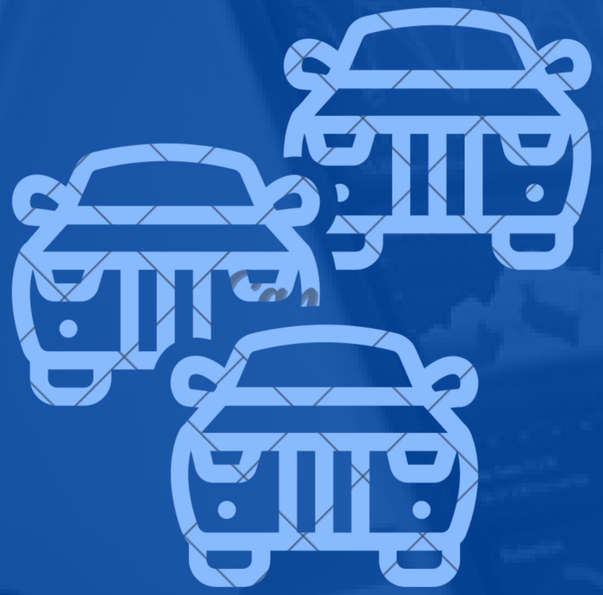


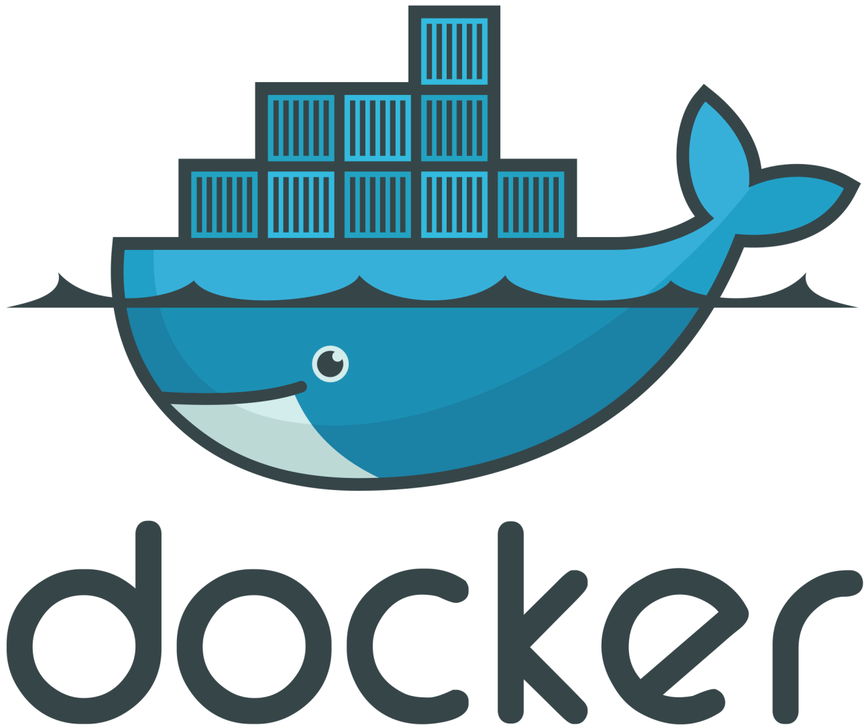
Трећи пројекат
Denver Mobility ML

Електронски факултет у Нишу
Вештачка интелигенција и машинско учење

Професор: **Др. Драган Х. Стојановић**
Студент: **Петковић Петар**

Садржај

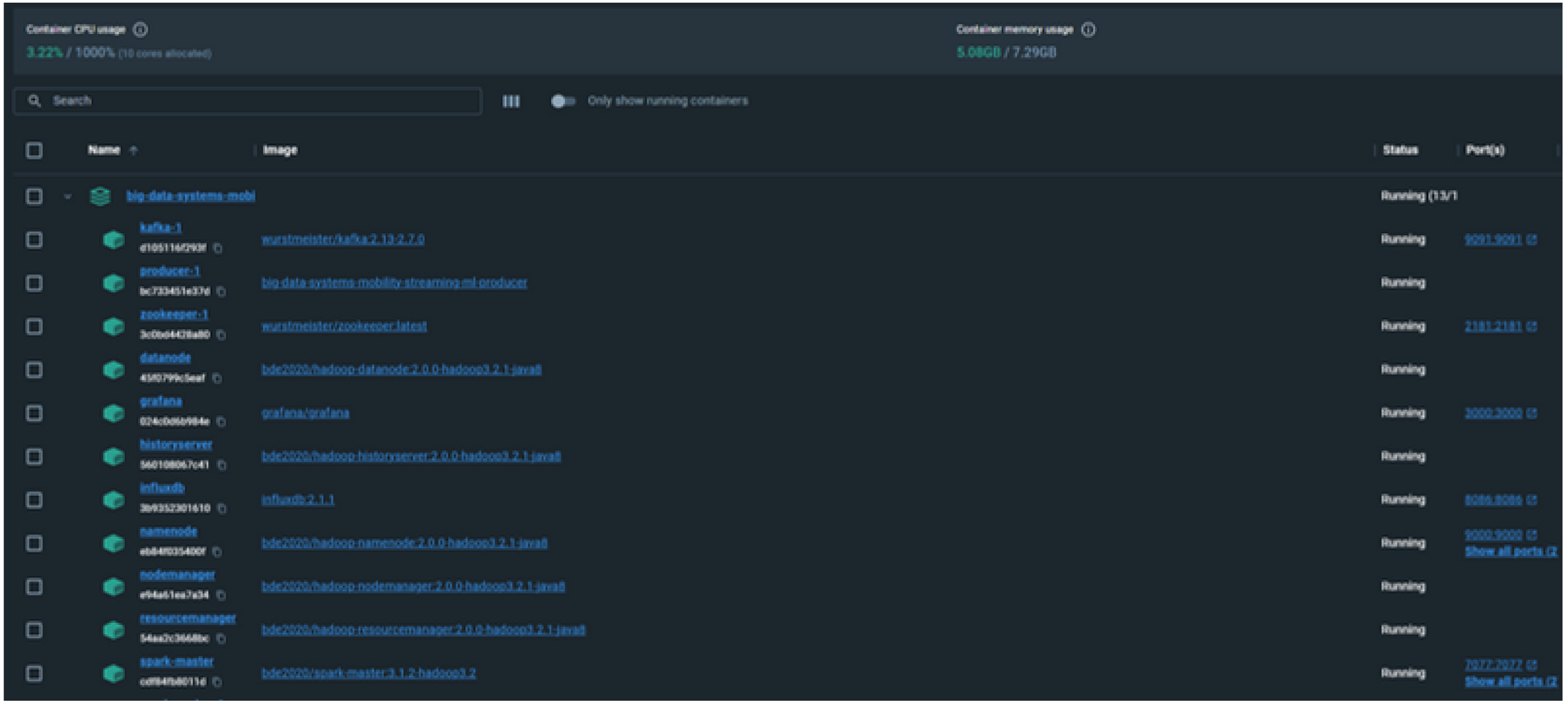
Подешавање окружења



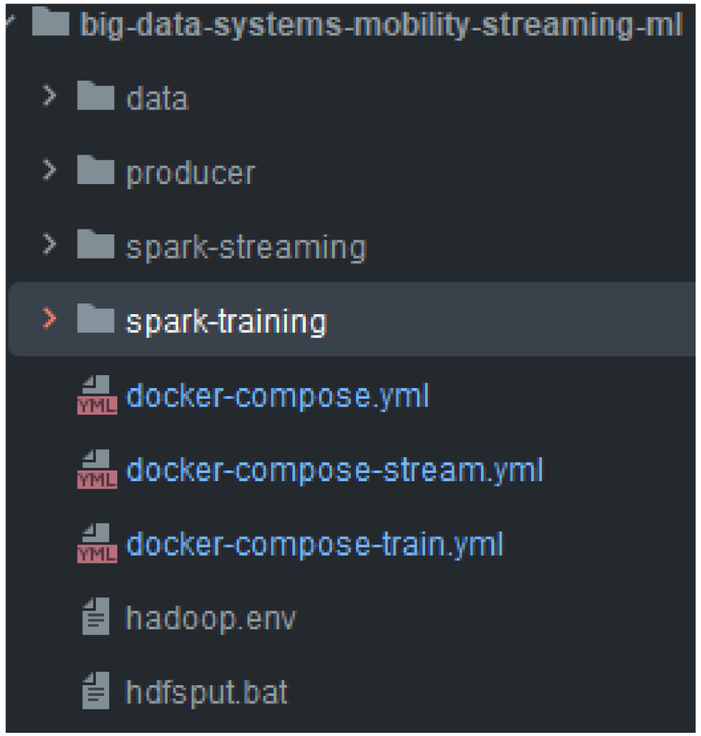
Кластер контејнера - Docker-Compose

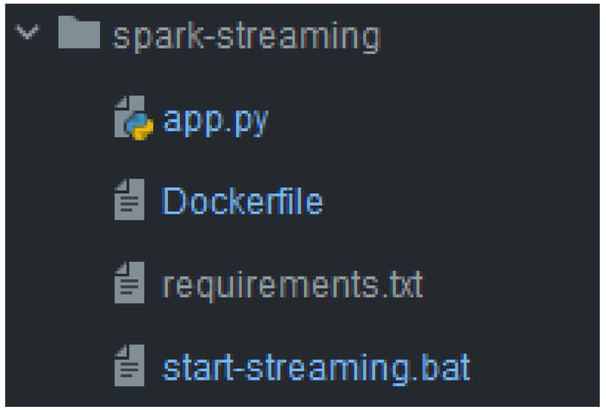
Кластер контејнера за Streaming

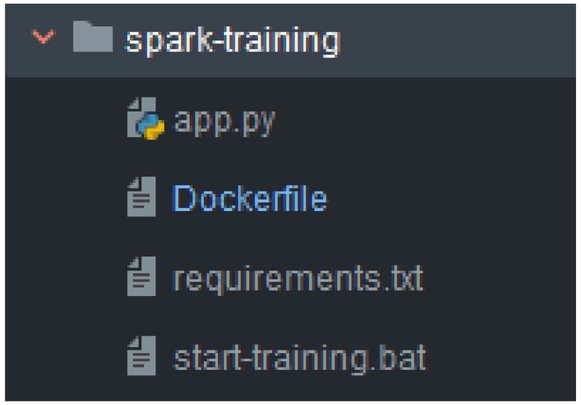
Кластер контејнера



Структура проjекта







Тренинг

```
1 usage   Petar
def process_data(dataframe):
    df = dataframe.withColumn("timestamp", to_timestamp("timestamp", "yyyy-MM-dd'T'HH:mm:ss.SSSXXX"))
    label = when((df["speed_kmh"] < 15), 1).otherwise(0)
    df = df.withColumn("zastoj", label)
    df = df.withColumn("timestamp", date_format("timestamp", "MM-dd-yyyy HH:mm:ss"))
    df = df.withColumn("latitude", df["latitude"].cast(DoubleType()))
    df = df.withColumn("longitude", df["longitude"].cast(DoubleType()))
    df = df.withColumn("speed_kmh", df["speed_kmh"].cast(DoubleType()))
    return df
```

Тренинг

```
def Regression(train, test):
    lr = (LinearRegression(featuresCol='scaled_features', labelCol="speed_kmh", predictionCol='prediction',
                           maxIter=10, regParam=0.3, elasticNetParam=0.8, standardization=False))
    linearModel = lr.fit(train)
    predictions = linearModel.transform(test)
    predictions.select('speed_kmh', 'prediction').show()
    linearModel.write().overwrite().save(MODEL_LOCATION)
    print("RMSE: {0}".format(linearModel.summary.rootMeanSquaredError))
    print("MAE: {0}".format(linearModel.summary.meanAbsoluteError))
    print("R2: {0}".format(linearModel.summary.r2))
```

Тренинг

```
if __name__ == '__main__':
    df = Inicijalizacija()
    df.show(n=5)
    df = df.dropna()

    indexer_model = StringIndexer(inputCols=['id', 'timestamp', 'type'],
                                  outputCols=['id_num', 'timestamp_num', 'type_num']).fit(df)
    indexer_model.write().overwrite().save(INDEXER_LOCATION)

    df_transformed = indexer_model.transform(df)
    df_transformed = df_transformed.drop('id')
    df_transformed = df_transformed.drop('timestamp')
    df_transformed = df_transformed.drop('type')

    df_transformed.show(n=5)

    columns = ["latitude", "longitude", "timestamp_num", "id_num", "type_num"]
    va = VectorAssembler().setInputCols(columns).setOutputCol('features').setHandleInvalid("skip").transform(
        df_transformed)
    va.describe().show()

    split = va.randomSplit([0.8, 0.2], 42)
    scaler = StandardScaler(inputCol="features", outputCol="scaled_features")
    scaler_model = scaler.fit(split[0])
    scaler_model.write().overwrite().save(SCALER_LOCATION)

    train = scaler_model.transform(split[0])
    test = scaler_model.transform(split[1])

    Regression(train, test)
```

Тренинг

```
+---------+------------------+
|speed_kmh|        prediction|
+---------+------------------+
|    48.82| 43.95797590658367|
|    45.65|46.269782025146924|
|    56.09|  40.9133364268273|
|     44.6|  46.1788110195921|
|    46.98| 43.68159447003018|
|    46.84| 45.99275262397168|
|    43.27| 45.90323542740225|
|    46.91| 45.81089777555644|
|    56.05| 40.30433119032932|
|    45.29| 45.62517158506125|
|    47.52|  43.1994278965376|
|     46.3|45.578494995002984|
|     43.2| 45.53476138143924|
|    55.12| 40.13453762904828|
|    54.43| 40.02514372130645|
|    54.36|39.916981760881754|
|    46.01| 45.12201183288744|
|     45.0| 44.75360850398283|
|    47.02| 42.20776089152969|
|    56.34|30.241103151836796|
+---------+------------------+
only showing top 20 rows

RMSE: 23.780321172709968
MAE: 20.347498369661583
R2: 0.089953423605200 98
```

Streaming

```python
def create_spark_session(app_name):
    """
    Create a Spark session with the given app name and set log level to ERROR.
    """
    spark = SparkSession.builder.appName(app_name).getOrCreate()
    spark.sparkContext.setLogLevel("ERROR")
    return spark



def load_models():
    """
    Load the machine learning models and scaler.
    """
    model = LinearRegressionModel.load(MODEL_LOCATION)
    scaler = StandardScalerModel.load(SCALER_LOCATION)
    indexer = StringIndexerModel.load(INDEXER_LOCATION)
    return model, scaler, indexer
```

Streaming

```
# Define schema for parsing Kafka data
schema = StructType([
    StructField("timestamp", StringType()),
    StructField("id", StringType()),
    StructField("type", StringType()),
    StructField("latitude", StringType()),
    StructField("longitude", StringType()),
    StructField("speed_kmh", StringType()),
    StructField("acceleration", StringType()),
    StructField("distance", StringType()),
    StructField("odometer", StringType()),
    StructField("pos", StringType())
])

# Read data from Kafka
df = (
    spark.readStream.format("kafka")
    .option("kafka.bootstrap.servers", KAFKA_URL)
    .option("subscribe", KAFKA_TOPIC)
    .option("startingOffsets", "latest")
    .load()
)
```

Streaming

```
# Transform the data
parsed_values = df.select(
    "timestamp", from_json(col("value").cast("string"), schema).alias("parsed_values")
)
parsed_values.printSchema()
df_org = parsed_values.select(
    "timestamp",
    "parsed_values.id",
    "parsed_values.type",
    "parsed_values.latitude",
    "parsed_values.longitude",
    "parsed_values.speed_kmh",
    "parsed_values.acceleration",
    "parsed_values.distance",
    "parsed_values.odometer",
    "parsed_values.pos"
)
df_org.printSchema()
print(" DATA TRANSFORMED ")
df_org = df_org.withColumn("latitude", col("latitude").cast("double"))
df_org = df_org.withColumn("longitude", col("longitude").cast("double"))
df_org = df_org.withColumn("speed_kmh", col("speed_kmh").cast("double"))
df_org = df_org.withColumn("acceleration", col("acceleration").cast("double"))
df_org = df_org.withColumn("distance", col("distance").cast("double"))
df_org = df_org.withColumn("odometer", col("odometer").cast("double"))
df_org = df_org.withColumn("pos", col("pos").cast("double"))
```

Streaming

```
# Manipulacija sa kolonom datetime
df_org = df_org.withColumn("datetime", concat(col("timestamp"), lit("Z")))
df_org = df_org.withColumn("datetime", to_timestamp("datetime", "yyyy-MM-dd'T'HH:mm:ss.SSSX"))
df_org = df_org.drop("timestamp")

# Formatiranje datuma
df_org = df_org.withColumn("datetime", date_format("datetime", "yyyy-MM-dd HH:mm:ss"))

# Prikazivanje rezultata

print("Poslednji df")
df_org.printSchema()
indexed = indexer.transform(df_org)

columns = ["latitude", "longitude", "speed_kmh", "acceleration", "distance"]

va = VectorAssembler().setInputCols(columns).setOutputCol('features').setHandleInvalid("skip").transform(indexed)

scaled = scaler.transform(va)

predictions = model.transform(scaled)
print("Stampam predikcije!")
predictions.printSchema()

query = predictions.writeStream \
    .foreach(InfluxDBWriter()) \
    .start()

query.awaitTermination()
```

Streaming

```
 DATA TRANSFORMED
Poslednji df
root
 |-- id: string (nullable = true)
 |-- type: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- speed_kmh: double (nullable = true)
 |-- acceleration: double (nullable = true)
 |-- distance: double (nullable = true)
 |-- odometer: double (nullable = true)
 |-- pos: double (nullable = true)
 |-- datetime: string (nullable = true)

Stampam predikcije!
root
 |-- id: string (nullable = true)
 |-- type: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- speed_kmh: double (nullable = true)
 |-- acceleration: double (nullable = true)
 |-- distance: double (nullable = true)
 |-- odometer: double (nullable = true)
 |-- pos: double (nullable = true)
 |-- datetime: string (nullable = true)
 |-- id_num: double (nullable = false)
 |-- type_num: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- scaled_features: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

Хвала на пажњи