Kauan Amarante

Arquiteturas para processamento de dados: Comparativo entre arquiteturas *Lambda* e arquiteturas *Kappa*

Araquari – SC

2022

Kauan Amarante

Arquiteturas para processamento de dados: Comparativo entre arquiteturas *Lambda* e arquiteturas *Kappa*

Trabalho de conclusão de curso apresentado como requisito parcial para a obtenção do grau de bacharel em Sistemas de Informação do Instituto Federal Catarinense.

Instituto Federal Catarinense – IFC

Campus Araquari

Bacharelado em Sistemas de Informação

Orientador: Prof. Dr. Eduardo da Silva

Araquari – SC

2022

Kauan Amarante
Arquiteturas para processamento de dados: Comparativo entre arquiteturas *Lambda* e arquiteturas *Kappa*/ Kauan Amarante. – Araquari – SC, 2022-
28 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Eduardo da Silva

Monografia (Graduação) – Instituto Federal Catarinense – IFC
Campus Araquari
Bacharelado em Sistemas de Informação, 2022.

1. Arquitetura. 2. Processamento de dados. 3. *Lambda*. 4. *Kappa*. I. Eduardo da Silva. II. Instituto Federal Catarinense. III. Câmpus Araquari. IV. Arquiteturas para processamento de dados: Comparativo entre arquiteturas *Lambda* e arquiteturas *Kappa*

Kauan Amarante

Arquiteturas para processamento de dados: Comparativo entre arquiteturas *Lambda* e arquiteturas *Kappa*

Trabalho de conclusão de curso apresentado como requisito parcial para a obtenção do grau de bacharel em Sistemas de Informação do Instituto Federal Catarinense.

Trabalho aprovado. Araquari – SC, 24 de novembro de 2016:

Prof. Dr. Eduardo da Silva
Orientador

Professor
Convidado 1

Professor
Convidado 2

Araquari – SC
2022

A todos aqueles que de alguma forma estiveram e estão próximos de mim, fazendo esta vida valer cada vez mais a pena.

Agradecimentos

"Entre o nascer e o morrer existe algo,
um momento efêmero, único, de uma
fascínio estúpido, que se chama vida"
(Lourenço Mutarelli)

Resumo

Segundo a **??**, 3.1-3.2), o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (. . .) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto.

Palavras-chave: latex. abntex. editoração de texto.

Abstract

This is the english abstract.

Keywords: latex. abntex. text editoration.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
abnTeX	ABsurdas Normas para TeX

Lista de símbolos

Γ	Letra grega Gama
Λ	Lambda
ζ	Letra grega minúscula zeta
$\in$	Pertence

Sumário

1 Introdução

2 Referencial Teórico

Para o desenvolvimento deste projeto, fez-se necessária a realização de uma revisão de literatura, de modo a obter uma base sólida de conhecimento para a condução das atividades pertinentes a este trabalho, e seguem descritas na sequência.

2.1 *Big Data*

(SAGIROGLU; SINANC, 2013)

2.2 Arquiteturas para processamento de dados

2.2.1 Arquitetura *Lambda*

2.2.2 Arquitetura *Kappa*

2.3 Apache Kafka

2.4 *Data Lake*

3 Metodologia

Com o objetivo de garantir uma infraestrutura robusta e escalável para a aplicação escolhida, ficou decidido pela utilização de um *cluster* Kubernetes, visto sua capacidade de resiliência e alta disponibilidade.

3.1 Área de Estudo

3.2 Método

Alterar inicio Com o intuito de realizar uma prova de conceito de baixo custo, optou-se pela utilização de uma distribuição Kubernetes local, chamada de Minikube,

4 Resultados

5

6 Considerações Finais

A PoC teve grande valia, uma vez que mostrou a ineficácia do Hudi ao tratar dados particionados na origem, além de somente funcionar no master node do EMR. Talvez seja por ser algo que ainda esteja em incubação pela Apache Software Foundation e em um futuro possam resolver estes problemas, por este motivo foi decidido dar a PoC como encerrada.

Referências

SAGIROGLU, S.; SINANC, D. Big data: A review. In: IEEE. *2013 international conference on collaboration technologies and systems (CTS)*. [S.l.], 2013. p. 42–47.

Apêndices

APÊNDICE A – Código para realizar a inserção de dados

```
package com.contaazul.jarvis.hudi.insert

import org.apache.hudi.DataSourceWriteOptions
import org.apache.hudi.config.HoodieWriteConfig
import org.apache.hudi.hive.MultiPartKeysValueExtractor
import org.apache.spark.sql.functions.{concat, lit}
import org.apache.spark.sql.types.DateType
import org.apache.spark.sql.{SaveMode, SparkSession}
import org.apache.spark.{SparkConf, SparkContext}

object HudiInsert {

  def processInsert(params: Map[String, String]) {

    val spark = SparkSession.builder().appName("insert " + params("table")).getOrCreate()

    import spark.implicits._

    // Read data from S3 and create a DataFrame with Partition and Record Key
    var insertDF = spark.read.json(
      "s3://cdc-tests/cdc/" + params("upsertOption") + "/" +
        params("instance") + "/" + params("database") + "/" +
        params("schema") + "/" + params("table") + "/" +
        params("year") + "/" + params("month") + "/" +
        params("day") + "/" + params("hour"))

    insertDF = insertDF.withColumn("committed_at", insertDF("committed_at").cast(DateType))

    val hudiTablePartitionKey = "partition_key"
    insertDF = insertDF.withColumn(hudiTablePartitionKey, concat(lit("year="), $"year", lit("/month="), $"month", lit("/day="), $"day", lit("/hour="), $"hour"))

    //Specify common DataSourceWriteOptions in the single hudiOptions variable
    val hudiOptions = Map[String, String](
      HoodieWriteConfig.TABLE_NAME -> params("table"),
```

```
35      DataSourceWriteOptions.STORAGE_TYPE_OPT_KEY -> "COPY_ON_WRITE",
36      DataSourceWriteOptions.RECORDKEY_FIELD_OPT_KEY -> "id",
37      DataSourceWriteOptions.PARTITIONPATH_FIELD_OPT_KEY ->
    hudiTablePartitionKey,
38      DataSourceWriteOptions.PRECOMBINE_FIELD_OPT_KEY -> "committed_at",
39      DataSourceWriteOptions.HIVE_SYNC_ENABLED_OPT_KEY -> "true",
40      DataSourceWriteOptions.HIVE_DATABASE_OPT_KEY -> params("database")
    ,
41      DataSourceWriteOptions.HIVE_TABLE_OPT_KEY -> params("table"),
42      DataSourceWriteOptions.HIVE_PARTITION_FIELDS_OPT_KEY -> "year,
    month,day,hour",
43      DataSourceWriteOptions.HIVE_PARTITION_EXTRACTOR_CLASS_OPT_KEY ->
    classOf[MultiPartKeysValueExtractor].getName
44     )
45
46     insertDF.write.format("org.apache.hudi")
47       .option(DataSourceWriteOptions.OPERATION_OPT_KEY,
    DataSourceWriteOptions.INSERT_OPERATION_OPT_VAL)
48       .options(hudiOptions)
49       .mode(SaveMode.Overwrite)
50       .save("s3://cdc-tests/hudi/" + params("table"))
51   }
52
53   def main(args: Array[String]) {
54
55     val conf = new SparkConf().setAppName("Update Users")
56     conf.set("spark.serializer", "org.apache.spark.serializer.
    KryoSerializer")
57     conf.set("spark.sql.hive.convertMetastoreParquet", "false")
58     conf.set("spark.executor.memory", "7G")
59     conf.set("spark.dynamicAllocation.executorIdleTimeout", "3600")
60     conf.set("spark.executor.cores", "1")
61     conf.set("spark.dynamicAllocation.initialExecutors", "16")
62     conf.set("spark.sql.parquet.outputTimestampType", "TIMESTAMP_MILLIS
    ")
63
64     val sc = new SparkContext(conf)
65
66     val instance = args(0)
67     val database = args(1)
68     val schema = args(2)
69     val table = args(3)
70     val year = args(4)
71     val month = args(5)
72     val day = args(6)
73     val hour = args(7)
74
```

```
75      val params = Map[String, String](
76        "instance" -> instance,
77        "database" -> database,
78        "schema" -> schema,
79        "table" -> table,
80        "year" -> year,
81        "month" -> month,
82        "day" -> day,
83        "hour" -> hour
84      )
85
86      processInsert(params)
87
88      sc.stop()
89    }
90  }
```

APÊNDICE B – Código para realizar a atualização e exclusão de dados

```
package com.contaazul.jarvis.hudi.insert

import org.apache.hudi.DataSourceWriteOptions
import org.apache.hudi.config.HoodieWriteConfig
import org.apache.hudi.hive.MultiPartKeysValueExtractor
import org.apache.spark.sql.functions.{concat, lit}
import org.apache.spark.sql.types.DateType
import org.apache.spark.sql.{SaveMode, SparkSession}
import org.apache.spark.{SparkConf, SparkContext}

object HudiInsert {

  def processInsert(params: Map[String, String]) {

    val spark = SparkSession.builder().appName("insert " + params("table
    ")).getOrCreate()

    import spark.implicits._

    // Read data from S3 and create a DataFrame with Partition and
    Record Key
    var insertDF = spark.read.json(
      "s3://cdc-tests/cdc/" + params("upsertOption") + "/" +
        params("instance") + "/" + params("database") + "/" +
        params("schema") + "/" + params("table") + "/" +
        params("year") + "/" + params("month") + "/" +
        params("day") + "/" + params("hour"))

    insertDF = insertDF.withColumn("committed_at", insertDF("
    committed_at").cast(DateType))

    val hudiTablePartitionKey = "partition_key"
    insertDF = insertDF.withColumn(hudiTablePartitionKey, concat(lit("
    year="), $"year", lit("/month="), $"month", lit("/day="), $"day", lit
    ("/hour="), $"hour"))

    //Specify common DataSourceWriteOptions in the single hudiOptions
    variable
    val hudiOptions = Map[String, String](
      HoodieWriteConfig.TABLE_NAME -> params("table"),
```

```
35      DataSourceWriteOptions.STORAGE_TYPE_OPT_KEY -> "COPY_ON_WRITE",
36      DataSourceWriteOptions.RECORDKEY_FIELD_OPT_KEY -> "id",
37      DataSourceWriteOptions.PARTITIONPATH_FIELD_OPT_KEY ->
    hudiTablePartitionKey,
38      DataSourceWriteOptions.PRECOMBINE_FIELD_OPT_KEY -> "committed_at",
39      DataSourceWriteOptions.HIVE_SYNC_ENABLED_OPT_KEY -> "true",
40      DataSourceWriteOptions.HIVE_DATABASE_OPT_KEY -> params("database")
    ,
41      DataSourceWriteOptions.HIVE_TABLE_OPT_KEY -> params("table"),
42      DataSourceWriteOptions.HIVE_PARTITION_FIELDS_OPT_KEY -> "year,
    month,day,hour",
43      DataSourceWriteOptions.HIVE_PARTITION_EXTRACTOR_CLASS_OPT_KEY ->
    classOf[MultiPartKeysValueExtractor].getName
44     )
45
46     insertDF.write.format("org.apache.hudi")
47       .option(DataSourceWriteOptions.OPERATION_OPT_KEY,
    DataSourceWriteOptions.INSERT_OPERATION_OPT_VAL)
48       .options(hudiOptions)
49       .mode(SaveMode.Overwrite)
50       .save("s3://cdc-tests/hudi/" + params("table"))
51   }
52
53   def main(args: Array[String]) {
54
55     val conf = new SparkConf().setAppName("Update Users")
56     conf.set("spark.serializer", "org.apache.spark.serializer.
    KryoSerializer")
57     conf.set("spark.sql.hive.convertMetastoreParquet", "false")
58     conf.set("spark.executor.memory", "7G")
59     conf.set("spark.dynamicAllocation.executorIdleTimeout", "3600")
60     conf.set("spark.executor.cores", "1")
61     conf.set("spark.dynamicAllocation.initialExecutors", "16")
62     conf.set("spark.sql.parquet.outputTimestampType", "TIMESTAMP_MILLIS
    ")
63
64     val sc = new SparkContext(conf)
65
66     val instance = args(0)
67     val database = args(1)
68     val schema = args(2)
69     val table = args(3)
70     val year = args(4)
71     val month = args(5)
72     val day = args(6)
73     val hour = args(7)
74
```

```
75      val params = Map[String, String](
76        "instance" -> instance,
77        "database" -> database,
78        "schema" -> schema,
79        "table" -> table,
80        "year" -> year,
81        "month" -> month,
82        "day" -> day,
83        "hour" -> hour
84      )
85
86      processInsert(params)
87
88      sc.stop()
89    }
90  }
```