Spark Job Submission Methods

The image outlines **four methods** to submit a Spark job on Kubernetes (K8s), summarized as follows:

✅ Method 1: CLI-based (spark-submit)

- **Tool**: `spark-submit`
- **How it works**:
  - User generates and executes the command using the CLI.
  - `spark-submit` interacts with the Kubernetes API to create a **driver pod**.
  - The driver pod then requests creation of **executor pods**.
- **Typical Use Case**: Manual job submission, testing, ad-hoc analysis.

✅ Method 2: Airflow DAG with SparkSubmitOperator

- **Tool**: Apache Airflow's `SparkSubmitOperator`
- **How it works**:
  - Airflow DAG task uses the operator to trigger a `spark-submit` job.
  - Follows the same path as Method 1 (driver → executors).
- **Typical Use Case**: Automated and scheduled Spark jobs in production pipelines.

✅ Method 3: Direct Kubernetes API or kubectl

- **Tool**: `kubectl` or Kubernetes client SDK (Go, Java, Python)
- **How it works**:
  - A driver pod is manually created using Kubernetes tools or client APIs.
  - The driver pod then requests executor pods.
- **Typical Use Case**: Custom integrations or direct Kubernetes-based workflows.

✅ Method 4: Spark Operator with CRD (Recommended for Production)

- **Tool**: Spark Operator + CustomResourceDefinition (CRD) `SparkApplication`
- **How it works**:
  - User creates a `SparkApplication` CRD object using `kubectl` or client SDK.
  - The Spark Operator watches the CRD and creates the driver and executor pods.
- **Typical Use Case**: Declarative Spark job management, GitOps, production deployments.

🔁 Common Flow in All Methods

- Driver pod is created first.
- Driver communicates with K8s API server to request executor pods.
- Executors are created and managed by K8s scheduler.

Let me know if you'd like a tabular comparison (e.g., pros/cons, use cases, automation level).
”