Creating an Application to Submit to a Cluster

Zeppelin / REPLs vs. Spark Applications

- Zeppelin and REPLs allow for interactive manipulation, exploration, and testing
- Spark applications run as independent programs for production applications
  - Can be integrated into workflows managed by Falcon/Oozie
- The differences between them are minimal, making code reuse easy

Writing an Application to Submit to YARN

- Zeppelin and the REPLs take care of a few things automatically
  - Import the `SparkContext` and `SparkConf` libraries
  - Set up the main program
  - Create a Spark configuration object
  - Create and initialize a SparkContext instance
- For production applications, this must be coded by the developer
  - Can be accomplished in about five lines of code

Importing Libraries

- The user must code the import all the libraries used by the application
- All applications will need the `SparkContext` and `SparkConf` libraries in addition to basic libraries such as `sys` and `os`

```
import os
import sys
from pyspark import SparkContext, SparkConf
```

- To import other Spark libraries, its the same as any other application

```
from pyspark.sql import SQLContext
from pyspark.sql.types import Row, IntegerType
```

Creating a "main" Program

- The developer must set up the main program for the application

```
import os
import sys
from pyspark import SparkContext, SparkConf, SQLContext
if __name__ == "__main__":
   #Spark Programming
```

Creating a Spark Configuration

- The `SparkConf` configuration object is used by the context
  - It identifies the app name, resource manager, resources to request, etc.
- The developer must add the creation of the configuration to the application
- `SparkConf` supports pipelining as well as "setting" configuration properties

```
conf = SparkConf().setAppName("appName").setMaster("yarnMode")
conf.set('spark.executor.instances', '5')
conf.set('configuration', 'value')
```

Creating the `SparkContext`

- The `SparkContext` is used for the application to communicate to the cluster, request resources, and schedule tasks to be run
- The developer creates the context using the configuration object

```
sc = SparkContext(conf=conf)
```

- `SparkContext` has configurations that can be set after its been created

```
sc.setLogLevel("ERROR")
```

- Always stop the context at the end of the application
  - Ensures resources are properly released

```
sc.stop()
```

A Complete Application (Python)

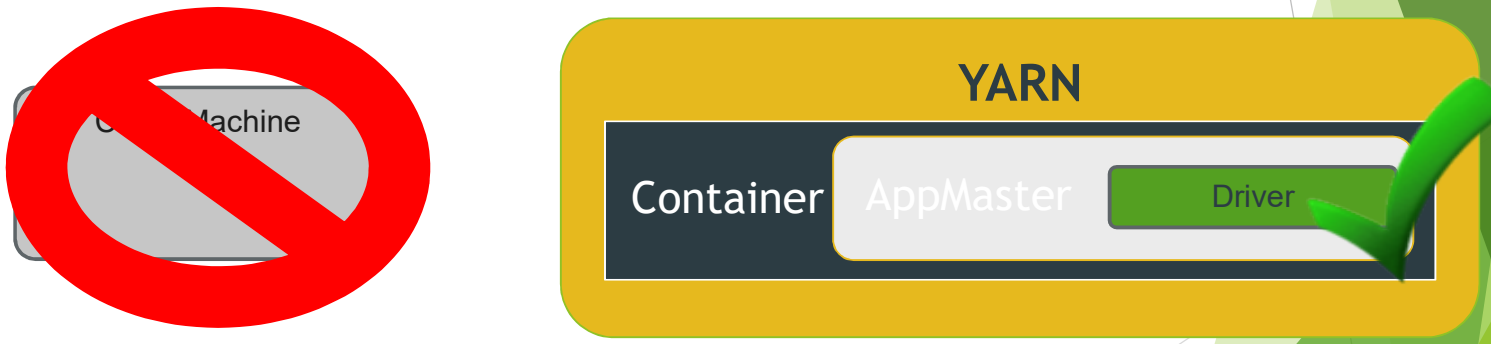
```
import os
import sys
from pyspark import SparkContext, SparkConf

if __name__ == "__main__":

    conf = SparkConf().setAppName("appName").setMaster("yarnMode")

    sc = SparkContext(conf=conf)

    sc.textFile("dataFile.txt”)

    ## Spark Programming

    sc.stop()
```

YARN Client vs. YARN Cluster

Spark Deployment Modes

yarn-client

yarn-cluster



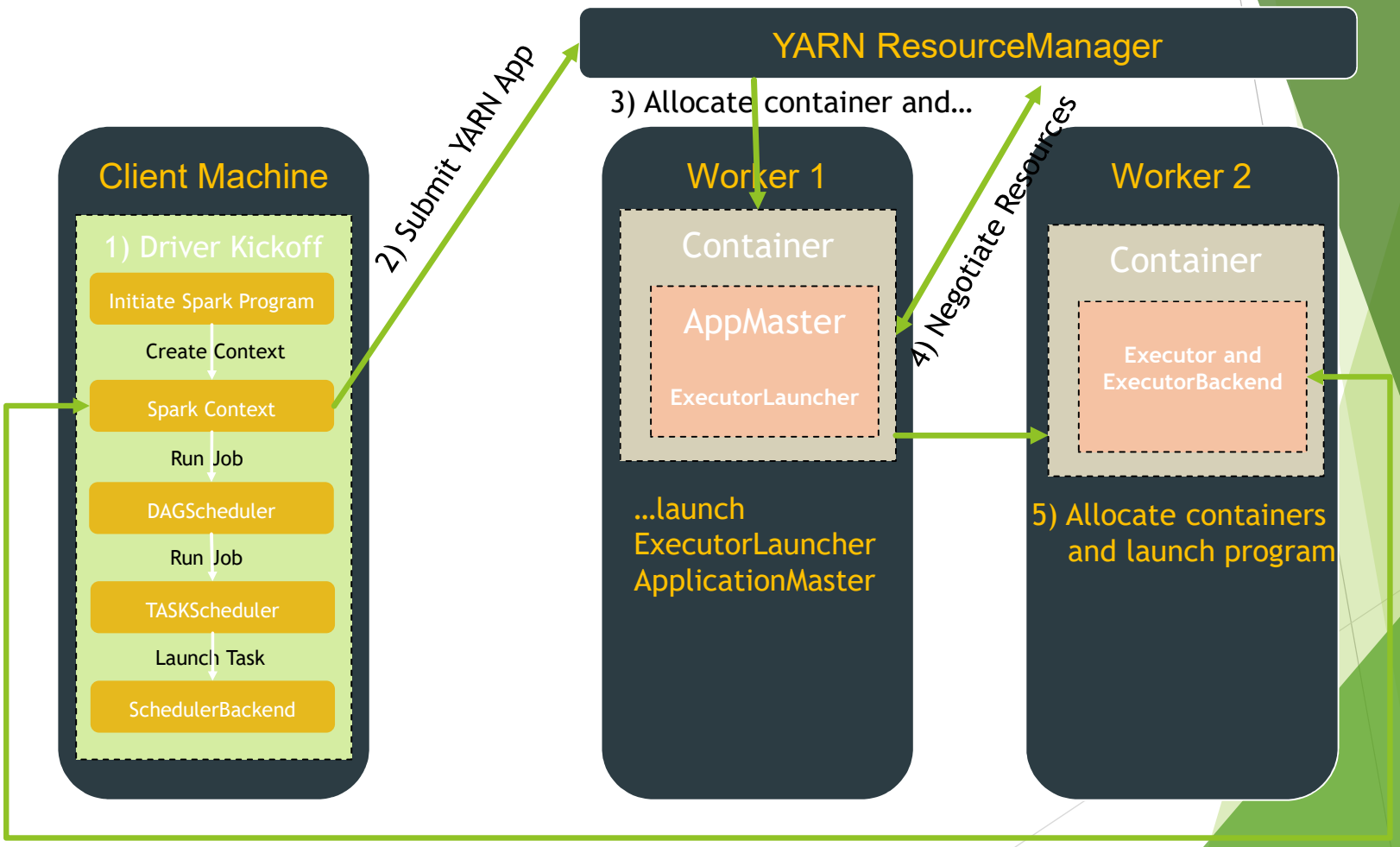
YARN Application Submission

- Spark YARN mode options:
  - `yarn-client`
  - `yarn-cluster`

- `yarn-client`
  - Developing applications
  - Testing of applications
  - REPLs and Zeppelin

- `yarn-cluster`
  - Running production applications

YARN Client Submission Process



YARN Cluster Submission Process