ACM generate XML from customer sales gz files

```
sc.version
```

```
res1: String = 2.4.0
```

```
z.put("acm", "/home/ubuntu/test_spark/acm-xml/")
```

```
%sh
```

```
ls -l {acm}
```

```
total 8
-rw-r--r-- 1 ubuntu ubuntu 597 Jul 17 09:26 customer_sales_20190716032832_00.gz
-rw-r--r-- 1 ubuntu ubuntu 512 Jul 17 09:26 customer_sales_20190716034903_00.gz
```

```
// Read into DF

val df_raw = spark.read.format("json").load(z.get("acm").toString)
```

```
df_raw: org.apache.spark.sql.DataFrame = [address: string, barcode: string ... 27 more fields]
```

```
df_raw.printSchema
```

```
root
 |-- address: string (nullable = true)
 |-- barcode: string (nullable = true)
 |-- canContact: long (nullable = true)
 |-- channelCode: string (nullable = true)
 |-- city: string (nullable = true)
 |-- country: string (nullable = true)
 |-- customerId: long (nullable = true)
 |-- emailAddress: string (nullable = true)
 |-- faceValueFees: long (nullable = true)
 |-- firstName: string (nullable = true)
 |-- lastName: string (nullable = true)
 |-- orderId: string (nullable = true)
 |-- orderlineId: string (nullable = true)
 |-- phoneNumber: string (nullable = true)
 |-- postcode: string (nullable = true)
 |-- price: long (nullable = true)
 |-- priceNet: long (nullable = true)
 |-- priceTypeName: string (nullable = true)
 |-- productId: string (nullable = true)
 |-- productWhen: string (nullable = true)
 |-- productWhere: string (nullable = true)
 |-- productWho: string (nullable = true)
 |-- quantity: long (nullable = true)
 |-- row: string (nullable = true)
 |-- salutation: string (nullable = true)
 |-- seat: string (nullable = true)
 |-- section: string (nullable = true)
 |-- state: string (nullable = true)
 |-- transactionDateTime: string (nullable = true)
```

```
df_raw.show()
```

```
+------------------+--------------------+----------+-----------+---------+---------+----------+--------------------+-------------+---------+--------+-------------+---------------+-----------+--------+
|           address|             barcode|canContact|channelCode|     city|  country|customerId|        emailAddress|faceValueFees|firstName|lastName|      orderId|    orderlineId|phoneNumber|postcode|
+------------------+--------------------+----------+-----------+---------+---------+----------+--------------------+-------------+---------+--------+-------------+---------------+-----------+--------+
|170 Charles Street |93cd75aad9623ef24...|         0|          M|   Seddon|Australia|  14589260|olivia.goodman@me...|          420|   OLIVIA| GOODMAN|20190716,19657|20190716,19657,1| 0403339261|    3011|
|170 Charles Street |653ab073d4005721f...|         0|          M|   Seddon|Australia|  14589260|olivia.goodman@me...|          420|   OLIVIA| GOODMAN|20190716,19657|20190716,19657,1| 0403339261|    3011|
|170 Charles Street |0841c8fce610978e3...|         0|          M|   Seddon|Australia|  14589260|olivia.goodman@me...|          400|   OLIVIA| GOODMAN|20190716,19657|20190716,19657,1| 0403339261|    3011|
|  20 Cusdin Street |21b528c39a9526965...|         0|          W|Glen Iris|Australia|   9007250|rodw@actionalumin...|          510|      Rod|  Walker|20190716,20663|20190716,20663,1| 0439660998|    3146|
|  20 Cusdin Street |c7e40b36bfebbb4e4...|         0|          W|Glen Iris|Australia|   9007250|rodw@actionalumin...|          510|      Rod|  Walker|20190716,20663|20190716,20663,1| 0439660998|    3146|
+------------------+--------------------+----------+-----------+---------+---------+----------+--------------------+-------------+---------+--------+-------------+---------------+-----------+--------+
```

```
df_raw.take(2)
```

```
res56: Array[org.apache.spark.sql.Row] = Array([170 Charles Street ,93cd75aad9623ef242f808665d2f2be0a8d912c4,0,M,Seddon,Australia,14589260,olivia.goodman@me.com,420,OLIVIA,GOODMAN,20190716,19657,20190716
```

```
// groupby address

df_raw.groupByKey(r => r.getString(0)).keys.show()
```

```
+------------------+
|             value|
+------------------+
|170 Charles Street |
|  20 Cusdin Street |
+------------------+
```

```
case class CustomerSales(address: String, barcode: String, canContact: Long, channelCode: String, city: String, country: String, customerId: Long, emailAddress: String, faceValueFees: Long, firstName: St
```

```
defined class CustomerSales
```

```
val ds = spark
            .read
            .format("json")
            .load(z.get("acm").toString)
            .as[CustomerSales]
```

```
ds: org.apache.spark.sql.Dataset[CustomerSales] = [address: string, barcode: string ... 27 more fields]
```

```
ds.printSchema
```

```
root
 |-- address: string (nullable = true)
 |-- barcode: string (nullable = true)
 |-- canContact: long (nullable = true)
 |-- channelCode: string (nullable = true)
 |-- city: string (nullable = true)
 |-- country: string (nullable = true)
 |-- customerId: long (nullable = true)
 |-- emailAddress: string (nullable = true)
 |-- faceValueFees: long (nullable = true)
 |-- firstName: string (nullable = true)
 |-- lastName: string (nullable = true)
 |-- orderId: string (nullable = true)
 |-- orderlineId: string (nullable = true)
 |-- phoneNumber: string (nullable = true)
 |-- postcode: string (nullable = true)
 |-- price: long (nullable = true)
 |-- priceNet: long (nullable = true)
 |-- priceTypeName: string (nullable = true)
 |-- productId: string (nullable = true)
 |-- productWhen: string (nullable = true)
 |-- productWhere: string (nullable = true)
 |-- productWho: string (nullable = true)
 |-- quantity: long (nullable = true)
 |-- row: string (nullable = true)
 |-- salutation: string (nullable = true)
 |-- seat: string (nullable = true)
 |-- section: string (nullable = true)
 |-- state: string (nullable = true)
 |-- transactionDateTime: string (nullable = true)
```

```
ds.show()
```

```
+------------------+--------------------+----------+-----------+---------+---------+----------+--------------------+-------------+---------+--------+-------------+---------------+-----------+--------+
|           address|             barcode|canContact|channelCode|     city|  country|customerId|        emailAddress|faceValueFees|firstName|lastName|      orderId|    orderlineId|phoneNumber|postcode|
+------------------+--------------------+----------+-----------+---------+---------+----------+--------------------+-------------+---------+--------+-------------+---------------+-----------+--------+
|170 Charles Street |93cd75aad9623ef24...|         0|          M|   Seddon|Australia|  14589260|olivia.goodman@me...|          420|   OLIVIA| GOODMAN|20190716,19657|20190716,19657,1| 0403339261|    3011|
|170 Charles Street |653ab073d4005721f...|         0|          M|   Seddon|Australia|  14589260|olivia.goodman@me...|          420|   OLIVIA| GOODMAN|20190716,19657|20190716,19657,1| 0403339261|    3011|
|170 Charles Street |0841c8fce610978e3...|         0|          M|   Seddon|Australia|  14589260|olivia.goodman@me...|          400|   OLIVIA| GOODMAN|20190716,19657|20190716,19657,1| 0403339261|    3011|
|  20 Cusdin Street |21b528c39a9526965...|         0|          W|Glen Iris|Australia|   9007250|rodw@actionalumin...|          510|      Rod|  Walker|20190716,20663|20190716,20663,1| 0439660998|    3146|
|  20 Cusdin Street |c7e40b36bfebbb4e4...|         0|          W|Glen Iris|Australia|   9007250|rodw@actionalumin...|          510|      Rod|  Walker|20190716,20663|20190716,20663,1| 0439660998|    3146|
+------------------+--------------------+----------+-----------+---------+---------+----------+--------------------+-------------+---------+--------+-------------+---------------+-----------+--------+
```

```
// Group by address

ds.groupByKey(cs => cs.address).keys.show()

+------------------+
|             value|
+------------------+
|170 Charles Street |
|  20 Cusdin Street |
+------------------+

// Group by multiple keys

ds.groupByKey(cs => (cs.address, cs.customerId)).keys.show()

+------------------+--------+
|                _1|      _2|
+------------------+--------+
|170 Charles Street |14589260|
|  20 Cusdin Street | 9007250|
+------------------+--------+

grouped: Unit = ()

val grouped1 = ds.groupByKey(cs => (
        cs.customerId,
        cs.orderlineId,
        cs.firstName,
        cs.lastName,
        cs.address,
        cs.city,
        cs.state,
        cs.postcode,
        cs.country,
        cs.phoneNumber,
        cs.emailAddress,
        cs.transactionDateTime,
        cs.channelCode,
        cs.canContact,
        cs.productId,
        cs.productWho,
        cs.productWhere,
        cs.productWhen
    ))

grouped1: org.apache.spark.sql.KeyValueGroupedDataset[(Long, String, String, String, String, String, String, String, String, String, String, String, String, String, Long, String, String, String, String),Customer

// Enough to group by orderlineId, customerID
// NOTE: also repartitioning to optimize (might not be needed)
// Also, maintaining 1 partition as its a very small dataset
val grouped = ds.repartition(1, $"orderlineId").groupByKey(cs => (cs.orderlineId, cs.customerId, cs.productId))

grouped: org.apache.spark.sql.KeyValueGroupedDataset[(String, Long, String),CustomerSales] = KeyValueGroupedDataset: [key: [_1: string, _2: bigint ... 1 more field(s)], value: [address: string, barcode:

grouped.keys.show()

+---------------+--------+------------+
|             _1|      _2|          _3|
+---------------+--------+------------+
|20190716,20663,1| 9007250|EVAM2019912DC|
|20190716,19657,1|14589260|  EVAM2019950|
+---------------+--------+------------+

// Create schema as per XML output

import org.apache.spark.sql.types._

val schema = new StructType()
                        .add("import_account_id", LongType, false)
                        .add("import_ref_no", StringType, false)
                        .add("fname", StringType, false)
                        .add("lname", StringType, false)
                        .add("street1", StringType, false)
                        .add("city", StringType, false)
                        .add("state", StringType, false)
                        .add("postal_code", StringType, false)
                        .add("country", StringType, false)
                        .add("phone", StringType, false)
                        .add("eaddress", StringType, false)
                        .add("order_dt", StringType, false)
                        .add("mos", StringType, false)
                        .add("opt_in", BooleanType, false)
                        .add("event", MapType(
                             StringType,
                             new StructType()
                                        .add("perf_code", StringType, false)
                                        .add("perf_name", StringType, false)
                                        .add("perf_dt", StringType, false)
                                        .add("venue", StringType, false)
                                        .add("seat_info", ArrayType(
                                               new StructType()
                                                           .add("section", StringType, false)
                                                           .add("row", StringType, false)
                                                           .add("seat", StringType, false)
                                                           .add("sale_refund", StringType, false)
                                                           .add("price_type", StringType, false)
                                                           .add("gross_price", LongType, false)
                                                           .add("barcode", StringType, false)
                                           ), false)

                            ), false)

import org.apache.spark.sql.types._
schema: org.apache.spark.sql.types.StructType = StructType(StructField(import_account_id,LongType,false), StructField(import_ref_no,StringType,false), StructField(fname,StringType,false), StructField(lna

case class SeatInfo(val section: String, val row: String, val seat: String, val sale_refund: String, val price_type: String, val gross_price: Long, val barcode: String)
case class Event(val perf_code: String, val perf_name: String, val perf_dt: String, val venue: String, val seat_info: Array[SeatInfo])
case class Client(val import_account_id: Long, val import_ref_no: String, val fname: String, val lname: String, val street1: String, val city: String, val state: String, val postal_code: String, val coun

defined class SeatInfo
defined class Event
defined class Client

case class TestXml(val import_account_id: Long, val import_ref_no: String)

defined class TestXml

def clientMapper(k: (String, Long, String), vals: Iterator[CustomerSales]): Iterator[Client] = {
    val oid = k._1
    val cid = k._2
    val pid = k._3

    val seatInfo = vals.map{cs => {
        SeatInfo(
            section = cs.section,
            row = cs.row,
            seat = cs.seat,
            sale_refund = if (cs.quantity == 1) "sale" else "refund",
            price_type = cs.priceTypeName,
            gross_price = cs.price + cs.faceValueFees,
            barcode = cs.barcode
        )
    }}

    val cs = vals.next

    val e = Event(
            perf_code = pid,
```

```
            perf_name = cs.productWho,
            perf_dt = cs.productWhen,
            venue = cs.productWhere,
            seat_info = seatInfo.toArray
        )

    Iterator(
        Client(
            import_account_id = cid,
            import_ref_no = oid,
            fname = cs.firstName,
            lname = cs.lastName,
            street1 = cs.address.trim.slice(0, 64),
            city = cs.city,
            state = cs.state,
            postal_code = cs.postcode,
            country = cs.country,
            phone = cs.phoneNumber,
            eaddress = cs.emailAddress,
            order_dt = cs.transactionDateTime,
            mos = cs.channelCode,
            opt_in = if (cs.canContact == 1) true else false,
            event = e
        )
    )
}

clientMapper: (k: (String, Long, String), vals: Iterator[CustomerSales])Iterator[Client]

case class TestXml1(val import_account_id: Long, val import_ref_no: String, val perf_code: String, val perf_name: String, val seat_info: Array[SeatInfo])

def clientMapper1(k: (String, Long, String), vals: Iterator[CustomerSales]): Iterator[TestXml1] = {
    val oid = k._1
    val cid = k._2
    val pid = k._3

    val seatInfo = vals.map(cs => {
        SeatInfo(
            section = cs.section,
            row = cs.row,
            seat = cs.seat,
            sale_refund = if (cs.quantity == 1) "sale" else "refund",
            price_type = cs.priceTypeName,
            gross_price = cs.price + cs.faceValueFees,
            barcode = cs.barcode
        )
    }}

    val cs = vals.next

    // val e = Event(
    //         perf_code = pid,
    //         perf_name = cs.productWho,
    //         perf_dt = cs.productWhen,
    //         venue = cs.productWhere,
    //         seat_info = seatInfo
    //     )

    // Iterator(
    //     Client(
    //         import_account_id = cid,
    //         import_ref_no = oid,
    //         fname = cs.firstName,
    //         lname = cs.lastName,
    //         street1 = cs.address,
    //         city = cs.city,
    //         state = cs.state,
    //         postal_code = cs.postcode,
    //         country = cs.country,
    //         phone = cs.phoneNumber,
    //         eaddress = cs.emailAddress,
    //         order_dt = cs.transactionDateTime,
    //         mos = cs.channelCode,
    //         opt_in = if (cs.canContact == 1) true else false,
    //         event = e
    //     )
    // )

    Iterator(
        TestXml1(
            import_account_id = cid,
            import_ref_no = oid,
            perf_code = pid,
            perf_name = cs.productWho,
            seat_info = seatInfo.toArray
        )
    )
}

defined class TestXml1
clientMapper1: (k: (String, Long, String), vals: Iterator[CustomerSales])Iterator[TestXml1]

val groupedDS1 = grouped.flatMapGroups((k, viter) => Iterator(TestXml(k._2, k._1)))

groupedDS1: org.apache.spark.sql.Dataset[TestXml] = [import_account_id: bigint, import_ref_no: string]

groupedDS1.show()

+----------------+---------------+
|import_account_id|   import_ref_no|
+----------------+---------------+
|         9007250|20190716,20663,1|
|        14589260|20190716,19657,1|
+----------------+---------------+

// Convert each group into Dataset[Client]

val groupedDS = grouped.flatMapGroups(clientMapper)

groupedDS: org.apache.spark.sql.Dataset[Client] = [import_account_id: bigint, import_ref_no: string ... 13 more fields]

groupedDS.show

+----------------+---------------+------+-------+-----------------+---------+-----+-----------+---------+----------+--------------------+--------------------+---+------+--------------------+
|import_account_id|   import_ref_no| fname|  lname|          street1|     city|state|postal_code|  country|     phone|            eaddress|          order_dt|mos|opt_in|               event|
+----------------+---------------+------+-------+-----------------+---------+-----+-----------+---------+----------+--------------------+--------------------+---+------+--------------------+
|         9007250|20190716,20663,1|   Rod| Walker|  20 Cusdin Street|Glen Iris|  VIC|       3146|Australia|0439660998|rodw@actionalumin...|2019-07-16T03:48:45Z|  W| false|[EVAM2019912DC, D...|
|        14589260|20190716,19657,1|OLIVIA|GOODMAN|170 Charles Street|   Seddon|  VIC|       3011|Australia|0403339261|olivia.goodman@me...|2019-07-16T03:27:59Z|  M| false|[EVAM2019950, HUM...|
+----------------+---------------+------+-------+-----------------+---------+-----+-----------+---------+----------+--------------------+--------------------+---+------+--------------------+

// Accessing elements in nested structures

groupedDS.select($"import_account_id", $"import_ref_no", $"event.perf_code", $"event.seat_info", $"event.seat_info.sale_refund").show

+----------------+---------------+------------+--------------------+------------+
|import_account_id|   import_ref_no|    perf_code|           seat_info| sale_refund|
+----------------+---------------+------------+--------------------+------------+
|         9007250|20190716,20663,1|EVAM2019912DC|[[SSTALLS, P, 29,...|      [sale]|
|        14589260|20190716,19657,1|  EVAM2019950|[[SSTALLS-2, A, 2...|[sale, sale]|
+----------------+---------------+------------+--------------------+------------+

groupedDS.printSchema

root
 |-- import_account_id: long (nullable = false)
 |-- import_ref_no: string (nullable = true)
 |-- fname: string (nullable = true)
```

```
 |-- lname: string (nullable = true)
 |-- street1: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- postal_code: string (nullable = true)
 |-- country: string (nullable = true)
 |-- phone: string (nullable = true)
 |-- eaddress: string (nullable = true)
 |-- order_dt: string (nullable = true)
 |-- mos: string (nullable = true)
 |-- opt_in: boolean (nullable = false)
 |-- event: struct (nullable = true)
 |    |-- perf_code: string (nullable = true)
 |    |-- perf_name: string (nullable = true)
 |    |-- perf_dt: string (nullable = true)
 |    |-- venue: string (nullable = true)
 |    |-- seat_info: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- section: string (nullable = true)
 |    |    |    |-- row: string (nullable = true)
 |    |    |    |-- seat: string (nullable = true)
 |    |    |    |-- sale_refund: string (nullable = true)
 |    |    |    |-- price_type: string (nullable = true)
 |    |    |    |-- gross_price: long (nullable = false)
 |    |    |    |-- barcode: string (nullable = true)
```

Convert dataframe/dataset to XML

```
groupedDS1.rdd.partitions.size
```

```
res18: Int = 200
```

Reducing number of partitions before writing.

- **repartition** will shuffle and create equal partitions.
- **coalesce** won't shuffle and data won't be evenly distributed.
- In this case coalesce is okay as we are reducing the number of partitions to just 1.

```
import com.databricks.spark.xml._

groupedDS1.coalesce(1)
                .write
                     .option("rootTag", "data")
                     .option("rowTag", "client")
                     .xml("grouped_ds1_xml")
```

```
import com.databricks.spark.xml._
```

```
%sh

cat grouped_ds1_xml/part-00000
```

```
<data>
    <client>
        <import_account_id>9007250</import_account_id>
        <import_ref_no>20190716,20663,1</import_ref_no>
    </client>

    <client>
        <import_account_id>14589260</import_account_id>
        <import_ref_no>20190716,19657,1</import_ref_no>
    </client>

</data>
```

```
import com.databricks.spark.xml._

groupedDS
        .coalesce(1)
        .write
            .option("rootTag", "data")
            .option("rowTag", "client")
            .xml("grouped_ds_xml")
```

```
import com.databricks.spark.xml._
```

```
%sh

cat grouped_ds_xml/part-00000
```

```
<data>
    <client>
        <import_account_id>9007250</import_account_id>
        <import_ref_no>20190716,20663,1</import_ref_no>
        <fname>Rod</fname>
        <lname>Walker</lname>
        <street1>20 Cusdin Street</street1>
        <city>Glen Iris</city>
        <state>VIC</state>
        <postal_code>3146</postal_code>
        <country>Australia</country>
        <phone>0439660998</phone>
        <eaddress>rodw@actionaluminium.com.au</eaddress>
        <order_dt>2019-07-16T03:48:45Z</order_dt>
        <mos>W</mos>
        <opt_in>false</opt_in>
        <event>
            <perf_code>EVAM2019912DC</perf_code>
            <perf_name>DAVID CAMPBELL</perf_name>
            <perf_dt>Thu 24 Oct 2019 8:00pm</perf_dt>
            <venue>Hamer Hall</venue>
            <seat_info>
                <section>SSTALLS</section>
                <row>P</row>
                <seat>29</seat>
                <sale_refund>sale</sale_refund>
                <price_type>Admit</price_type>
                <gross_price>10230</gross_price>
                <barcode>c7e40b36bfebbb4e4f0d132193b1f9725daf60b0</barcode>
            </seat_info>
        </event>
    </client>

    <client>
        <import_account_id>14589260</import_account_id>
        <import_ref_no>20190716,19657,1</import_ref_no>
        <fname>OLIVIA</fname>
        <lname>GOODMAN</lname>
        <street1>170 Charles Street</street1>
        <city>Seddon</city>
        <state>VIC</state>
        <postal_code>3011</postal_code>
        <country>Australia</country>
        <phone>0403339261</phone>
        <eaddress>olivia.goodman@me.com</eaddress>
        <order_dt>2019-07-16T03:27:59Z</order_dt>
        <mos>M</mos>
        <opt_in>false</opt_in>
        <event>
            <perf_code>EVAM2019950</perf_code>
            <perf_name>HUMANS BY CIRCA</perf_name>
            <perf_dt>Sat 30 Nov 2019 7:30pm</perf_dt>
            <venue>Playhouse</venue>
            <seat_info>
                <section>SSTALLS-2</section>
```

```
                <row>A</row>
                <seat>21</seat>
                <sale_refund>sale</sale_refund>
                <price_type>Adult - My Ticketek Presale</price_type>
                <gross_price>4990</gross_price>
                <barcode>653ab073d4005721f029d29b44cc45b75a0e0d7c</barcode>
            </seat_info>
            <seat_info>
                <section>SSTALLS-2</section>
                <row>A</row>
                <seat>22</seat>
                <sale_refund>sale</sale_refund>
                <price_type>Junior (3 to 16 years) - My Ticketek Presale</price_type>
                <gross_price>3970</gross_price>
                <barcode>0841c8fce610978e39d014556e07fd15683b8c8a</barcode>
            </seat_info>
        </event>
    </client>

</data>

%sh
```