Smarkets

ETL pipeline to achieve reliability at scale

By Isabel López Andrade

Accounting at Smarkets

	account_id	source	money	timestamp
0	14302286364420949	deposit	£20.11	2018-02-01 12:47:37.039161
1	14302286364420949	order.execute	£20.11	2018-02-01 13:14:36.794810
2	14302286364420949	order.book.accept	£20.11	2018-02-01 13:14:36.794810
3	14302286364420949	order.create	£20.11	2018-02-01 13:14:36.794810
4	14302286364420949	market.settle	£10.91	2018-02-01 13:25:08.737379
5	14302286364420949	order.execute	£10.91	2018-02-01 13:28:20.156321
6	14302286364420949	order.book.accept	£10.91	2018-02-01 13:28:20.156321

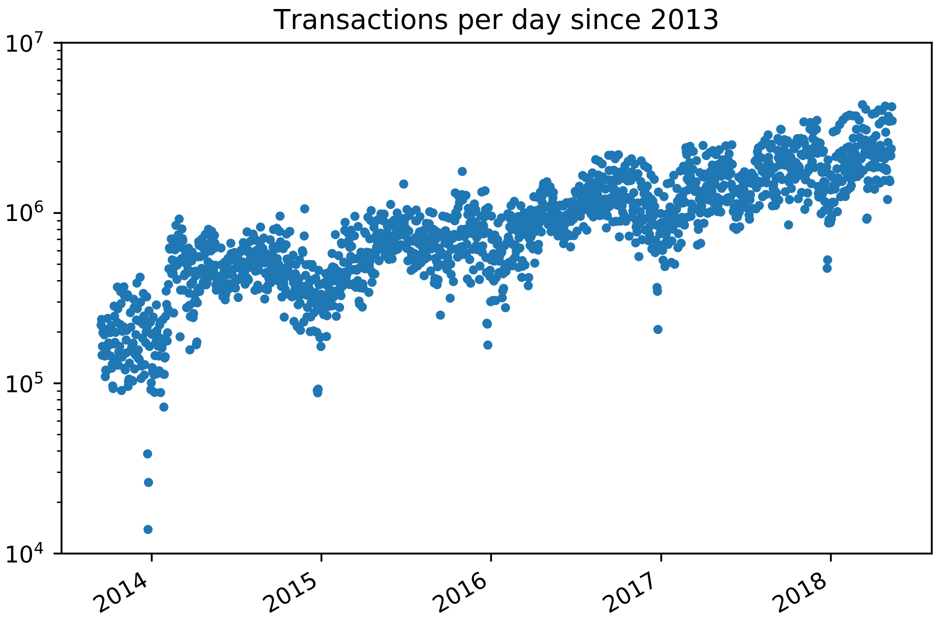
	account_id	year	month	stake	deposit	withdraw
0	14302286364420949	2018	1	£10.00	£50.00	£0.00
1	14302286364420949	2018	2	£15.00	£10.00	£0.00
2	14302286364420949	2018	3	£5.00	£0.00	£40.00
3	14302286364420949	2018	4	£30.00	£0.00	£10.00
4	14302286364420949	2018	5	£10.00	£0.00	£0.00
5	14302286364420949	2018	6	£10.00	£10.00	£0.00

Accounting at Smarkets

Reports generated daily using transactions from the exchange.

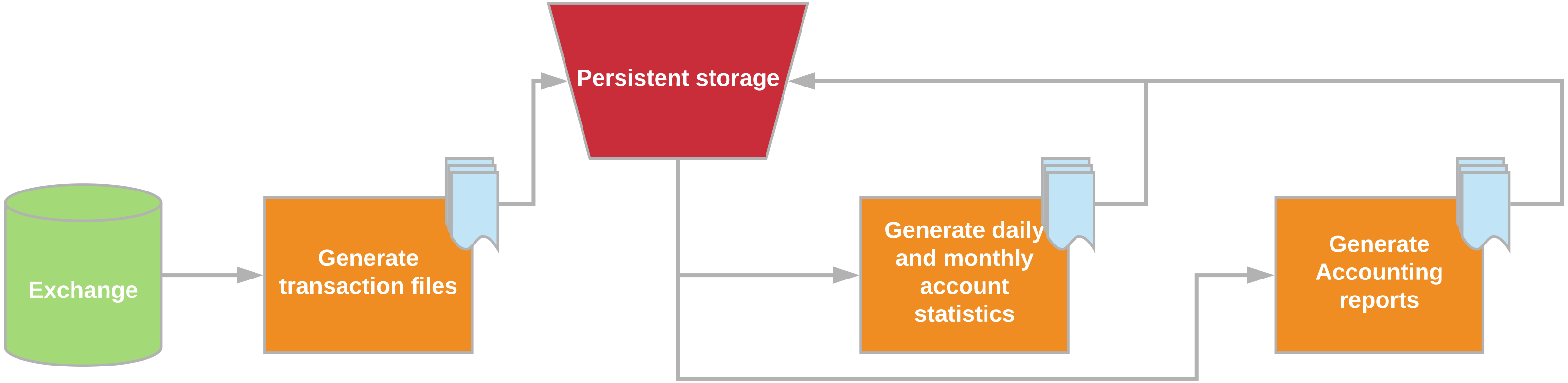
In 2013, the average number of daily transactions was under 190K.

In 2018, this figure is over 8.8M.



💾 Original pipeline

- Difficult to identify errors. 🔍
- Manual work to regenerate reports and expert knowledge of the system. 🤯
- System too slow and unable to scale. It took more than one day to run. 🐢
- Costly storage. 💸



Requirements

- Fault tolerance and reliability.
- Fast io, availability, durability, and cost efficient.
- Good processing performance.
- Scalable.

Fault tolerance and reliability

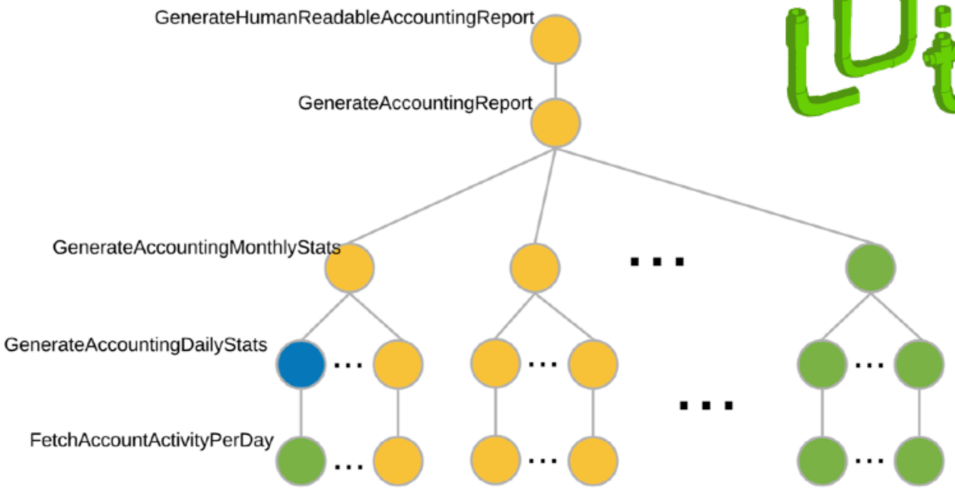
Vulnerabilities

- Communication with exchange may fail.
- Hardware or software errors may happen while the job is running.

Design solutions

- Store transactions per day.
- Compute financial statistics per day.
- Retrieve the last two days worth of transactions.
- Break the accounting job into modular Luigi tasks.

```python
class GenerateHumanReadableAccountingReport(AccountingTask):

    def requires(self) -> luigi.Task:
        return GenerateAccountingReport()

    def run(self) -> None:
        with self.input().operate('r') as target_path:
            df_accounting = pd.read_parquet(target_path)

        with self.output().open('w') as file_:
            df_accounting.to_csv(file_, sep='\t', index=False)

    def output(self) -> luigi.Target:
        return self.get_target(path='data/reports/accounting-report.tsv')
```



Efficient storage

- Columnar storage.
- Only read the columns needed for the task.
- Minimised I/O.
- Efficient compression and encoding.
- Python support.

Parquet

account	source	amount
1	deposit	30
2	deposit	50
1	bet	15
1	withdraw	60
2	bet	40

Row-based

1	deposit	30	2	deposit	50	1	bet	15	1	withdraw	60	2	bet	40

Column-based

1	2	1	1	2	deposit	deposit	bet	withdraw	bet	30	50	15	60	40

Efficient storage

- High durability.
- High availability.
- Low maintenance.
- Cost efficient.
- Decoupling of processing and storage.
- Python library boto/boto3.
- Web interface.

Good performance

Requirements

- Fast data processing.
- Scalable.

Solution

- General purpose data processing engine.
- Massive parallel. Spark builds its own execution plans.
- Caches data in RAM.
- Python support.

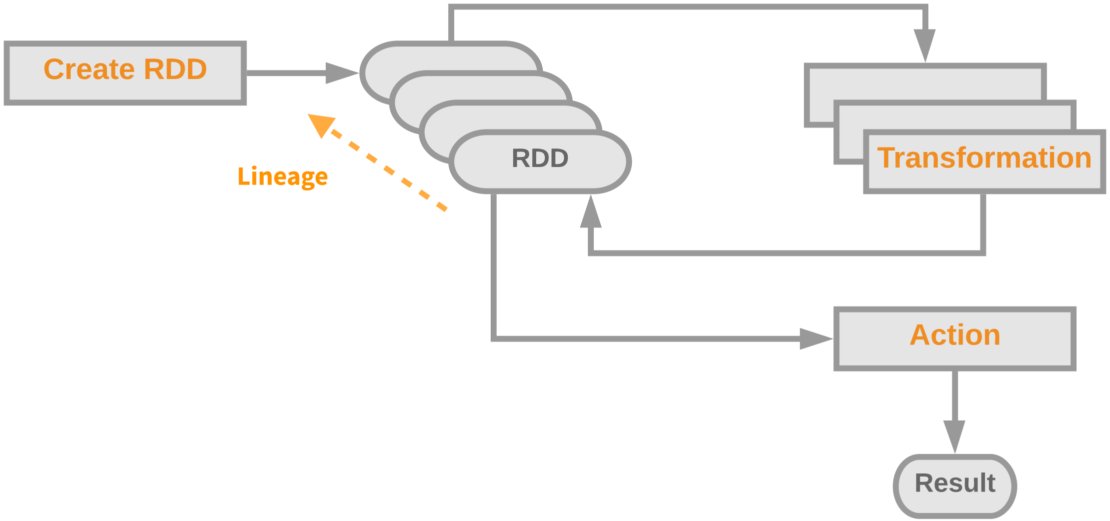
Spark key concepts

RDD

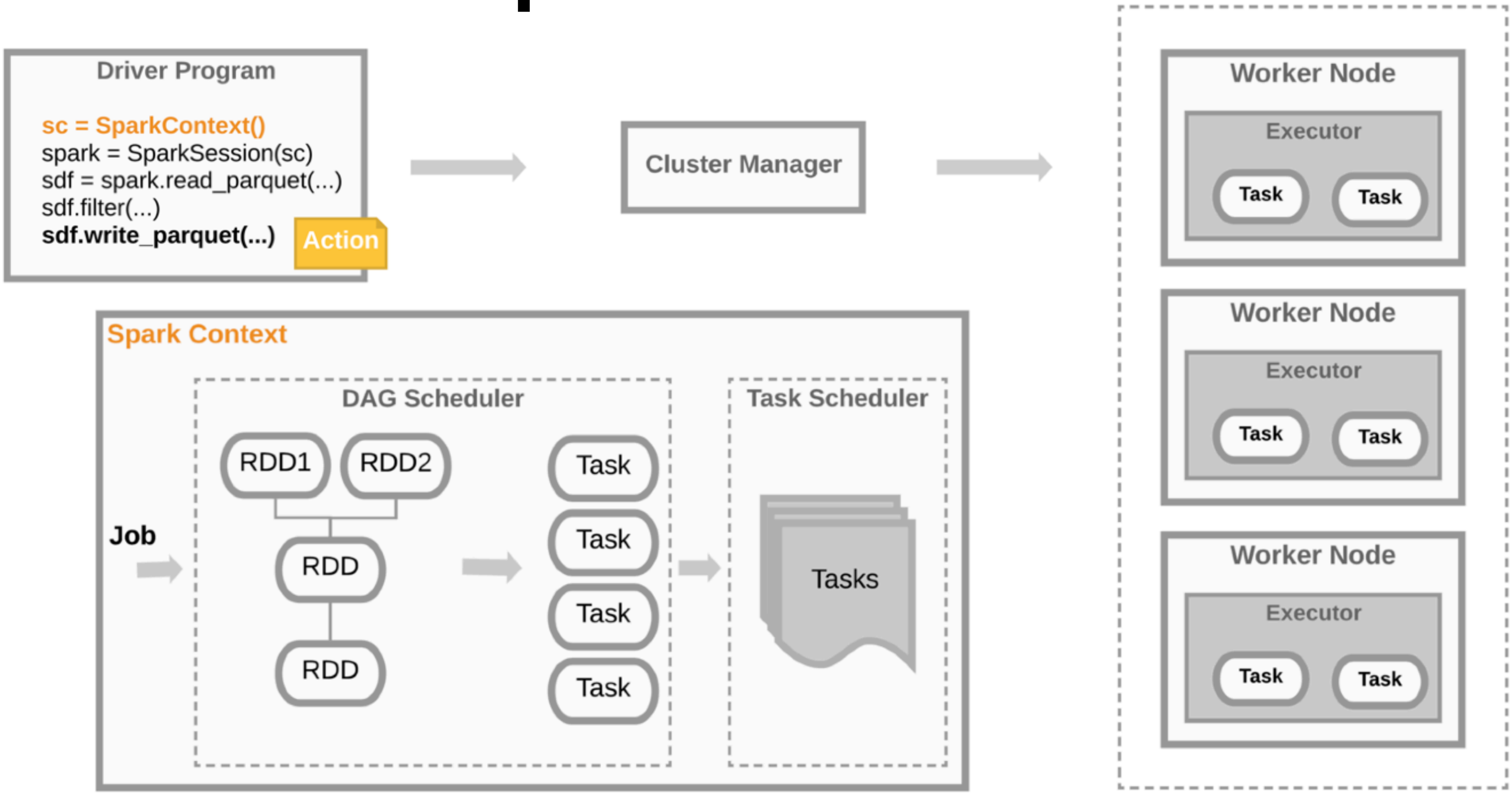
Resilient: fault-tolerant.
Distributed: partitioned across multiple nodes.
Dataset: collection of data.

Dataframes

Data organised in columns built on top of RDDs.
Better performance than RDDs.
User friendly API.



Execution on Spark

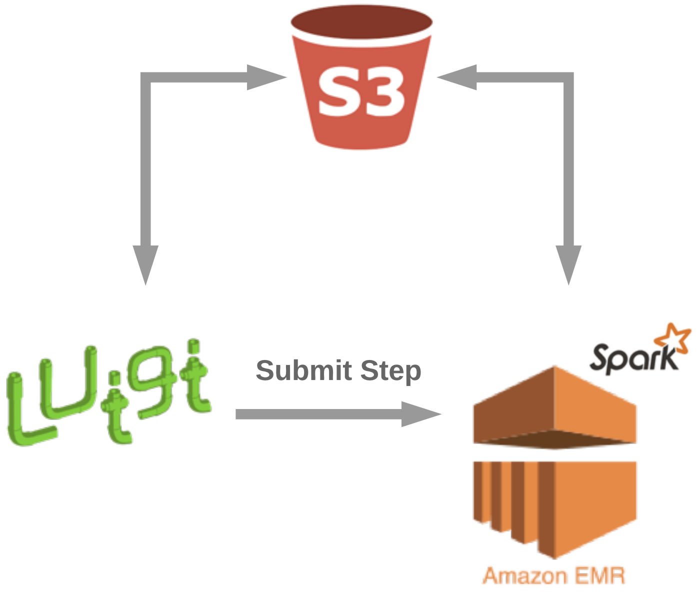


Spark job from Luigi

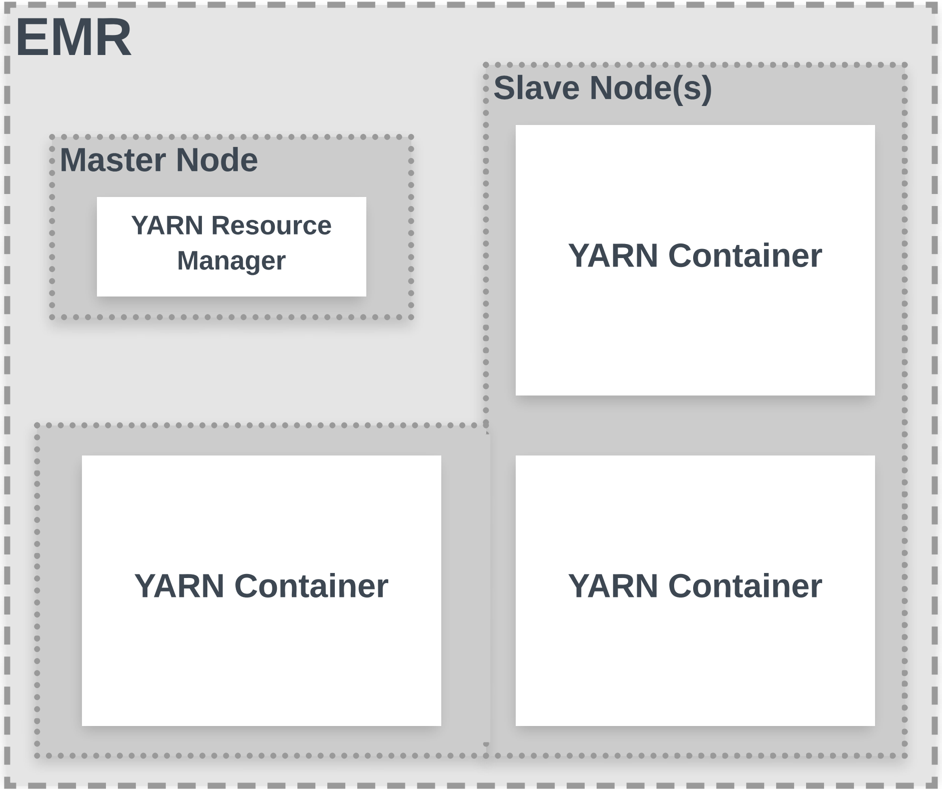
```
class GenerateSmarketsAccountReport(PySparkTask, AccountingTask):

    def requires(self) -> luigi.Task:
        return GenerateAccountingReport()

    def main(self, sc: pyspark.SparkContext) -> None:
        spark = pyspark.sql.SparkSession(sc)
        sdf_per_account = read_parquet(spark, self.input())
        sdf_smarkets = sdf_per_account.filter(
          sdf_per_account.account_id == SMARKETS_ACCOUNT_ID
        )
        write_parquet(sdf_smarkets, self.output())

    def output(self) -> luigi.Target:
        return self.get_target(
          path='data/reports/accounting-report-smarkets.parquet'
        )
```

Scalability

- Spark cluster.
- Fast deployment.
- Easy to use.
- Flexible.
- Seamless integration with S3 - EMRFS.
- Ability to shutdown the cluster when job is done without data loss.
- Low cost.
- Nice web interface.

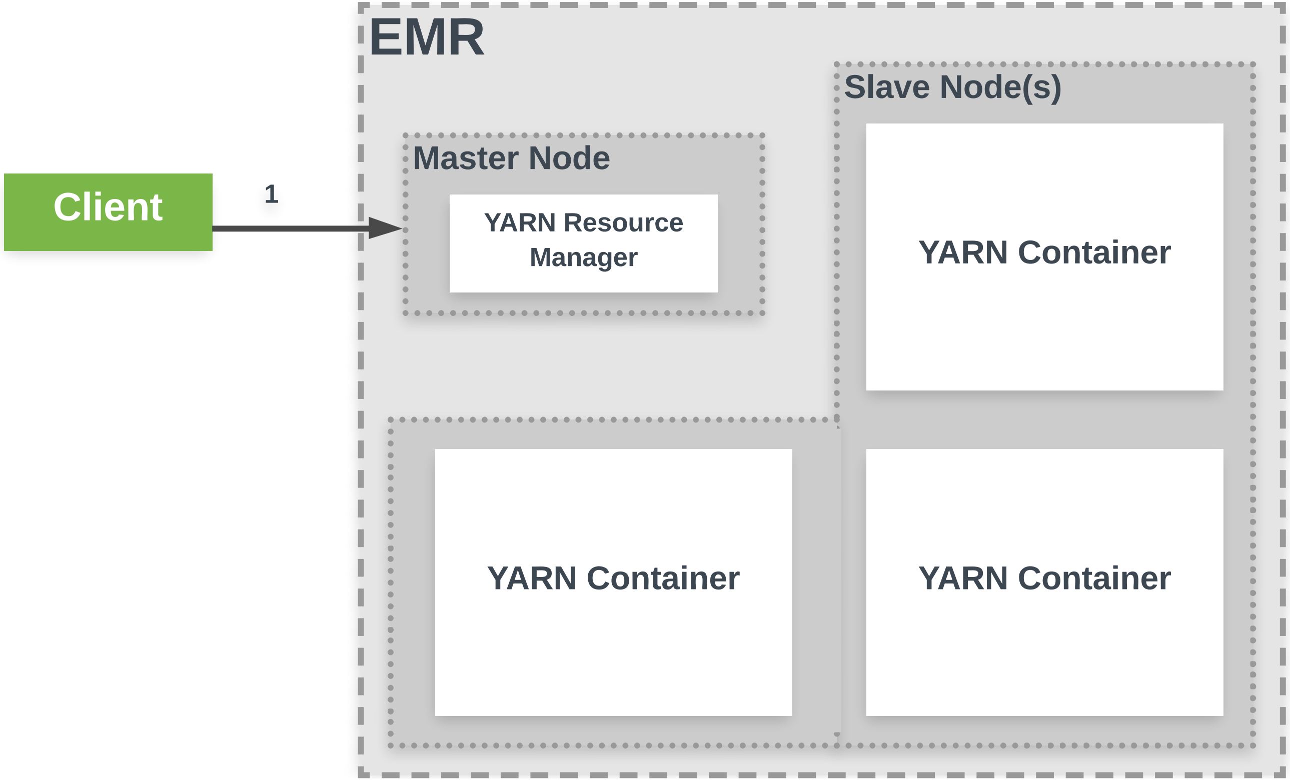


Spark on EMR



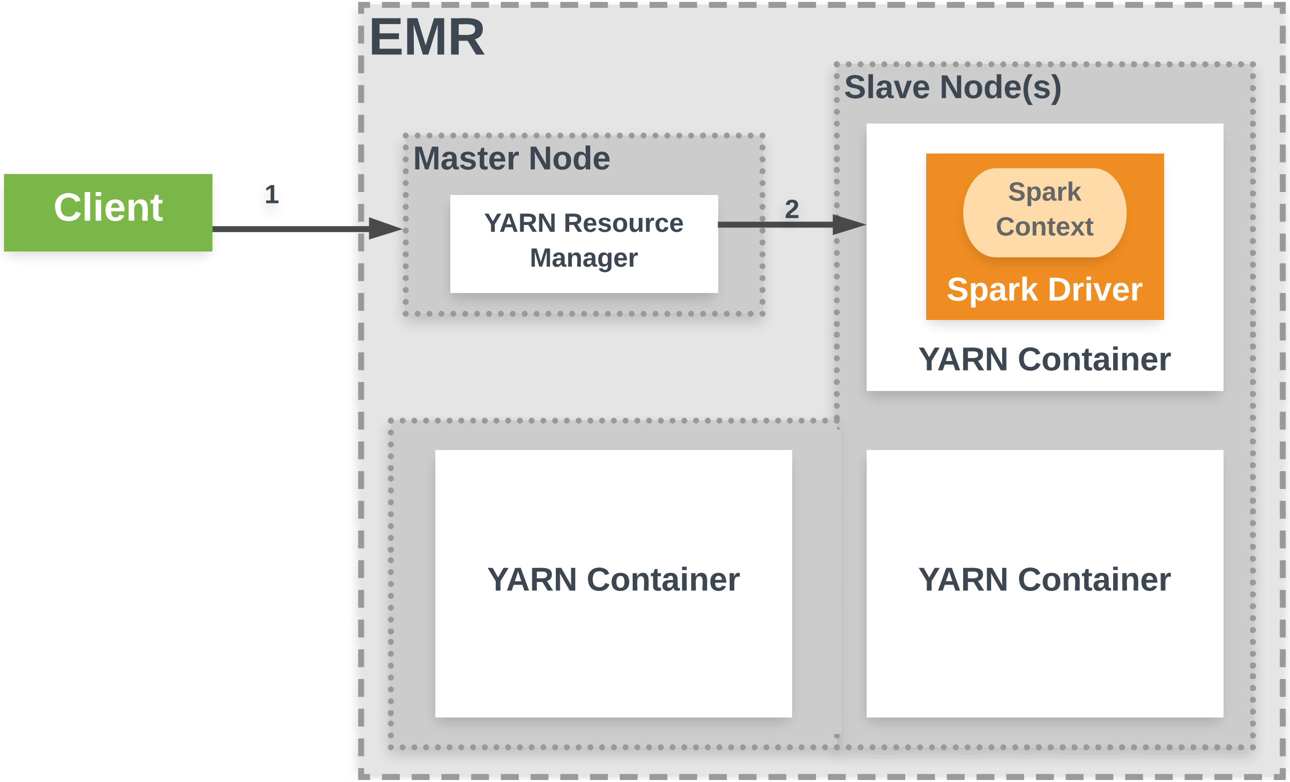
EMR
Slave Node(s)
Master Node
YARN Resource Manager
YARN Container
YARN Container
YARN Container

Spark on EMR

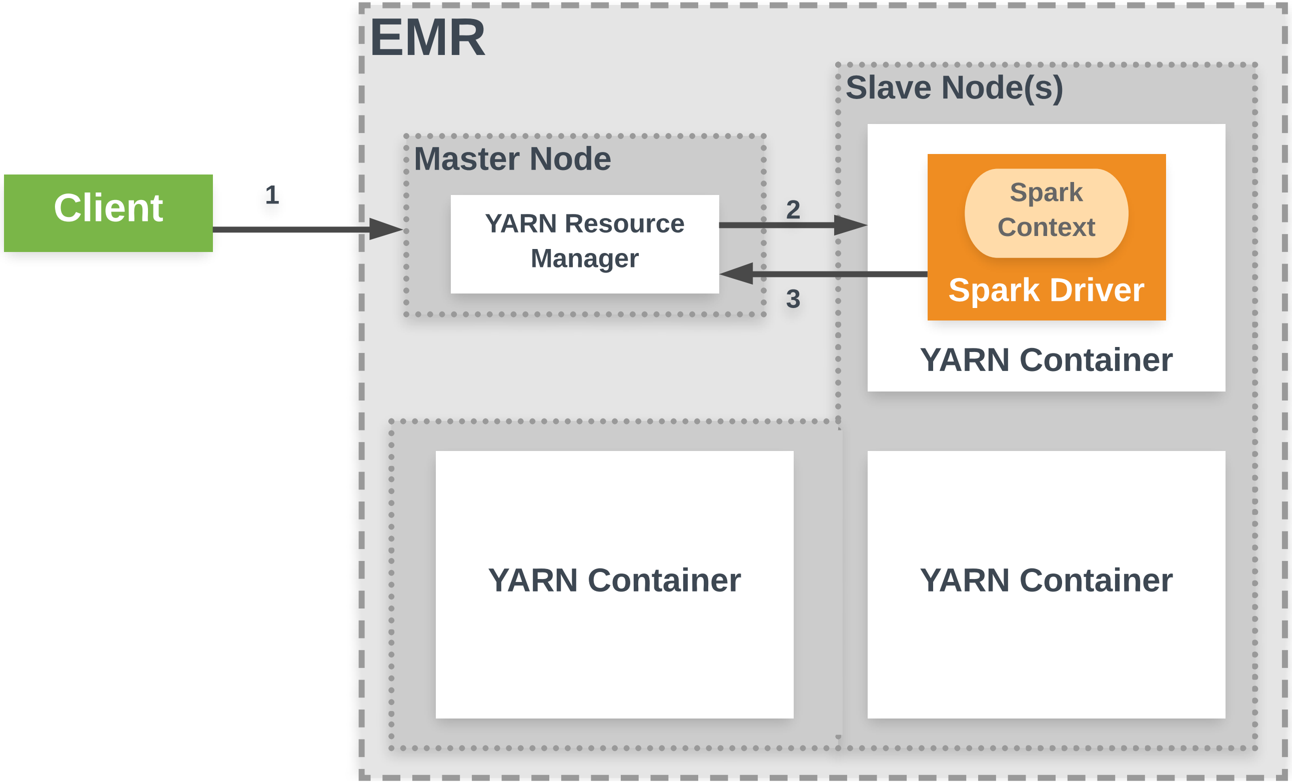


Client
1
EMR
Master Node
YARN Resource Manager
Slave Node(s)
YARN Container
YARN Container
YARN Container

Spark on EMR

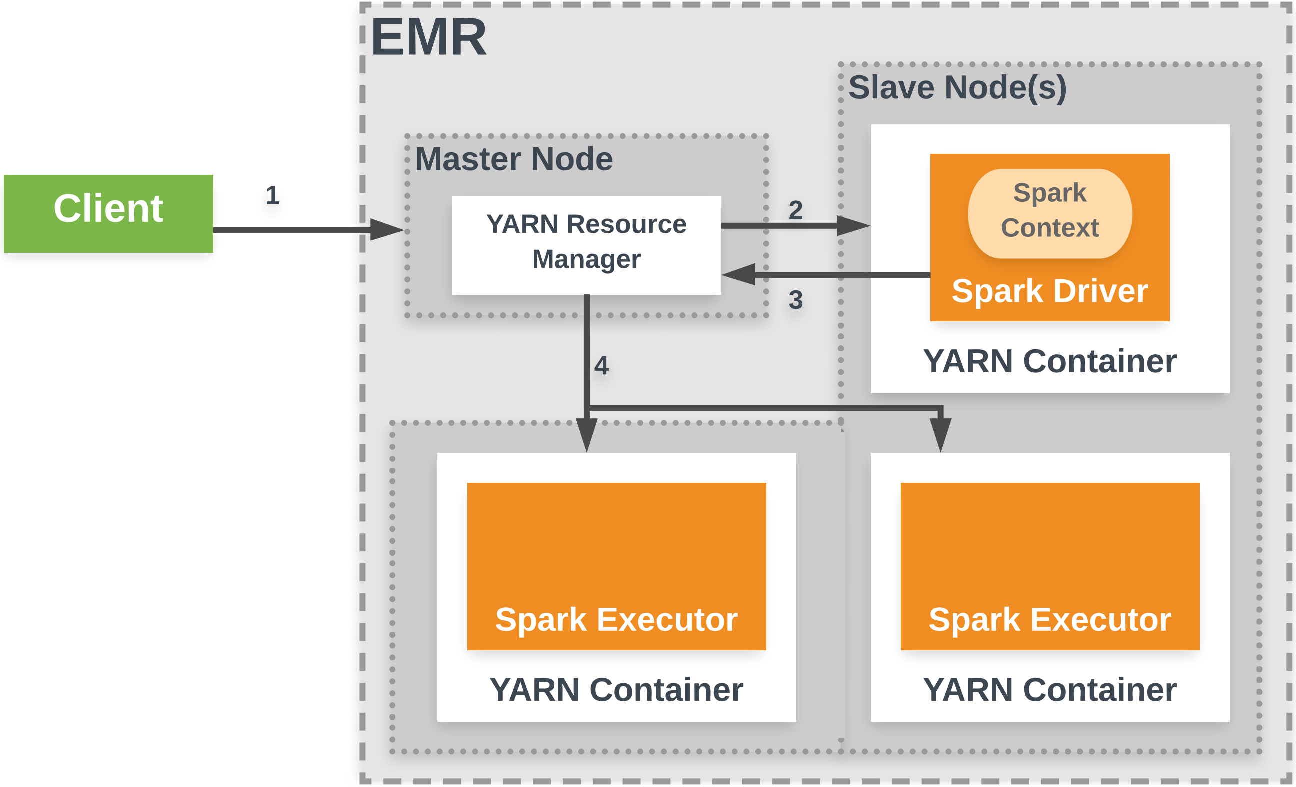

Client
1
EMR
Master Node
YARN Resource Manager
2
Slave Node(s)
Spark Context
Spark Driver
YARN Container
YARN Container
YARN Container

Spark on EMR

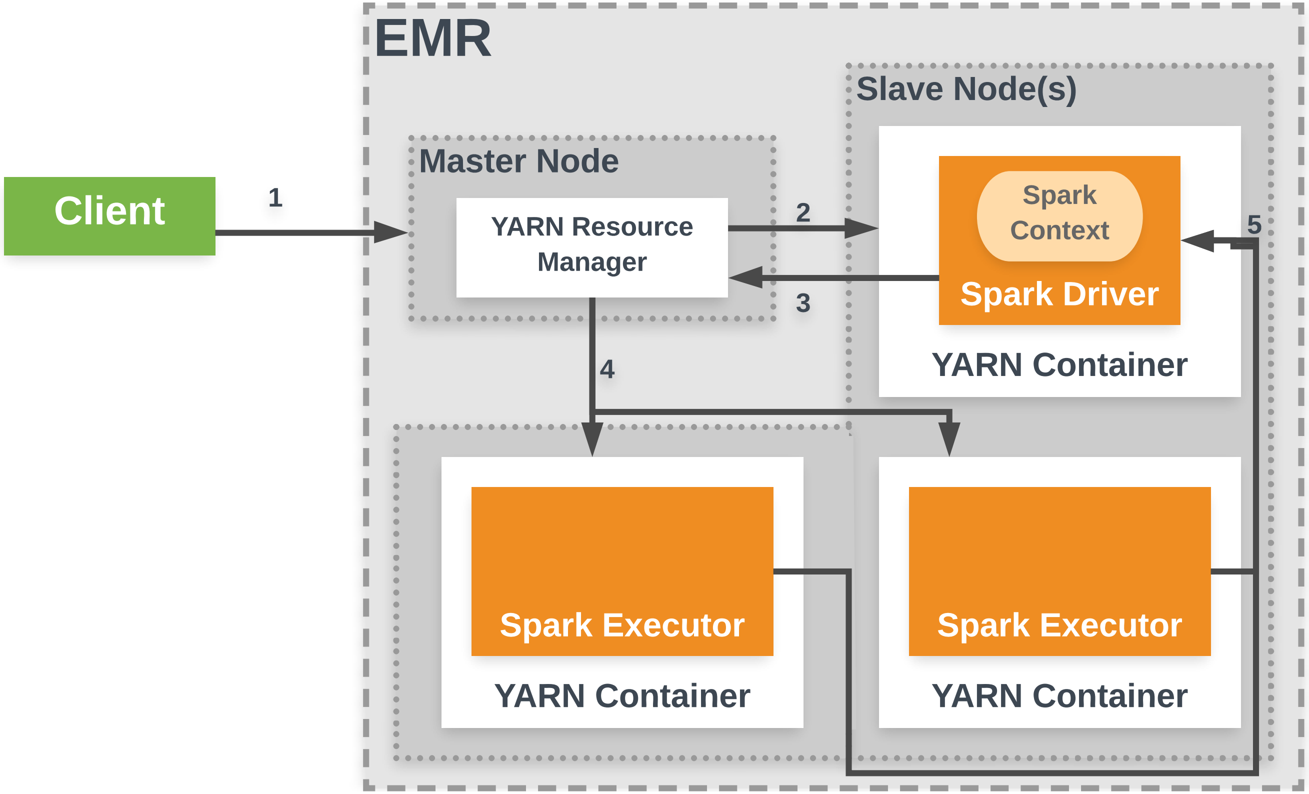


Client
1
EMR
Master Node
YARN Resource Manager
2
3
Slave Node(s)
Spark Context
Spark Driver
YARN Container
YARN Container
YARN Container

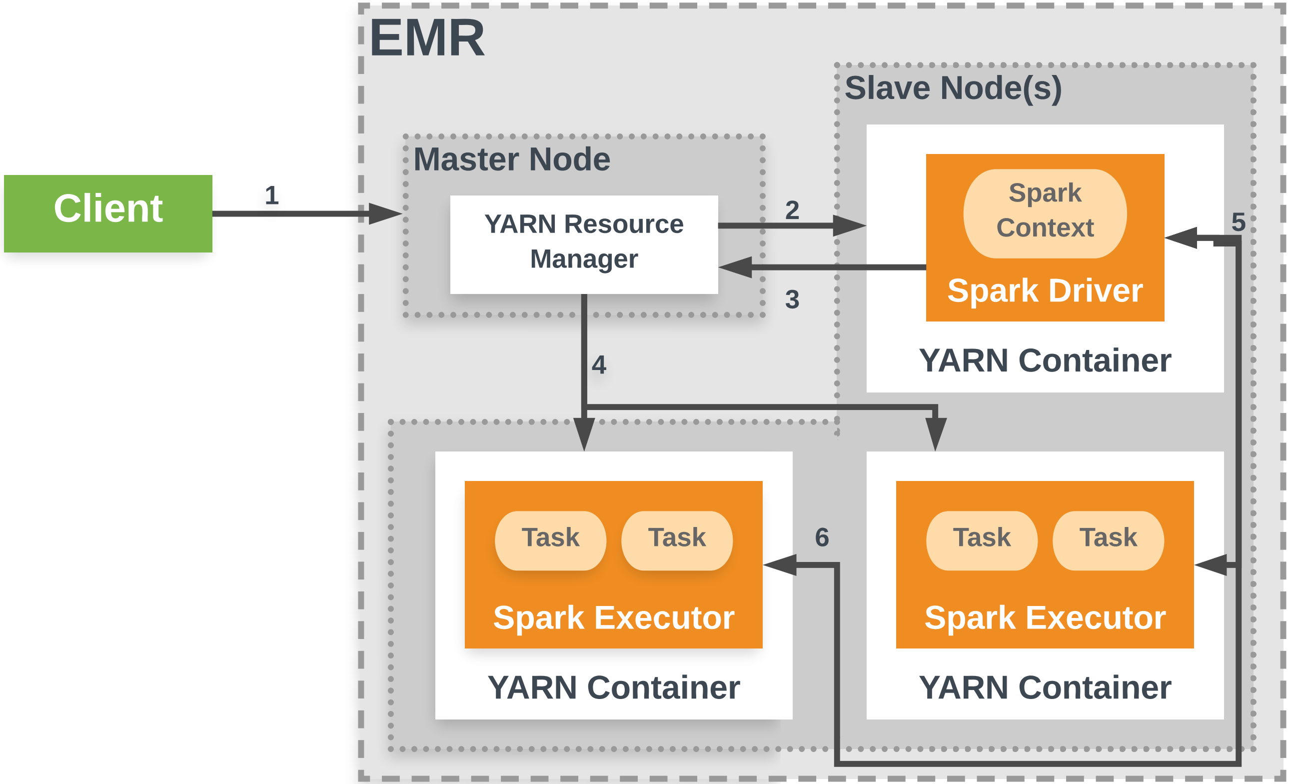
Spark on EMR


Client
1
EMR
Master Node
YARN Resource Manager
2
3
Slave Node(s)
Spark Context
Spark Driver
YARN Container
4
Spark Executor
YARN Container
Spark Executor
YARN Container

Spark on EMR


EMR
Slave Node(s)
Master Node
Client
1
YARN Resource Manager
2
3
Spark Context
Spark Driver
5
YARN Container
4
Spark Executor
YARN Container
Spark Executor
YARN Container

Spark on EMR



Client
1
EMR
Master Node
YARN Resource Manager
2
3
Slave Node(s)
Spark Context
Spark Driver
YARN Container
5
4
Task
Task
Spark Executor
YARN Container
6
Task
Task
Spark Executor
YARN Container

Run Accounting job

Create accounting container

Start Luigi Central Scheduler

Run Accounting wrapper task

Create Spark cluster on EMR

Submit Luigi tasks to EMR cluster

Destroy EMR cluster

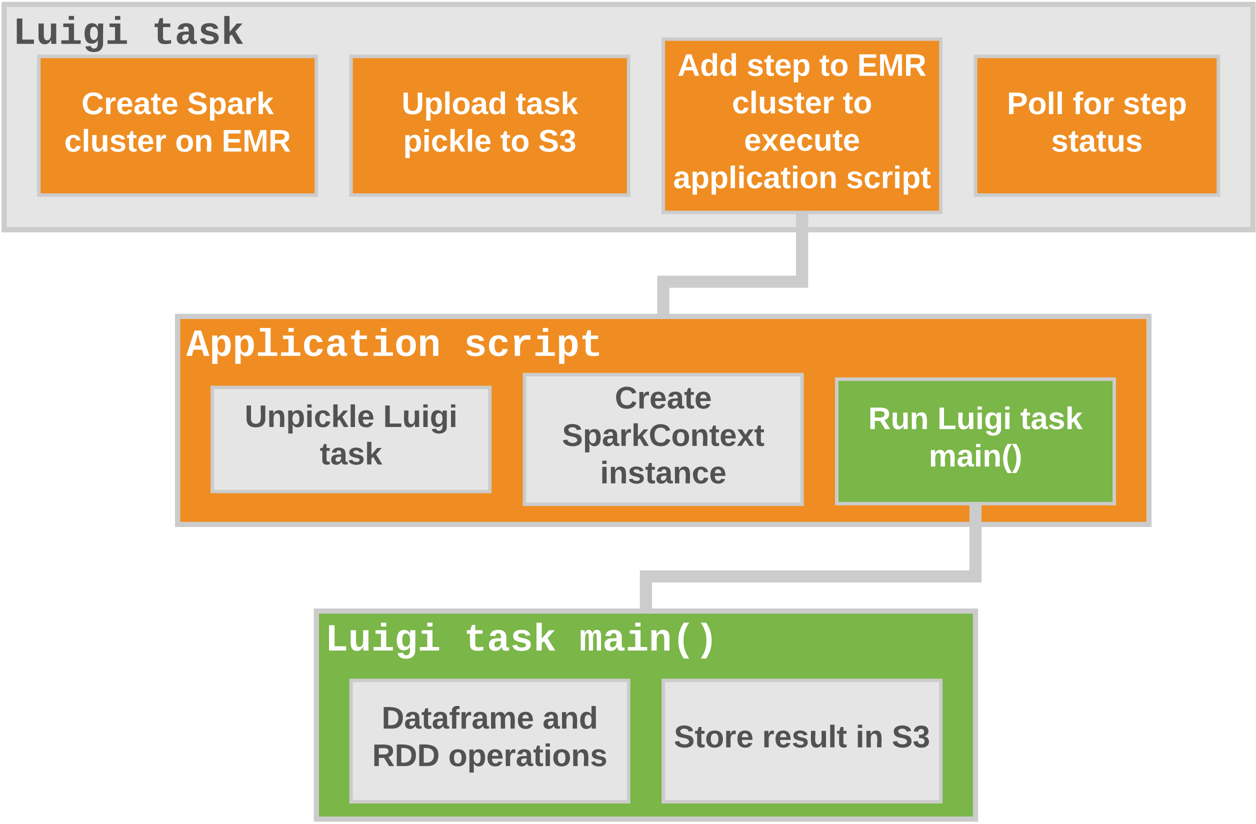
Accounting container and EMR cluster share/save files using S3

Thanks!

```python
class FetchMemberDetails(AccountingTask):

    def run(self) -> None:
        user_service_client = UserServiceClient()
        members = user_service_client.get_members()
        df_member_details = pd.DataFrame.from_records(members)

        with self.output().open('w') as file_:
            df_member_details.to_parquet(file_, engine='pyarrow', compression='SNAPPY', flavor='spark')

    def output(self) -> AccountingTarget:
        return self.get_target(path='data/raw/member-details.parquet')
```

```python
class S3DirectoryTarget(Target):

    @contextmanager
    def operate(self, mode: str) -> Generator[str, None, None]:
        if mode not in ('r', 'w'):
            raise ValueError('Unsupported open mode "{}"'.format(mode))

        output_tmp_dir = os.path.join(self.output_tmp_dir, mode)
        pathlib.Path(output_tmp_dir).parent.mkdir(parents=True, exist_ok=True)

        if mode == 'w':
            yield output_tmp_dir
            self.client.put_dir(local_dir=output_tmp_dir, destination_s3_dir=self.s3_dir, flag=self.flag)
        elif mode == 'r':
            self.client.get_dir(s3_dir=self.s3_dir, destination_local_dir=output_tmp_dir, flag=self.flag)
            yield output_tmp_dir

    def exists(self) -> bool:
        return self.client.exists(self.s3_dir)
```

Submit Spark application to EMR from Luigi



Shutdown EMR cluster

- A task won’t raise an event if one dependency has failed.
- In case of a dependency failure, we want to destroy cluster if the only tasks left depend on failing task.
- Information about pending tasks and task dependencies fetched from Luigi Central Scheduler.