/08/2021 03:19 AM	0	0	132
7	trump	12/08/2021 03:19 AM	-1	0	87
8	trump	12/08/2021 03:19 AM	-2	0	135
9	trump	12/08/2021 03:19 AM	-1	0	0
10	trump	12/08/2021 03:19 AM	0	0	1209
11	trump	12/08/2021 03:19 AM	0	0	559
12	trump	12/08/2021 03:19 AM	-2	0	44
13	trump	12/08/2021 03:19 AM	0	0	4

995	Biden	12/09/2021 07:19 AM	0	0	7661
996	Biden	12/09/2021 07:19 AM	0	0	320
997	Biden	12/09/2021 07:19 AM	0	0	21
998	Biden	12/09/2021 07:19 AM	0	0	649
999	Biden	12/09/2021 07:19 AM	-1	0	5607
1000	Biden	12/09/2021 07:19 AM	0	0	14
1001	Biden	12/09/2021 07:19 AM	-2	0	217

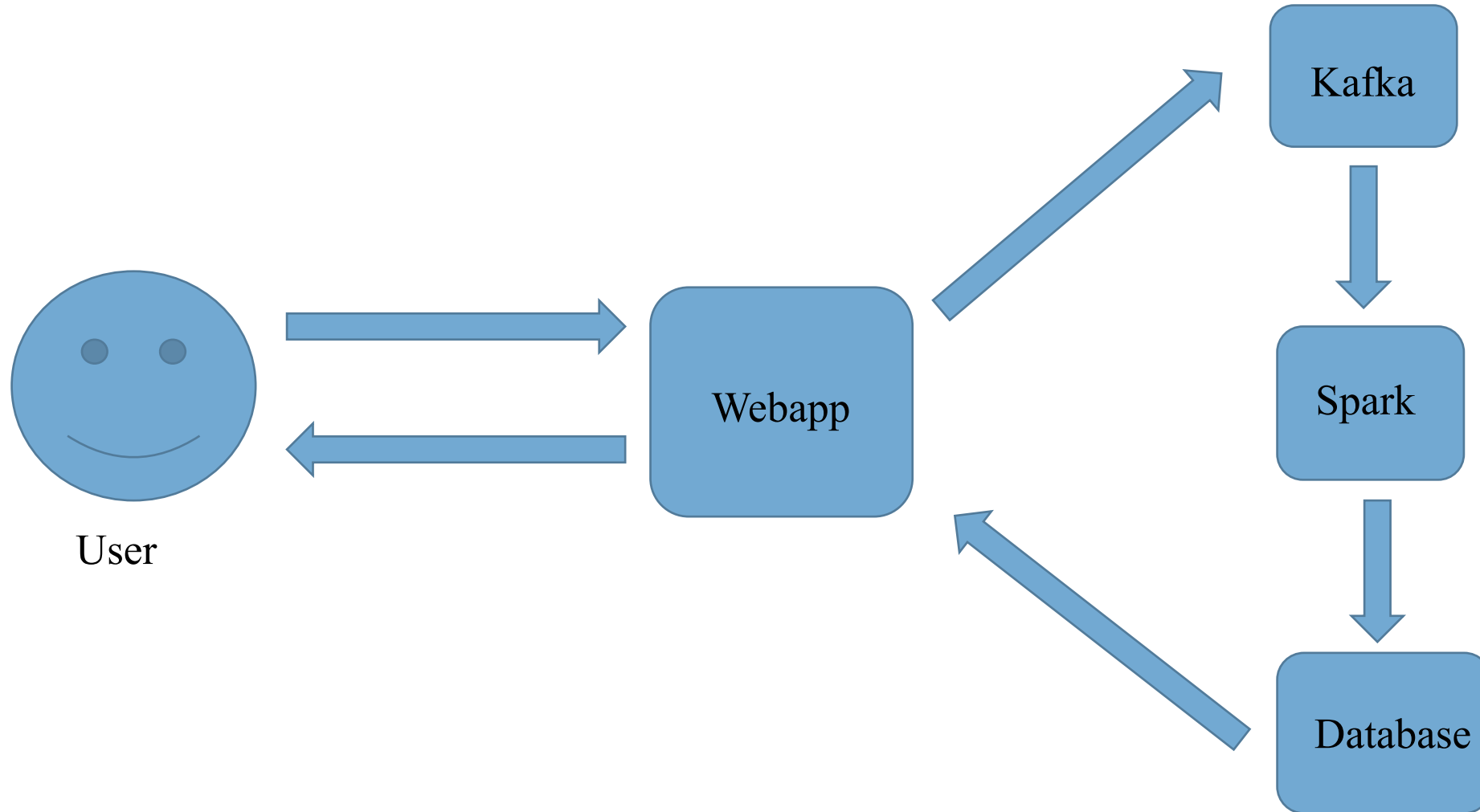
The background of the slide is a complex network of thin, light blue lines connecting numerous small, semi-transparent blue and black dots, creating a web-like or molecular structure.

Implementation: AWS Deploy

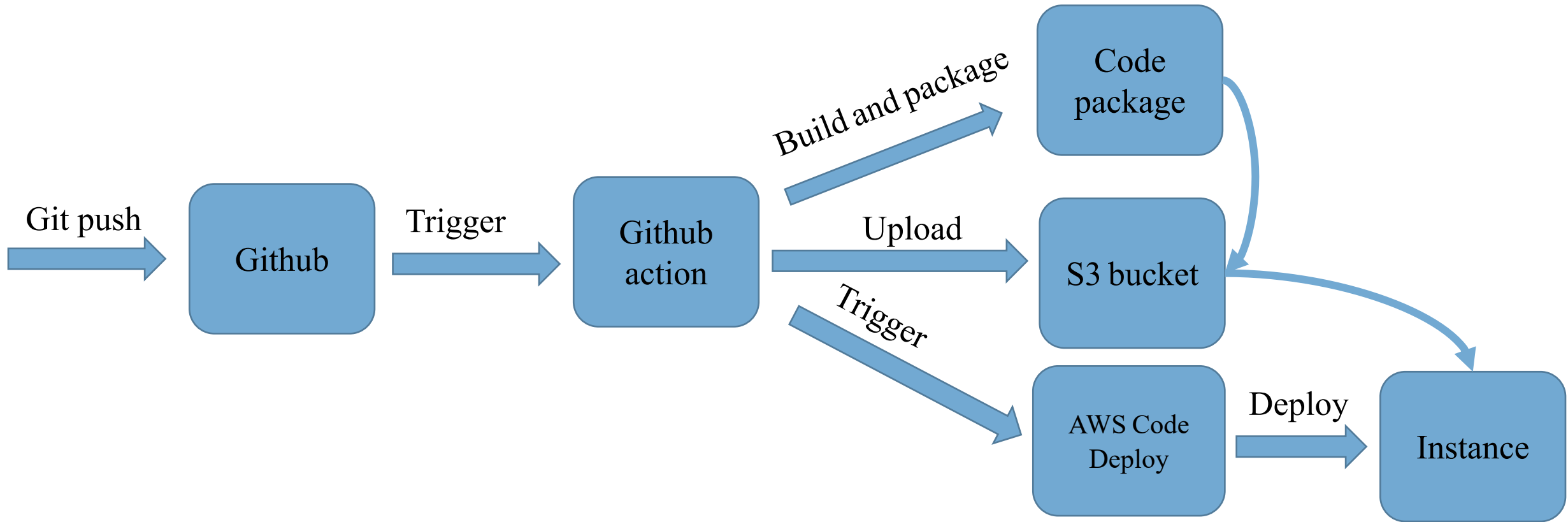
Implementation: AWS Deploy Cloud



Implementation: AWS Deploy Workflow



Implementation: AWS Deploy Continuous Integration and Code Deployment / CI/CD



Implementation: AWS Deploy Terraform

```
resource "aws_security_group" "terraform" {
  name = "terraform"
  description = "Terraform security group"
  vpc_id = aws_vpc.main.id

  ingress = [
    {
      description = "SSH"
      from_port = 22
      to_port = 22
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    },
    {
      description = "HTTP"
      from_port = 80
      to_port = 80
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    },
    {
      description = "HTTPS"
      from_port = 443
      to_port = 443
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  ]

  egress = [
    {
      description = "All traffic"
      from_port = 0
      to_port = 0
      protocol = "all"
      cidr_blocks = ["0.0.0.0/0"]
    }
  ]
}

resource "aws_instance" "terraform" {
  ami = "ami-0c55b1e9"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.main.id
  vpc_id = aws_vpc.main.id
  security_group_ids = [aws_security_group.terraform.id]
  key_name = "terraform"
  user_data = <<-EOF
    #!/bin/bash
    echo "Terraform deployment script"
    sudo apt-get update
    sudo apt-get install -y terraform
    EOF
  tags = {
    Name = "terraform"
  }
}
```

```
resource "aws_security_group" "terraform" {
  name = "terraform"
  description = "Terraform security group"
  vpc_id = aws_vpc.main.id

  ingress = [
    {
      description = "SSH"
      from_port = 22
      to_port = 22
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    },
    {
      description = "HTTP"
      from_port = 80
      to_port = 80
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    },
    {
      description = "HTTPS"
      from_port = 443
      to_port = 443
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  ]

  egress = [
    {
      description = "All traffic"
      from_port = 0
      to_port = 0
      protocol = "all"
      cidr_blocks = ["0.0.0.0/0"]
    }
  ]
}

resource "aws_instance" "terraform" {
  ami = "ami-0c55b1e9"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.main.id
  vpc_id = aws_vpc.main.id
  security_group_ids = [aws_security_group.terraform.id]
  key_name = "terraform"
  user_data = <<-EOF
    #!/bin/bash
    echo "Terraform deployment script"
    sudo apt-get update
    sudo apt-get install -y terraform
    EOF
  tags = {
    Name = "terraform"
  }
}
```

```
resource "aws_security_group" "database" {
  name = "database"
  description = "Database security group"
  vpc_id = aws_vpc.main.id

  ingress = [
    {
      description = "MySQL"
      from_port = 3306
      to_port = 3306
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    },
    {
      description = "MySQL"
      from_port = 3306
      to_port = 3306
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  ]

  egress = [
    {
      description = "MySQL"
      from_port = 3306
      to_port = 3306
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    },
    {
      description = "MySQL"
      from_port = 3306
      to_port = 3306
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  ]
}

resource "aws_instance" "database" {
  ami = "ami-0c55b1e9"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.main.id
  vpc_id = aws_vpc.main.id
  security_group_ids = [aws_security_group.database.id]
  key_name = "database"
  user_data = <<-EOF
    #!/bin/bash
    echo "Database deployment script"
    sudo apt-get update
    sudo apt-get install -y mysql
    EOF
  tags = {
    Name = "database"
  }
}
```


Implementation: AWS Deploy Packer

Why Packer?



Rapid Infrastructure Deployment

Use Terraform to launch completely provisioned and configured machine instances with Packer images in seconds.



Multi-provider Portability

Identical images allow you to run dev, staging, and production environments across platforms.



Improved Stability

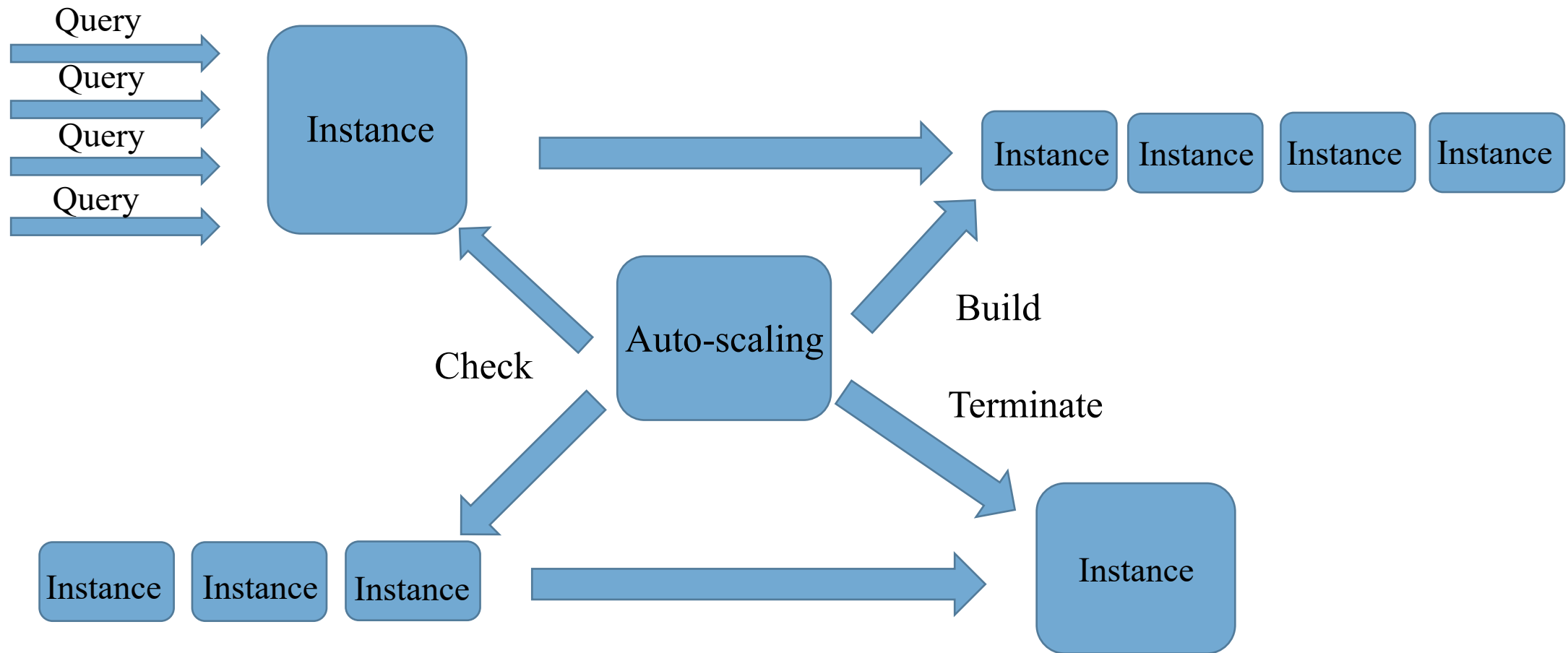
By provisioning instances from stable images installed and configured by Packer, you can ensure buggy software does not get deployed.



Increased Dev / Production Parity

Keep dev, staging, and production environments as similar as possible by generating images for multiple platforms at the same time.

Implementation: AWS Deploy Auto-Scaling



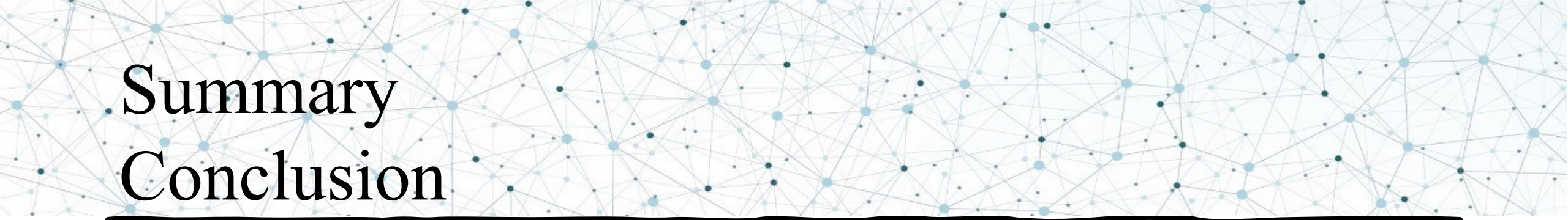


Summary

Summary

Acceptance Criteria

1. This system can ingest the relative streaming tweets from Twitter according to the input keywords by user. [Shown in Web UI and Data ingestion part](#)
2. This system can process about 150 tweets for every query and perform the sentiment analysis for these real-time tweets in the seconds. [Shown in Spark Streaming and MySQL part](#)
3. This system will be expected to achieve 70% sentiment analysis accuracy. [Shown in Spark MLlib part](#)
4. This system will show the sentiment analysis results in the visualization chart as the feedback to the user. [Shown in Web UI part](#)
5. This system will be deployed on AWS. [Shown in AWS deploy part](#)

A decorative background pattern at the top of the slide, consisting of a dense network of light blue and grey dots connected by thin lines, resembling a data network or a molecular structure.

Summary Conclusion

By completing the project, we know how to design and build the big data system. We have a deeper understanding of big data frameworks, tools and cloud computing. Most importantly, we will achieve good, practical, working knowledge of Scala.



Thank You!