# Submitting Spark jobs

## Spark-submit command

To run a Spark script using this option we need to follow those steps:

* Build a Docker image containing our Spark app and script
* Push our Docker image to a could container registry
* Use the spark-submit command in terminal to run that image on the Kubernetes cluster.

Instead of pushing Docker image to a container registry we can use an image saved on each Kubernetes’ node.

**Cons:**

* Resource cleanup - We need to clean up driver and executor pods after running a job
* Observability
  + We need to get logs from each Driver and Executor Pods
  + Job status difficult to see
  + More difficult to monitor with Prometheus, Grafana, Spark History Server
* Job templating and parametrization
  + It is done through CLI commands
  + More difficult to work with (especially when there is a lot of parameters and they are different for different tasks)
* Retry support
  + We can use Airflow to retry failed jobs
  + It retries the entire job
  + it can’t retry only one Executor Pod

**Pros:**

* Easier to use

## SparkSession

In the SparkSession, in the Spark script, we can define parameters regarding Kubernetes cluster on which we want to run a Spark code.

It is similar to the spark-submit command. We provide the same parameters in the SparkSession as in the spark-submit command.

## SparkApplication resource

Here are the steps we need to follow to submit a Spark job using this method:

* Prepare a Spark script
* Build a Docker image containing our Spark script (we can use the official Spark image as the base image)
* Push that Docker image to the container registry
* Create a SparkApplication YAML manifest
* Deploy that manifest (run the Spark script) using kubectl or Airflow

In the SparkApplication YAML manifest we are specifying which Docker image we want to use and which Spark script we want to run from that image.

**Pros:**

* Observality
  + All the logs well structured in SparkAplication resource (no need to check multiple Pods)
  + Job status easy to see in SparkApplication
  + Easy integration with Prometheus, Grafana, Spark History Server
* Resource cleanup
  + Automatically cleans up driver and executor pods after executing a job
  + We only need to clean sparkApplication resource
* Job templating and parametrization
  + Specifying job parameters using YAML file
  + Easier to work with compared to CLI commands
* Retry support
  + It can retry a single Executor Pod.

**Cons:**

* Difficult to set up.

**Prerequisites:**

In order to use the SparkApplication YAML manifest we need to:

* Install Spark Operator using Helm
* Create a Service Account with proper permissions.

That Service Account will be used by the Spark Driver Pod to create Executors Pods.

Spark Operator will have automatically assigned a Service Account with proper permissions when using Helm.

**How this works in more detail:**

1. We prepare a SparkApplication YAML manifest
2. We deploy it, for example using the ‘kubectl apply’ command or using Airflow operator. That creates a SparkApplication resource.
3. Spark Operator notices that we created a SparkApplication resource and starts processing it.
4. Spark Operator creates a Spark Driver Pod.
5. Spark Driver Pod creates Spark Executor Pods.
6. Spark Pods executes Spark scripts.
7. Once the scripts are finished, Spark Pods are terminated.