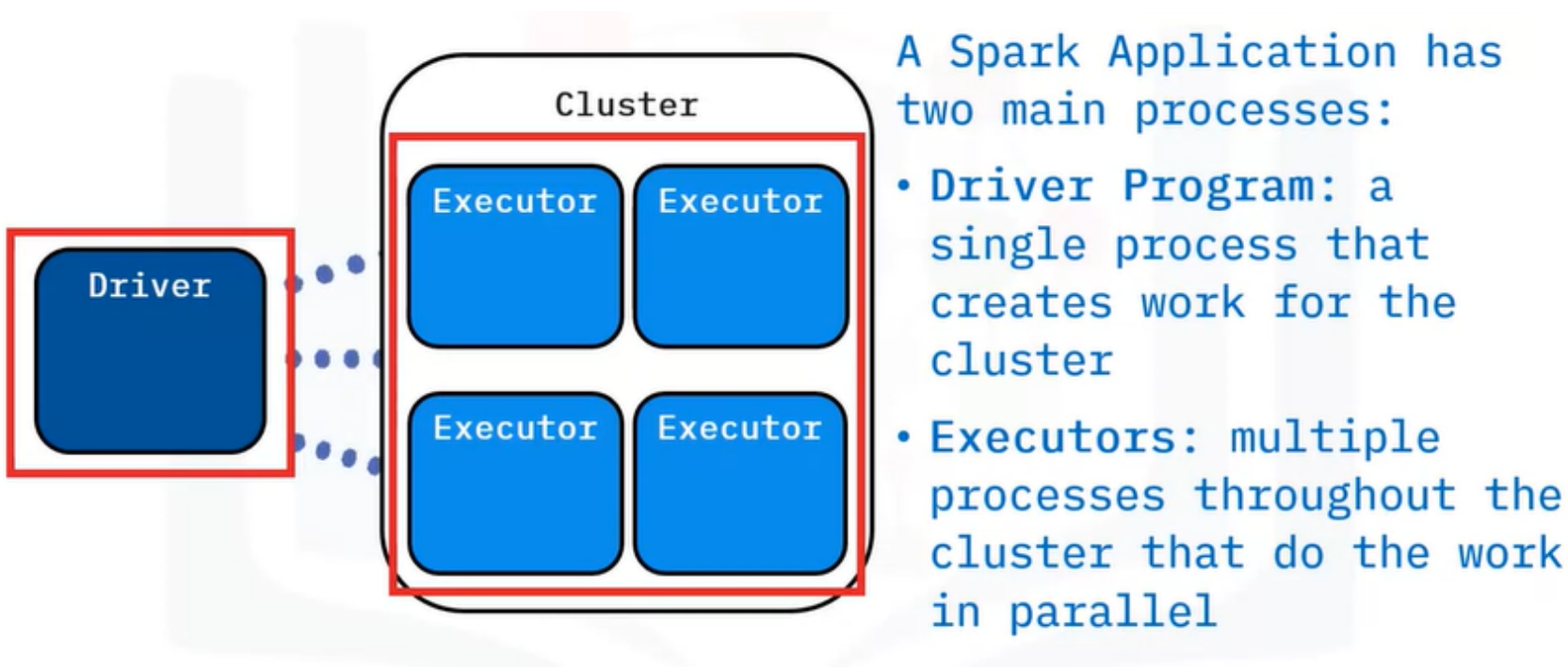
Week 5 - Development and Runtime Environment options

Spark Architecture

Apache Spark Architecture

Spark Application processes



A Spark application has two main processes.

`The driver program` - runs as one process per application. The driver process can be run on a cluster node or another machine as a client to the cluster. The driver runs the application's user code, creates work and sends it to the cluster.

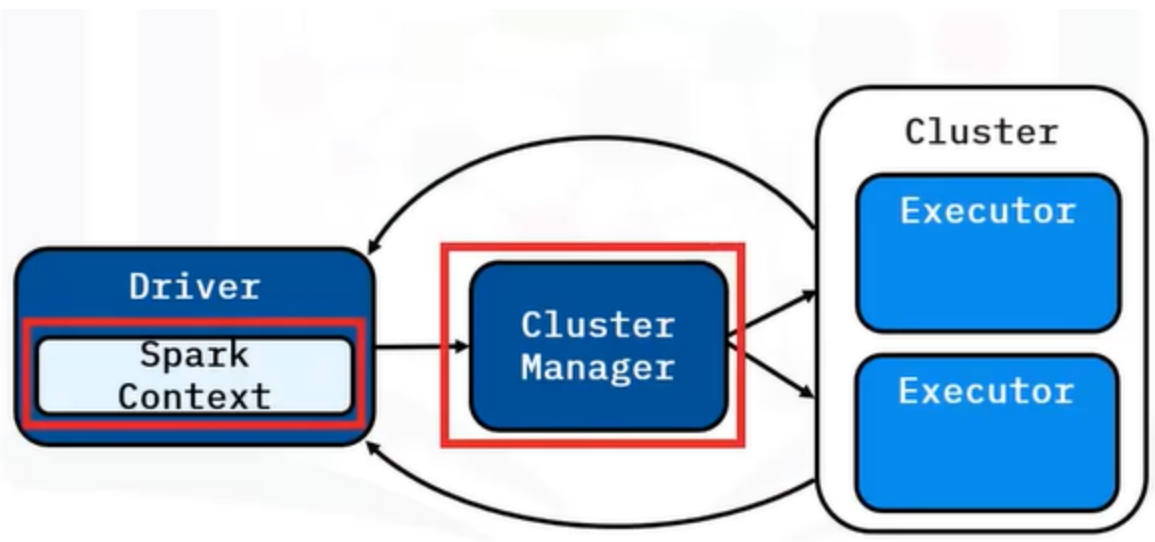
`An executor` - is a process running multiple threads to perform work concurrently for the cluster. Executors work independently. There can be many throughout a cluster and one or more per node, depending on configuration.

Spark Context

- Communicates with the Cluster Manager
- Is defined in the Driver, with one Spark Context per Spark Application

The Spark Context starts when the application launches and must be created in the driver before DataFrames or RDDs.

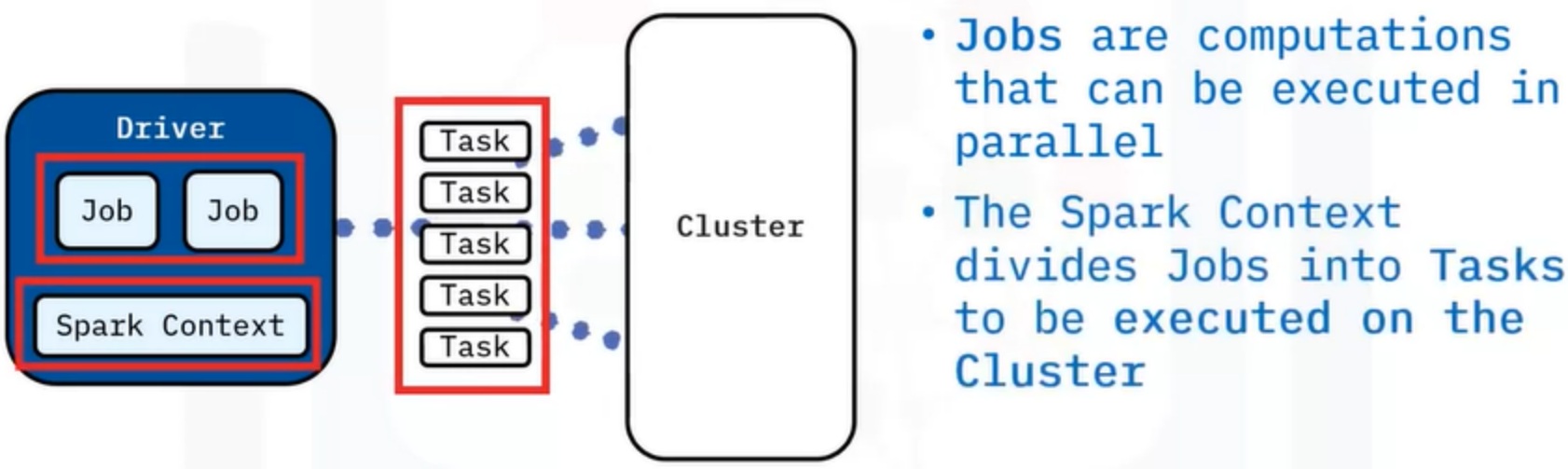
Any DataFrames or RDDs created under the context are tied to it and the context must remain active for the life of them.



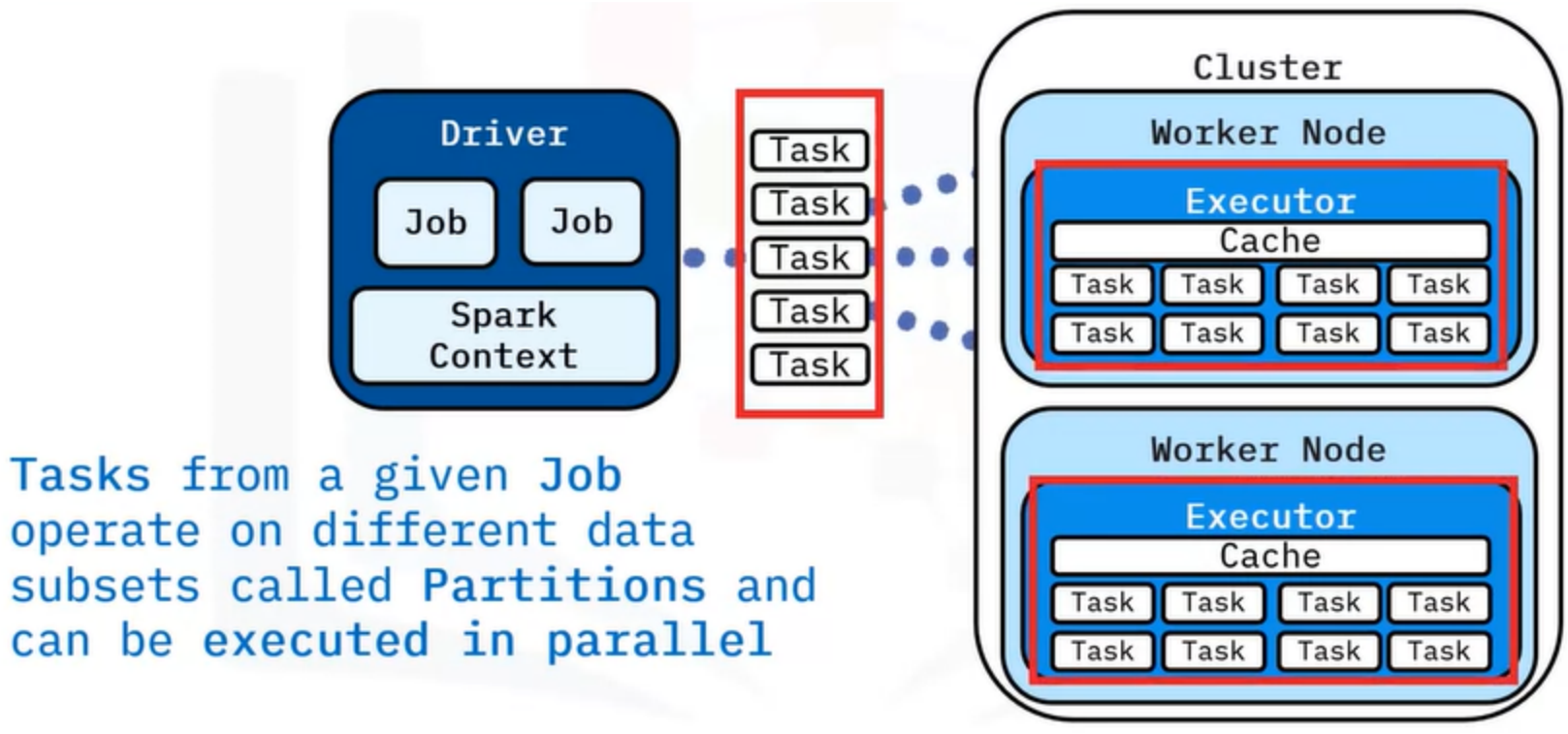
Spark Job

The driver program creates work from the user code called “Jobs” (or computations that can be performed in parallel).

The Spark Context in the driver divides the jobs into tasks to be executed on the cluster.



Spark Tasks

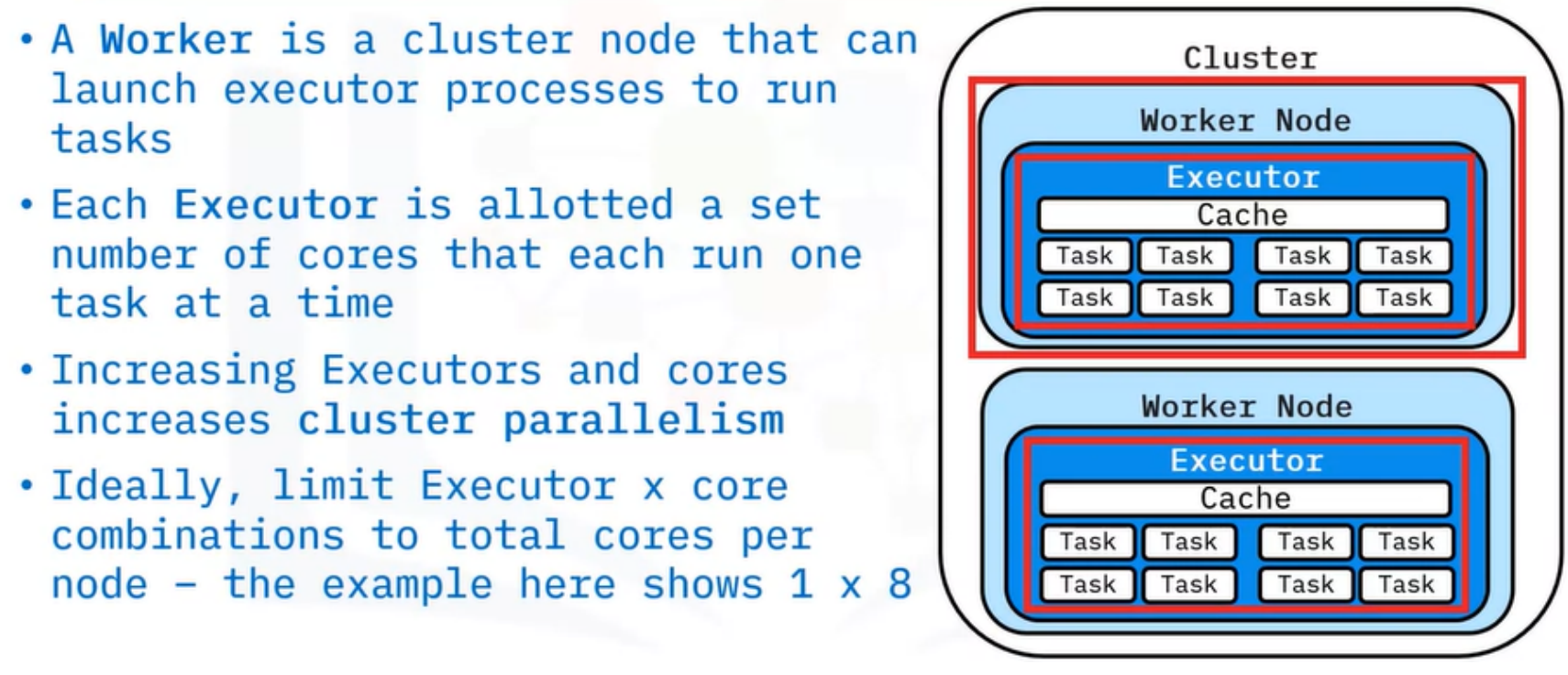


Tasks from a given job operate on different data subsets, called **Partitions**.

This means tasks can run in parallel in the Executors.

Spark Executors run tasks from a job

A Spark Worker is a cluster node that performs work.



A Spark Executor utilizes a set portion of local resources as memory and compute cores, running one task per available core.

Each executor manages its data caching as dictated by the driver.

In general, increasing executors and available cores increases the **cluster's parallelism**.

Tasks run in separate threads until all cores are used. When a task finishes, the executor puts the results in a new RDD partition or transfers them back to the driver.

Ideally, limit utilized cores to total cores available per node.

For instance, an 8-core node could have 1 executor process using 8 cores.

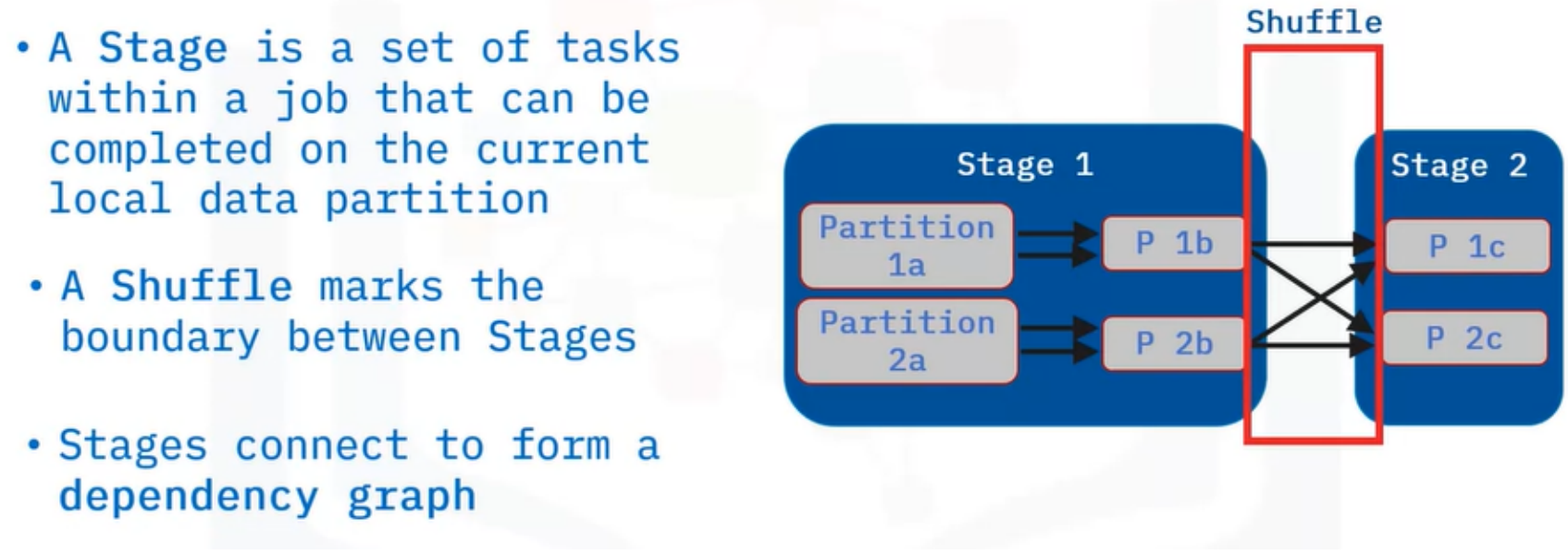
Spark Stage and Shuffle

A "stage" in a Spark job represents a set of tasks an executor can complete on the current data partition.

When a task requires other data partitions, Spark must perform a "shuffle."

A shuffle marks the boundary between stages.

Subsequent tasks in later stages must wait for that stage to be completed before beginning execution, creating a dependency from one stage to the next.

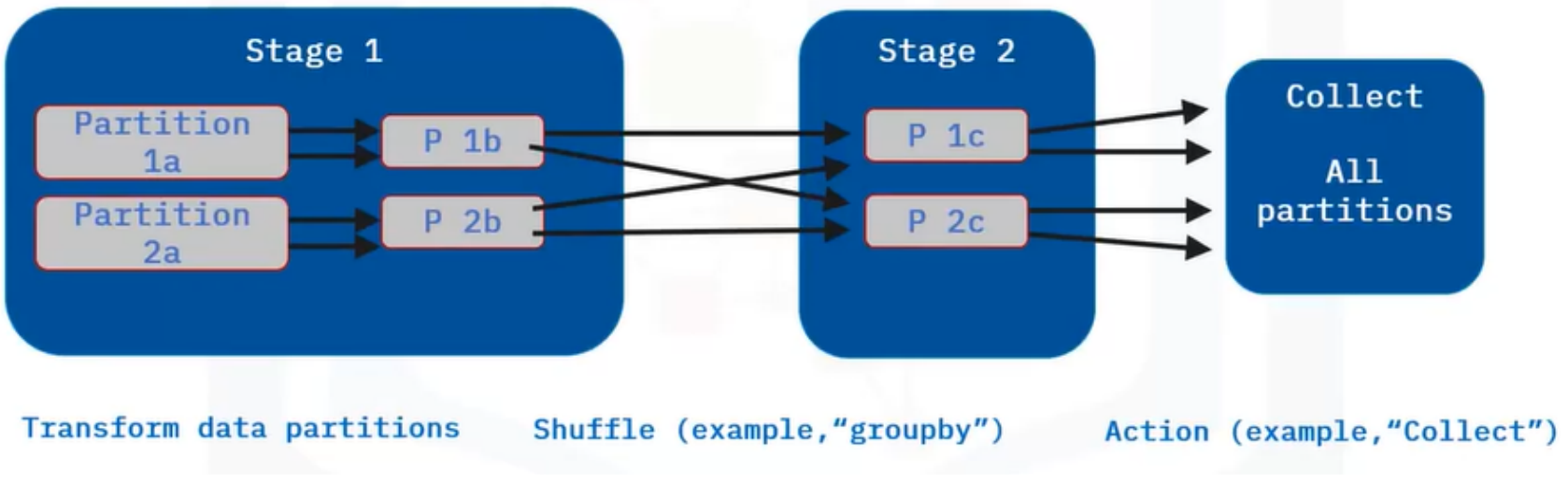


Why Shuffle data?

Shuffles are:

- `Costly` - require data serialization, disk and network I/O.
- This is because they enable tasks to "pass over" other dataset partitions outside the current partition.
  An example would be a "group by" with a given key that requires scanning each partition to find matching records.
- When Spark performs a shuffle, it redistributes the dataset across the cluster.

Shuffle example



This example shows two stages separated by a shuffle.

In Stage 1, a transformation (such as a map) is applied on dataset "a" which has 2 partitions ("1a" and "2b").

This creates data set "b".

The next operation requires a shuffle (such as a “groupby”).

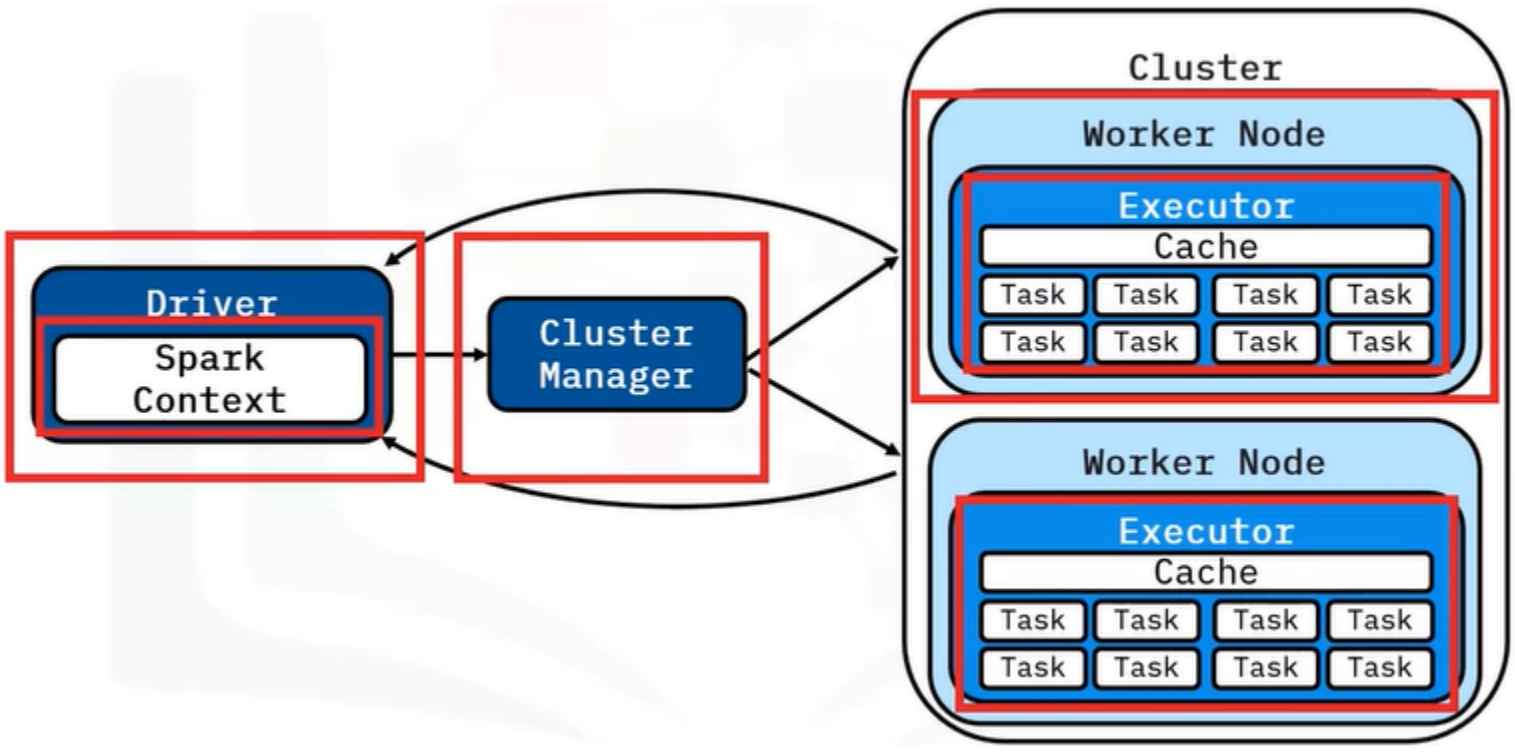
Key values could exist in any other partition, so to group keys of equal value together, tasks must scan each partition to pick out the matching records.

Transformation results are placed in Stage 2. Here results have the same number of partitions, but this depends on the operation.

Final results are sent to the driver program as an action, such as collect.

NOTE: It is not advised to perform a collection to the driver on a large data set as it could easily use up the driver process memory. If the data set is large, apply a reduction before collection.

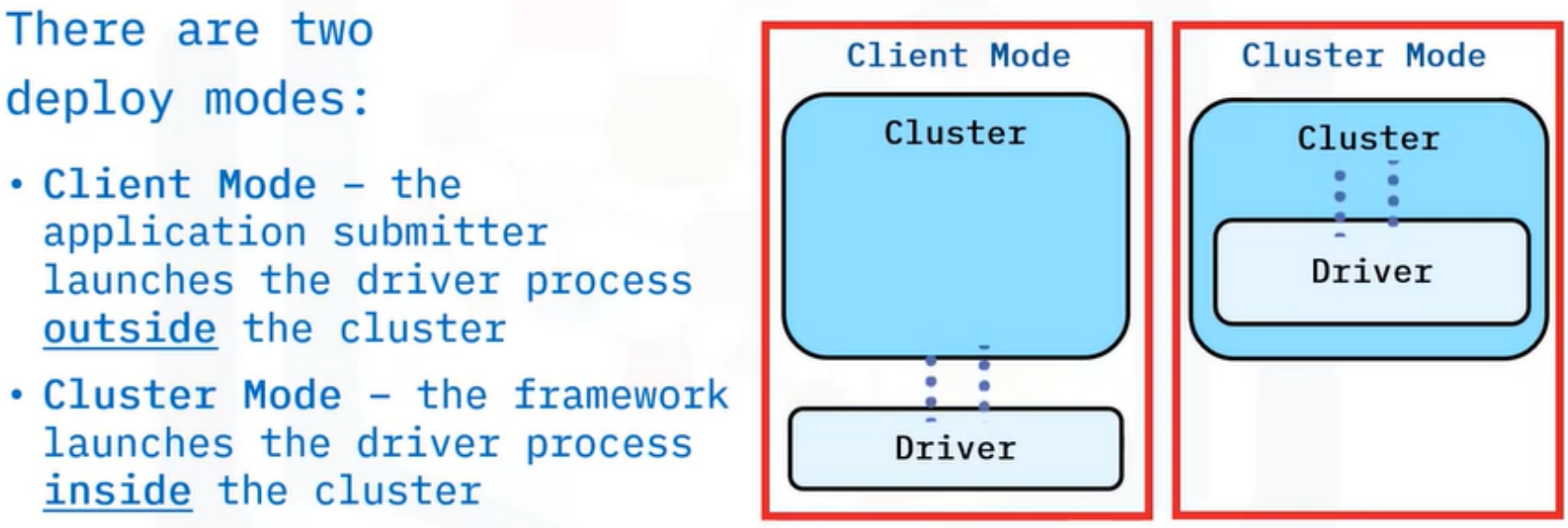
Apache Spark Architecture recap



It consists of **the driver** and **the executor processes**.

- The cluster comprises the Cluster Manager and worker nodes.
- The Spark Context schedules tasks to the cluster
- The Cluster Manager manages the cluster’s resources.

Drive deploy modes



The driver program can be run in either **client mode** or **cluster mode**.

- In client mode the application submitter (such as a user machine terminal) launches the driver **outside** the cluster.
- In cluster mode, the driver program is sent to and run on an available Worker node **inside** the cluster.

The driver must be able to communicate with the cluster while it is running, whether it is in client or cluster mode.

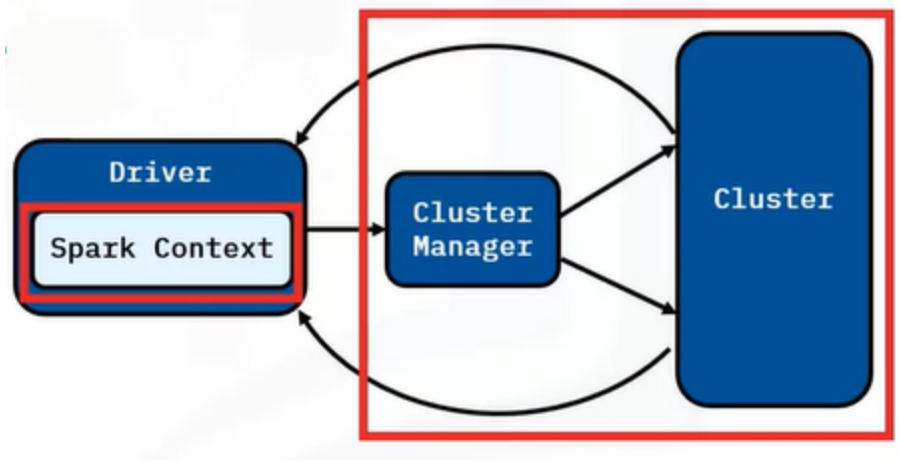
Conclusion

- Spark Architecture has driver and executor processes, coordinated by the Spark Context in the driver.
- The driver creates jobs and splits them into tasks which can be run in parallel in the executors.
- Stages are a set of tasks that are separated by a data shuffle.
- Shuffles are costly, as they require data serialization, disk and network I/O.
- The Driver can be run in either or both Client or Cluster mode.
  - The Client Mode connects the driver outside the cluster
  - The Cluster Mode runs the driver in the cluster.

Overview of Apache Spark Cluster Modes

Spark Cluster Manager

- Communicates with a cluster to acquire resources for an application to run.
- Run as a service outside the application and abstracts the cluster type



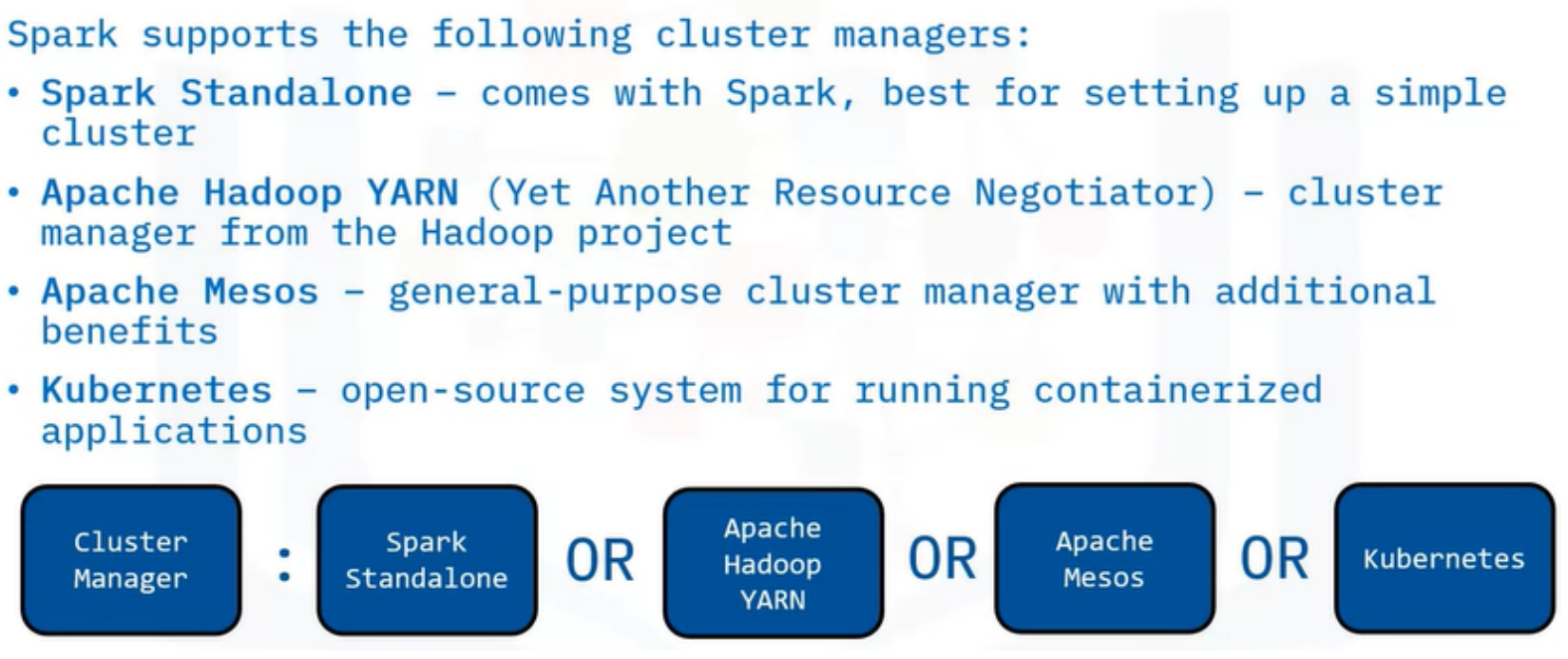
While an application is running, the Spark Context creates tasks and communicates to the cluster manager what resources are needed.

Then the cluster manager reserves executor cores and memory resources. Once the resources are reserved, tasks can be transferred to the executor processes to run.

Type of Cluster Manager

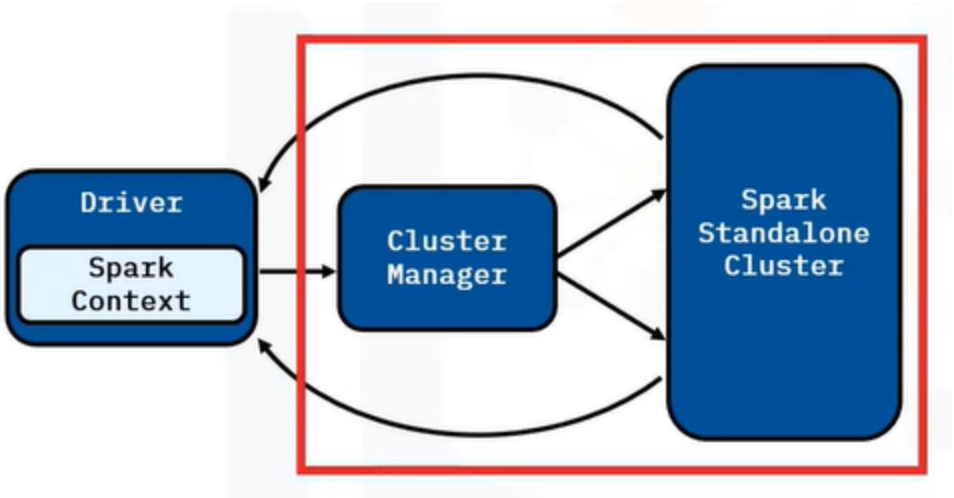
Spark has built-in support for several cluster managers.

- `Spark Standalone` - is included with the Spark installation. It's best for setting up a simple cluster.
- `Apache Hadoop YARN (Yet Another Resource Negotiator)` - is a cluster manager from the Hadoop project.
- `Apache Mesos` - is a general-purpose cluster manager that Spark can run with additional benefits.
- `Kubernetes` - is an open-source system for running containerized applications.



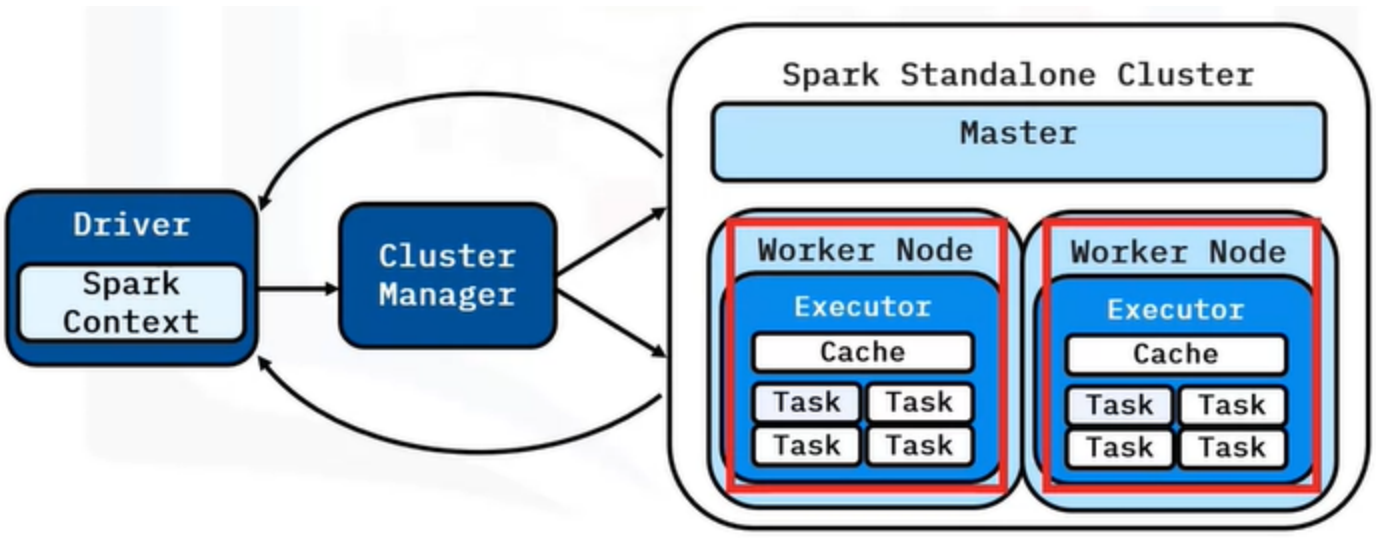
Spark Standalone

Why use Spark Standalone?



- comes packaged with the Spark installation → no additional dependencies required to configure and deploy.
- Spark Standalone is specifically designed to run Spark and is often the fastest way to get a cluster up and running applications.

Spark Standalone components



2 main components: **Workers** and **the Master**.

- `The Worker`
  - Run an Executor process to receive tasks
  - The workers run on cluster nodes. They start an executor process with one or more reserved cores.
- `The Master`
  - Connects and adds workers to the cluster
  - There must be one master available which can run on any cluster node. It connects workers to the cluster and keeps track of them with heartbeat polling.

💡 However, if the master is together with a worker, do not reserve all the node’s cores and memory for the worker.

How to setup Spark Standalone

1. Start the Master to output the Master URL

```
$ ./sbin/start-master.sh
```

2. Start Worker(s) with the Master URL

```
$ ./sbin/start-slave.sh spark://<master-spark-URL>:7077
```

3. Launch Spark application on cluster by specifying the Master URL

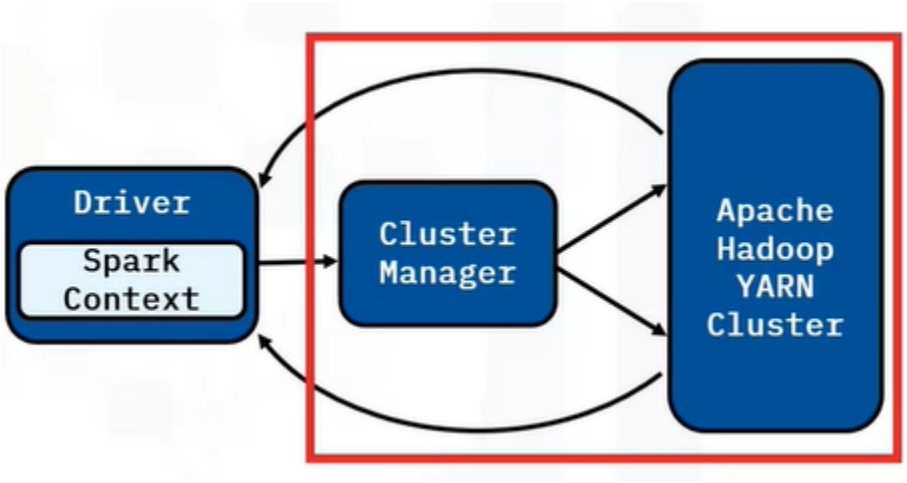
```
$ ./bin/spark-submit \
    --master spark://<spark-master-URL>:7077 \
    <additional configuration>
```

To manually set up a Spark Standalone cluster:

- Start the Master.
- The Master is assigned a URL with a hostname and port number.
- After the master is up, use the Master URL to start workers on any node using bi-directional communication with the master.
- Once the master and the workers are running, launch a Spark application on the cluster by specifying the master URL as an argument to connect them.

Apache Hadoop YARN

Why use Apache Hadoop YARN?



When choosing YARN, consider that it:

- A general-purpose cluster manager.
- It’s popular in the big data ecosystem and supports many other frameworks besides Spark.
- YARN clusters have their own dependencies, set-up and configuration requirements so deploying them is more complex than Spark Standalone.

How to run Spark on existing YARN cluster?

1. Specify the Master option `--master YARN` with ‘spark-submit’

```
$ ./bin/spark-submit \
    --master YARN \
    <additional configuration>
```

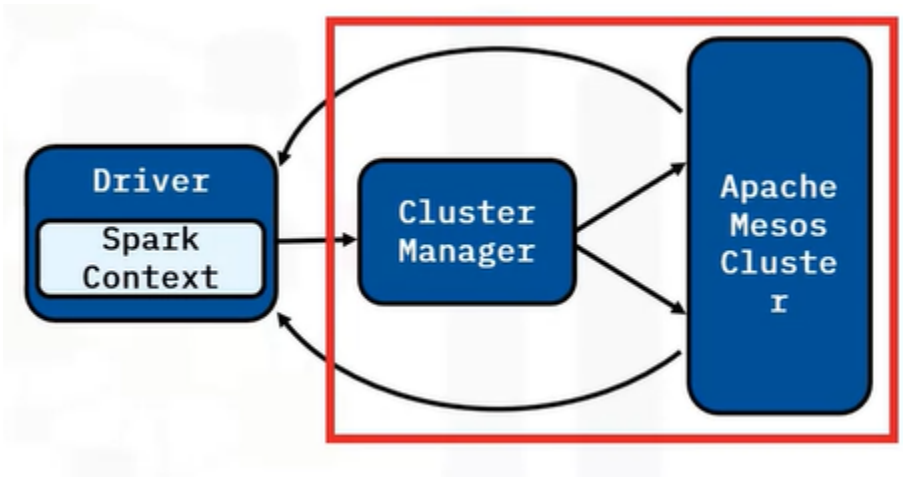
2. Spark will automatically connect with YARN using Hadoop configuration

To run Spark on an existing YARN cluster:

- Use the ‘--master’ option with the keyword YARN.
- When Spark sees that the cluster manager type is YARN, it will look for the standard Hadoop configuration files to decide how to connect with the YARN cluster.
- No other configuration from Spark is needed.

Apache Mesos

Why use Apache Mesos?



Apache Mesos is another general-purpose cluster manager supported by Spark.

Apache Mesos Cluster Managers can run Spark with other benefits, such as making partitioning:

- Provide **dynamic** partitioning between Spark and other big data frameworks
- **Scalable** partitioning between multiple Spark instances.

However, running Spark on Apache Mesos might require some additional setup depending on your configuration requirements. More details are provided on the *spark.apache.org* website.

Kubernetes

Why use Kubernetes?

A Kubernetes cluster runs containerized applications.

- Makes Spark applications more portable
- Helps automate deployment,
- Simplify dependency management
- Scale the cluster as needed.

How to run Spark on Kubernetes?

To launch a Spark Application on Kubernetes use:

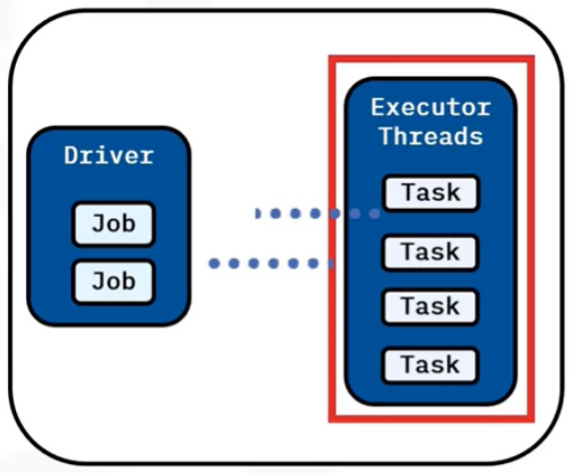
```
$ ./bin/spark-submit \
    --master k8s://https://<k8s-apiserver-host>:<k8s-apiserver-port> \
    <additional configuration>
```

Spark uses a built-in native Kubernetes scheduler.

To run Spark on Kubernetes, use the example call shown.

Local Mode

Why use Local Mode?



Local mode runs a Spark application as a single process locally on the machine.

- Executors are run as separate threads in the main process that calls 'spark-submit'.
- Local mode does not connect to any cluster or require configuration outside a basic Spark installation.
- Local mode can be run on a laptop. That’s useful for testing or debugging a Spark application, for example, testing a small data subset to verify correctness before running the application on a cluster.
- Being constrained by a single process means local mode is not designed for optimal performance.

How to setup Local Mode?

To run Spark in local mode, use the master option `--master local[#]` where `#` specifies the number of cores to use

```
# Launch Spark in local mode with 8 cores
$ ./bin/spark-submit \
    --master local[8] \
    <additional configuration>
```

Use an asterisk ‘*’ to specify using all available cores

```
# Launch Spark in local mode with all available cores
$ ./bin/spark-submit \
    --master local[*] \
    <additional configuration>
```

To run in local mode

- Use the ‘--master’ option with the keyword ‘local’ followed by a number to indicate how many cores the Spark application can use for the executor.
- To use all available cores, replace the number with an asterisk wildcard.

Keep in mind not all configuration for a cluster will be valid in local mode.

Conclusion

- Cluster managers acquire resources and run as an abstracted service outside the application.
- Spark can run on Spark Standalone, Apache Hadoop YARN, Apache Mesos or Kubernetes cluster managers, with specific set-up requirements.
- Choosing a cluster manager depends on your data ecosystem and factors such as ease of configuration, portability, deployment or data partitioning needs.
- Spark can run in local mode, useful for testing or debugging an application.

How to Run an Apache Spark Application

What is ‘spark-submit’?

Spark comes with a unified interface for submitting applications

- it is the ‘spark-submit’ script
- found in the ‘bin/’ directory.
- ‘spark-submit’ can be used for all supported cluster types and accepts many configuration options for the application or cluster.
- “Unified interface” means you can switch from running Spark in local mode to cluster by changing a single argument.
- ‘spark-submit’ works the same way, no matter the application language.

```
$ ./bin/spark-submit <config and options> <application files>
```

💡 For example, a cluster can run Python and Java applications simultaneously by passing in the required files.

Using the ‘spark-submit’ script

The ‘spark-submit’ script:

1. Parses any command line arguments or options.
2. Reads any additional configuration specified in the ‘spark-defaults’ folder.
3. The ‘--master’ argument tells the application how to connect to the cluster or run locally.
4. Any application files (including JARs or Python files) must be specified so they can start the driver program and distribute files to run in the cluster.

Common ‘spark-submit’ options

Option/Setting	Form	Mandatory ?
Tell Spark what cluster manager to connect with	‘--master’	Yes
Specify the program entry point if using Java or Scala application	`--class <full-class-name>`	Yes
Set how the driver is deployed (Client or Cluster). Default is Client mode	‘deploy-mode’	No
Set CPU core and memory usage in executors	`--executor-cores` and `--executor-memory`	No
See available options by cluster manager	`./bin/spark-submit --help`	N/A

Some ‘spark-submit’ options are mandatory, such as specifying the master option to tell Spark which cluster manager to connect to.

If the application is written in Java or Scala and packaged in a JAR, you must specify the full class name of the program entry point.

Other options include driver deploy mode (run as a client or in the cluster)

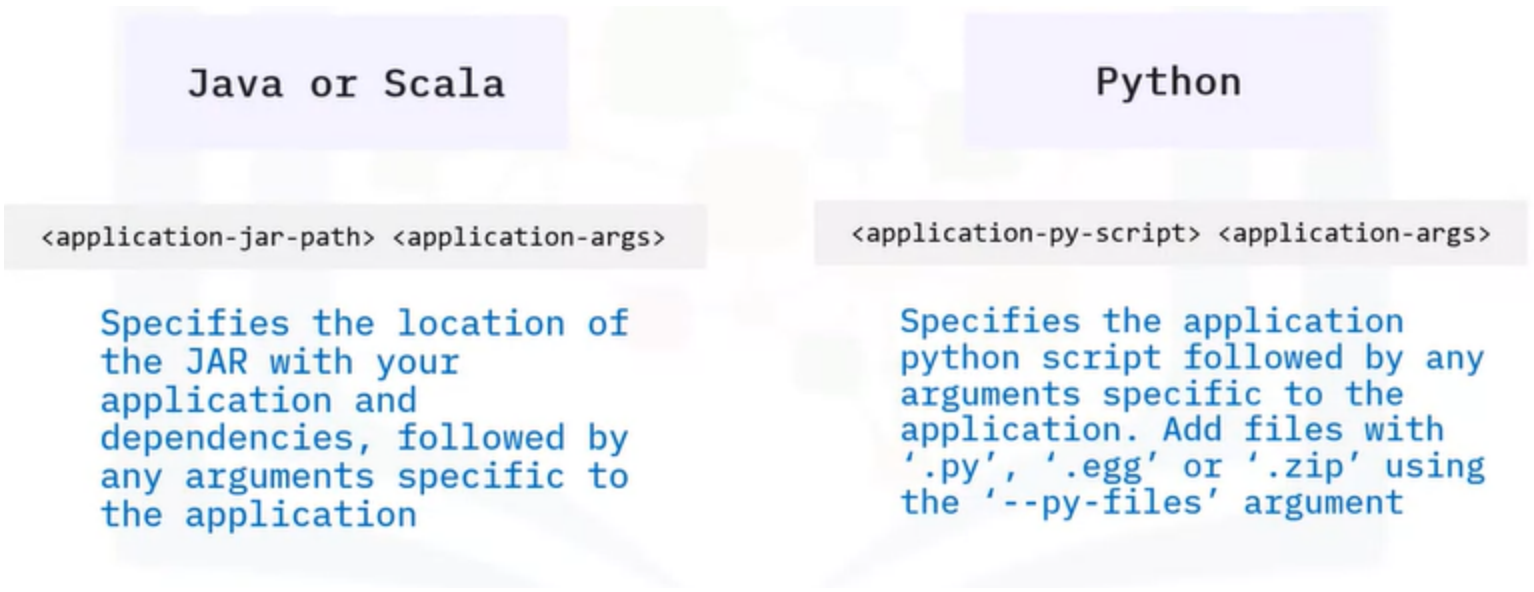
Eexecutor resource settings (such as reserving memory or cores).

Driver deploy mode defaults to client mode.

Some options relate to specific cluster managers.

‘spark-submit’ application files

Additional Spark configuration can be specified on the command line using the ‘--conf’ argument followed by key=value.



For Java or Scala and Python, final arguments will be the path to the application JAR or Python script. Next are optional application-specific arguments.

You can add Python files with ‘.py,’ ‘.egg’, or ‘.zip’ using the ‘--py-files’ argument. These files will be distributed to the cluster and available in the Python environment.

‘spark-submit’ examples

1. Launch Scala SparkPi using a jar, with master YARN. Estimate Pi with 1000 samples

```
# Launching Scala SparkPi to a YARN cluster

./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master YARN \
  /path/to/examples.jar \
  1000
```

2. Launch Python SparkPi to a Spark Standalone cluster. Estimate Pi with 1000 samples

```
# Launching Python SparkPi to a standalone cluster with master at 207.184.161.138

./bin/spark-submit \
  --master spark://207.184.161.138:7077 \
  examples/src/main/python/pi.py \
  1000
```

The first launches SparkPi, written in Scala and packaged in a JAR. YARN is the master. The application uses an additional argument (how many samples to take in the cluster to compute Pi).

The second example launches a SparkPi Python version. The master is a Spark Standalone cluster and we simply pass the path to the Python script.

Application dependencies

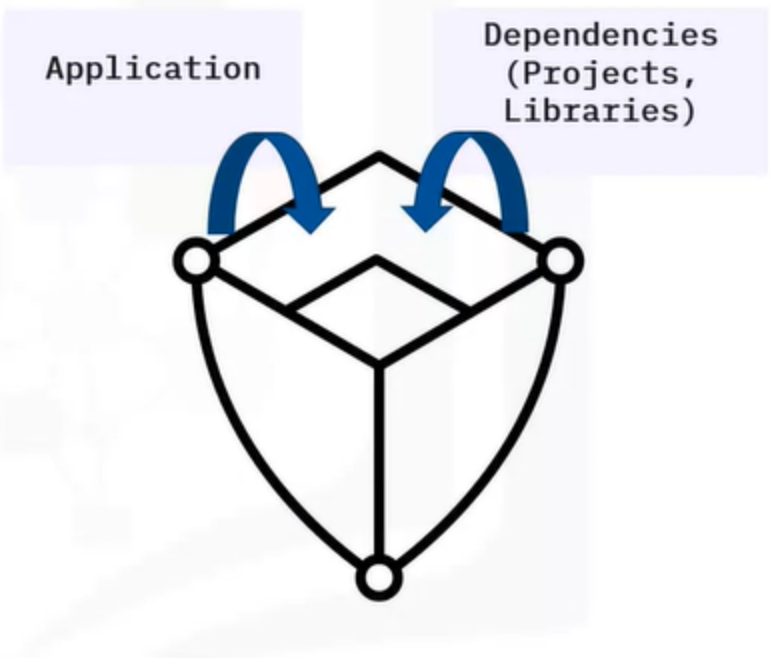
To manage Spark dependencies, the application project or library must be accessible for the driver and cluster executor processes.

Java or Scala-based application

It’s best to create an uber-JAR. This is a single JAR file with all dependencies (including transitive ones) so the application is portable throughout the cluster.

To manage Spark dependencies:

- Bundle projects or libraries with application so they are accessible to driver and executor processes
- For Java or Scala-based programs, create an uber-JAR with application and dependencies together so it is easy to distribute to the cluster



Python applications

Ensure:

(1) the cluster nodes access required dependencies with the same version AND

(2) the same Python version is used.

Solutions for Python dependencies may involve creating virtual environments so that applications can run in separate, isolated environments.

Python dependencies can be included in the spark-submit script using the ‘--py-files’ argument: this will accept ‘.py’, ‘.zip’ or ‘.egg’ files.

To manage Python application dependencies, ensure:

- Cluster nodes can access the required dependencies with same version
- Same Python version is used
- You use the `--py-files` argument so that `.py`, `.zip` or `.egg` files can be distributed to the cluster

```
# Launching a PySpark application with dependency on “my_python_package”

$ ./bin/spark-submit <config and options> \
    --py-files my_python_package.zip \
    my_pyspark_application.py
```

Spark Shell

Spark Shell is available for Scala and Python, giving you access to Spark APIs for working with data as Spark jobs.

Spark Shell can be used in local or cluster mode, with all options available.

Spark Shell:

- Simple way to learn Spark API
- Powerful tool to analyze data interactively
- Use in local mode or with a cluster, same options as 'spark-submit'
- Can initiate in Scala ('bin/spark-shell') and Python ('bin/pyspark')

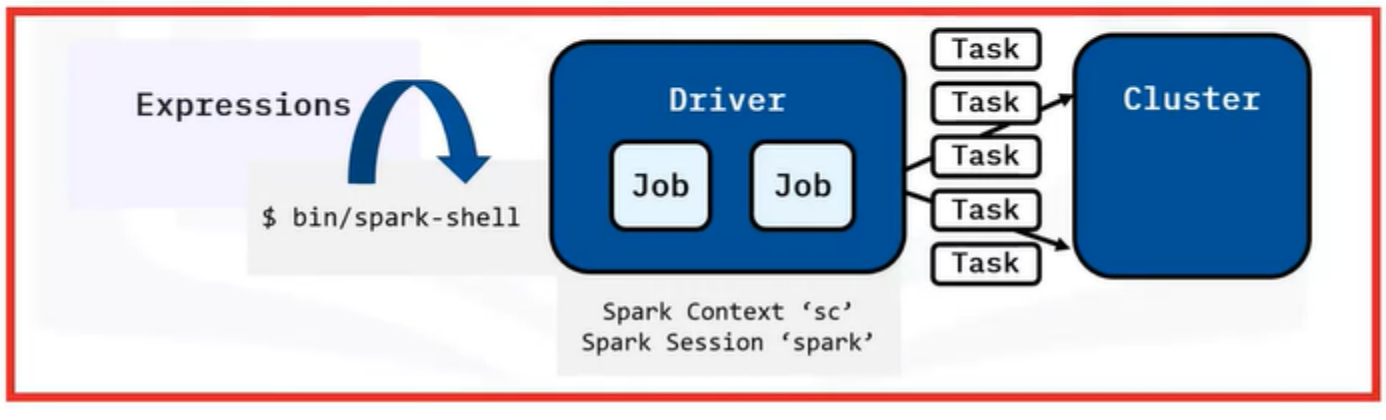
Spark Shell Environment

When Spark Shell starts, the environment automatically initializes the SparkContext and SparkSession variables.
This means you can start working with data immediately.

Expressions are entered in the shell and evaluated in the driver.
Entering an action on a shell DataFrame generates Spark jobs that are sent to the cluster to be scheduled as tasks.

For both Scala and Python shells:

- SparkContext is automatically initialized and is available as 'sc'
- SparkSession is automatically available as 'spark'
- Expressions are entered into the shell and then evaluated in the driver to become jobs that are scheduled as tasks for the cluster



This example shows the Spark Scala shell starting up in local mode.

Example: Start Scala Spark-Shell and run local mode by default

```
$ bin/spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://192.168.1.33:4040
Spark context available as 'sc' (master = local[*], app id = local-1614631042181).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_282)
Type in expressions to have them evaluated.
Type :help for more information.
scala>
```

The print-out includes:

- Spark's load log in the log4j-defaults.properties file;
- Spark's Web UI address, displaying running jobs in the shell;
- Variable names for the initialized SparkContext and SparkSession
- Version information for important libraries like JDK and Scala.

Spark Shell Information Provided

- Spark's load log in the log4j-defaults.properties files
- Spark's Web UI address to view information about jobs running in the shell
- Spark Context and Spark Session variables
- Versions of important libraries in use, for example JDK and Scala

```
$ bin/spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://192.168.1.33:4040
Spark context available as 'sc' (master = local[*], app id = local-1614631042181).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0 282)
Type in expressions to have them evaluated.
Type :help for more information.
scala>
```

Spark Shell Example - Run code

This example shows running code in the Scala shell.

In the first line, a DataFrame is created as a distributed range of integers from 0 to 9.

The second line transforms the DataFrame, adding a column calculating the modulo of 2.

Next is an action that prints out the first four records of the resulting DataFrame.

```
scala> val df = spark.range(10)
df: org.apache.spark.sql.Dataset[Long] = [id: bigint]


scala> df.withColumn("mod", expr("id % 2")).show(4)
+---+---+
| id|mod|
+---+---+
|  0|  0|
|  1|  1|
|  2|  0|
|  3|  1|
+---+---+
```

1. Launch Scala Spark Shell
2. Create distributed DataFrame with column 'id' and 10 values (0-9)
3. Add a column that evaluates an SQL expression for modulo of 2 and show first four (4) results

Conclusion

- 'spark-submit' is a unified interface to submit the Spark Application, no matter the cluster manager or application language.
- Mandatory options include telling Spark which cluster manager to connect to; other options set driver deploy mode or executor resourcing.
- To manage dependencies, application projects or libraries must be accessible for driver and executor processes, for example by creating a Java or Scala uber-JAR.
- Spark Shell simplifies working with data by automatically initializing the SparkContext and SparkSession variables and providing Spark API access.

Hands-on Lab: Submit Apache Spark Applications

Summary & Highlights

- Spark Architecture has driver and executor processes, coordinated by the Spark Context in the Driver.
- The Driver creates jobs and the Spark Context splits jobs into tasks which can be run in parallel in the executors on the cluster. Stages are a set of tasks that are separated by a data shuffle. Shuffles are costly, as they require data serialization, disk and network I/O. The driver program can be run in either client Mode (connecting the driver outside the cluster) or cluster mode (running the driver in the cluster).
- Cluster managers acquire resources and run as an abstracted service outside the application. Spark can run on Spark Standalone, Apache Hadoop YARN, Apache Mesos or Kubernetes cluster managers, with specific set-up requirements. Choosing a cluster manager depends on your data ecosystem and factors such as ease of configuration, portability, deployment, or data partitioning needs. Spark can also run using local mode, which is useful for testing or debugging an application.
- 'spark-submit' is a unified interface to submit the Spark application, no matter the cluster manager or application language. Mandatory options include telling Spark which cluster manager to connect to; other options set driver deploy mode or executor resourcing. To manage dependencies, application projects or libraries must be accessible for driver and executor processes, for example by creating a Java or Scala uber-JAR. Spark Shell simplifies working with data by automatically initializing the SparkContext and SparkSession variables and providing Spark API access.

Practice Quiz: Spark Architecture

1.Question 1

Identify the two main processes associated with a Spark application.

- [ ] The Driver Program, which performs the application's parallel processing.
- [ ] The Executors, which perform single processes throughout the cluster that do work in parallel.
- [x] ~~The Executors which perform multiple processes throughout the cluster that do the work in parallel~~

Yes! Executors which perform multiple processes throughout the cluster that do the work in parallel, are one of the two main processes associated with a Spark application.

- [x] ~~The Driver Program, which is a single process that creates work for the cluster.~~

Yes! The Driver Program, which is a single process that creates work for the cluster, is one of the two main processes associated with a Spark application.

2.Question 2

Why run a Spark application in local mode?

- To keep data secure
- **To test or debug an application before deployment**
- Because the application is so small, that server or cloud space is unneeded.
- To extend the number of cores available to the application.

Correct! Spark can run in local mode, which is helpful for testing or debugging an application before deployment.

3.Question 3

Based on the video, what are the described benefits of running Spark (v2.3 +) on Kubernetes as an additional deployment mode?

- Fewer developers needed for deployment
- **Containerization and better resource sharing.**
- Enhanced security
- Improved application performance on the workstation

Correct! The benefits of running Spark (v2.3 +) on Kubernetes as an additional deployment mode include containerization and better resource sharing.

Graded Quiz: Spark Architecture

1.Question 1

A Spark application has two processes, driver and executor process. Where can the driver process be run? Select all that apply.

- [x] ~~Another machine as client to the cluster~~
- [ ] Executor
- [ ] Driver
- [x] ~~Cluster node~~

2.Question 2

The Cluster Manager communicates with a cluster to acquire resources for an application. Which type of Cluster Manager is recommended for setting simple clusters?

- Kubernetes
- Apache Mesos
- Apache Hadoop YARN
- **Spark Standalone**

3.Question 3

The spark-submit script that is included with Spark to submit applications has several options you can use. Which option will show the available options per cluster manager?

- **`./bin/spark-submit --help`**
- `--class <full-class-name>`
- `--executor-cores`
- ‘deploy-mode’

Spark Runtime Environments

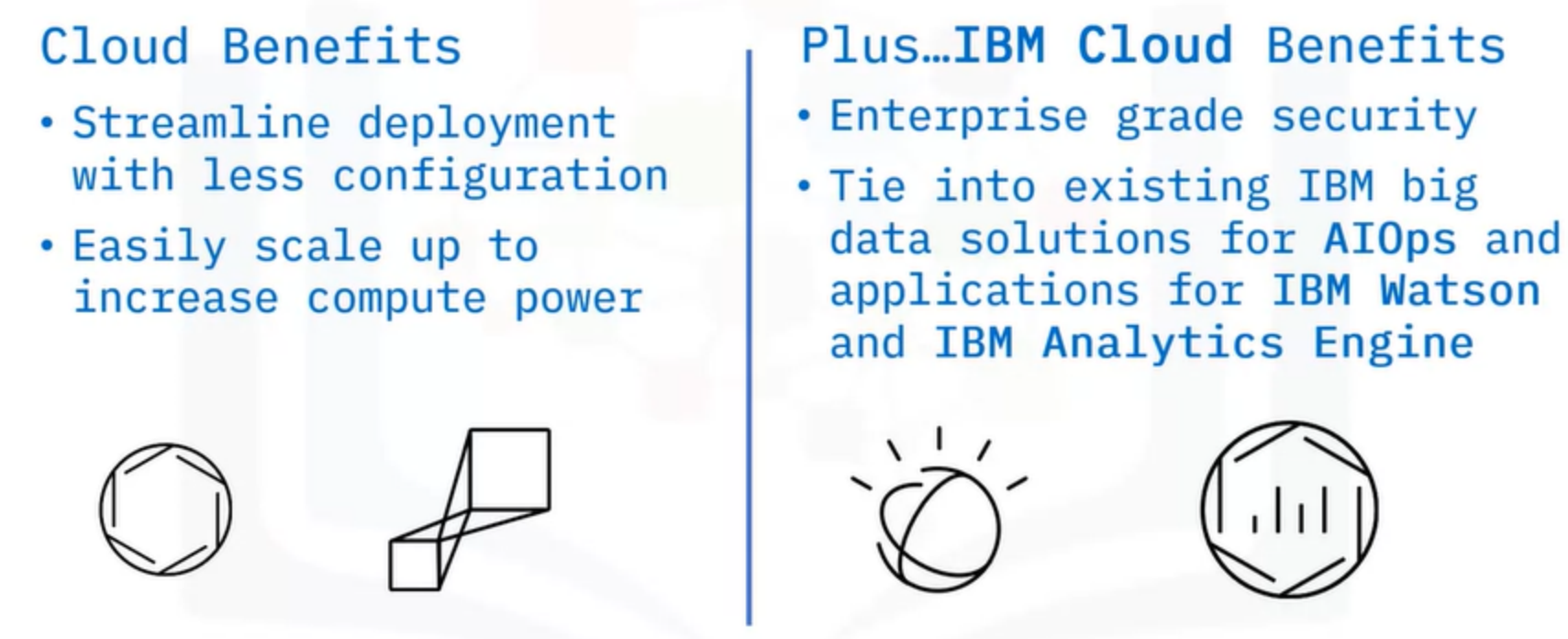
Using Apache Spark on IBM Cloud

Why use Apache Spark on IBM Cloud?

You can run Spark on a local cluster or a cloud. However, it may be difficult to configure nodes in a local cluster so that they are visible and communicate properly.

Running Spark on a cloud streamlines deployment by using pre-existing default configuration.

Makes it easy to scale up the cluster to add nodes and increase compute power as needed.



Running Spark on an IBM cloud brings added benefits:

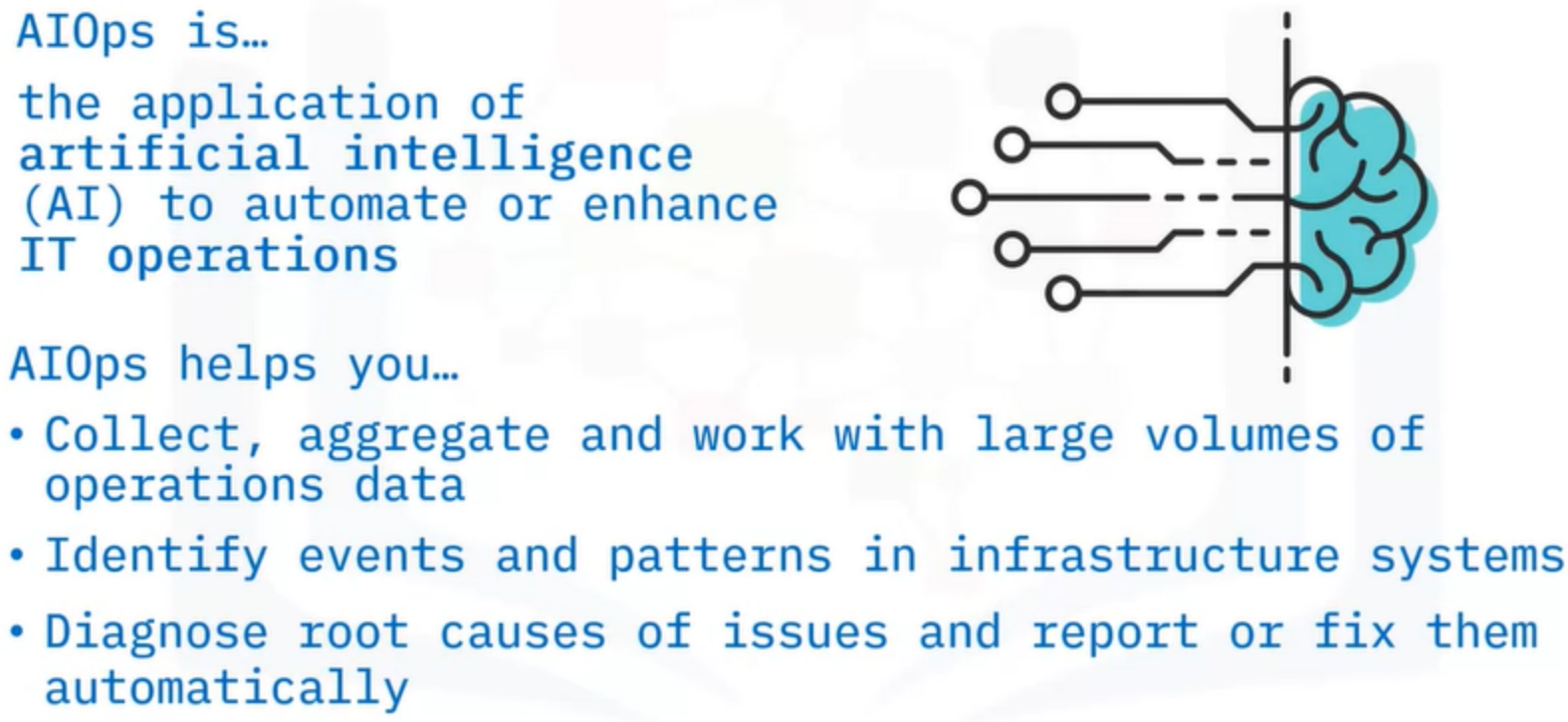
- Provides enterprise grade security.
- Tie into existing IBM big data solutions for AIOps and applications for IBM Watson and the IBM Analytics Engine.

What is AIOps?

`“AIOps” means applying artificial intelligence to automate or enhance IT operations.`

AIOps helps you:

- Collect, aggregate and work with large volumes of operations data
- Identify events and patterns in large or complex infrastructure systems
- Quickly diagnose the root causes of issues so you can report or fix them automatically.

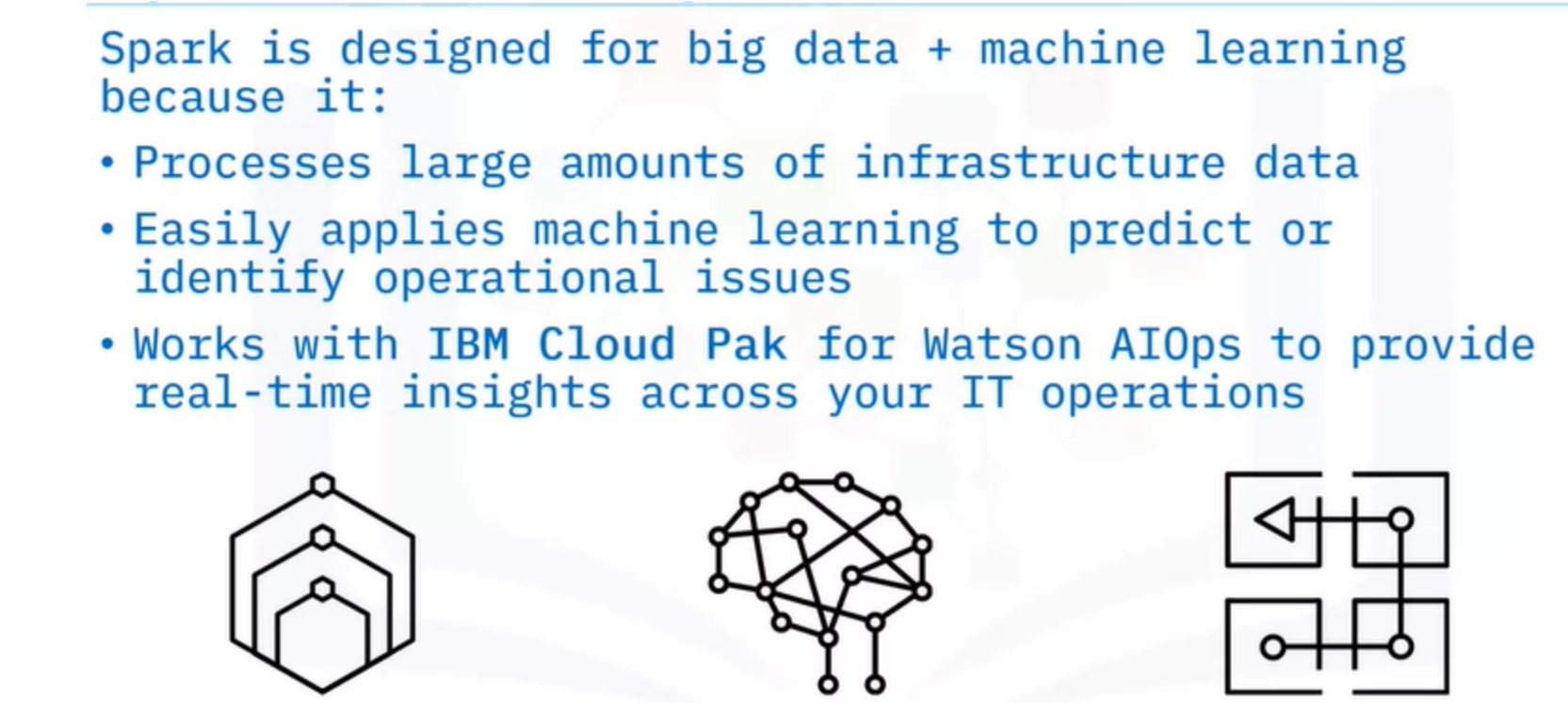


Spark and AIOps

Spark is particularly well-suited for big data analytics.

- Ideal for processing large volumes of infrastructure data
- Applying machine learning to predict or identify operational issues.
- IBM Cloud Pak for Watson AIOps offers solutions with Spark that can correlate data across your operations toolchain to bring insights or identify issues in real-time.

Examples could include data insights about tool availability, performance, or outages.



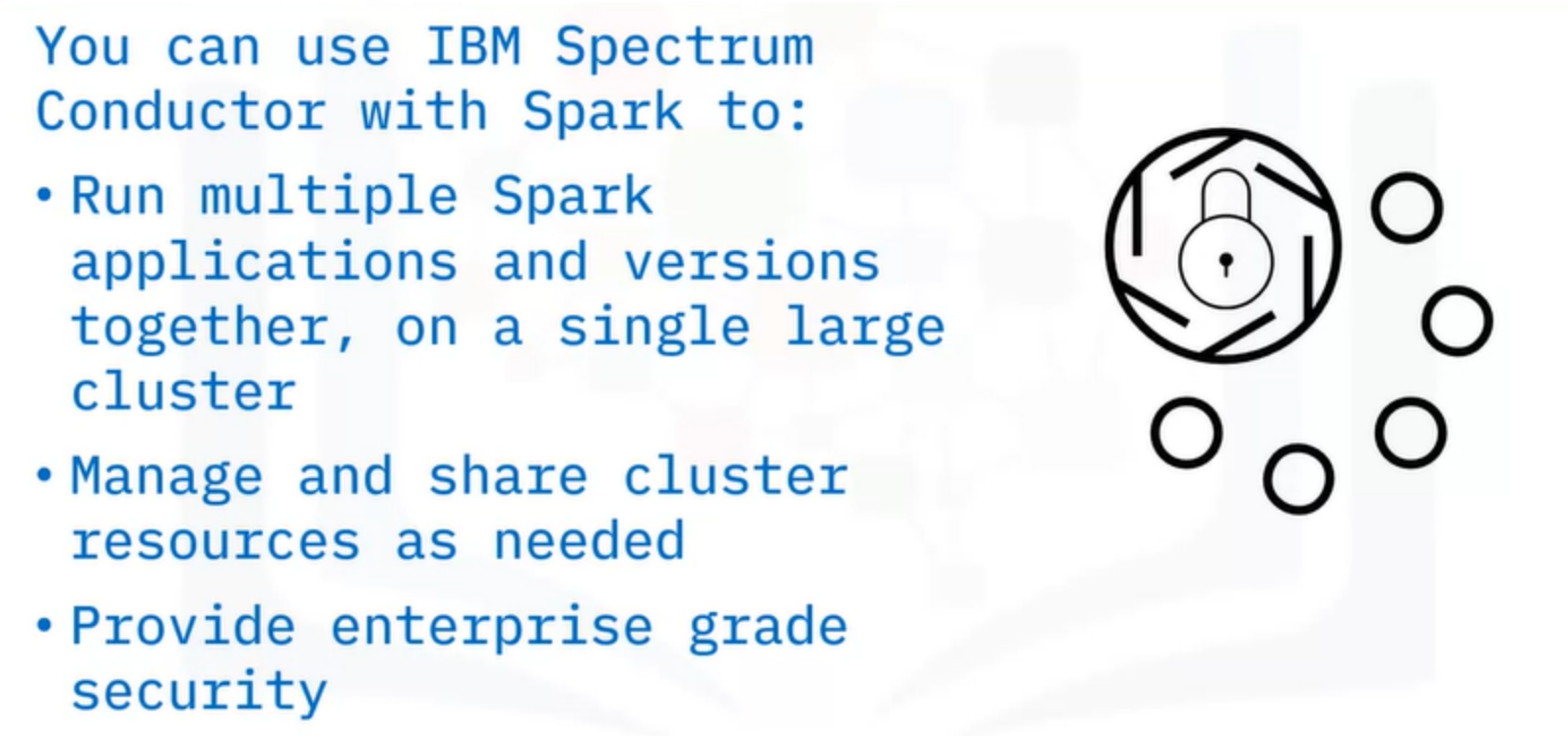
Why use Spark with IBM Spectrum Conductor?

`IBM Spectrum Conductor is a multi-tenant platform for deploying and managing Spark and other frameworks on a cluster with shared resources.`

This enables multiple Spark applications and versions to be run together, on a single large cluster.

Cluster resources can be divided up dynamically, avoiding downtime.

IBM Spectrum Conductor also provides Spark with enterprise grade security.



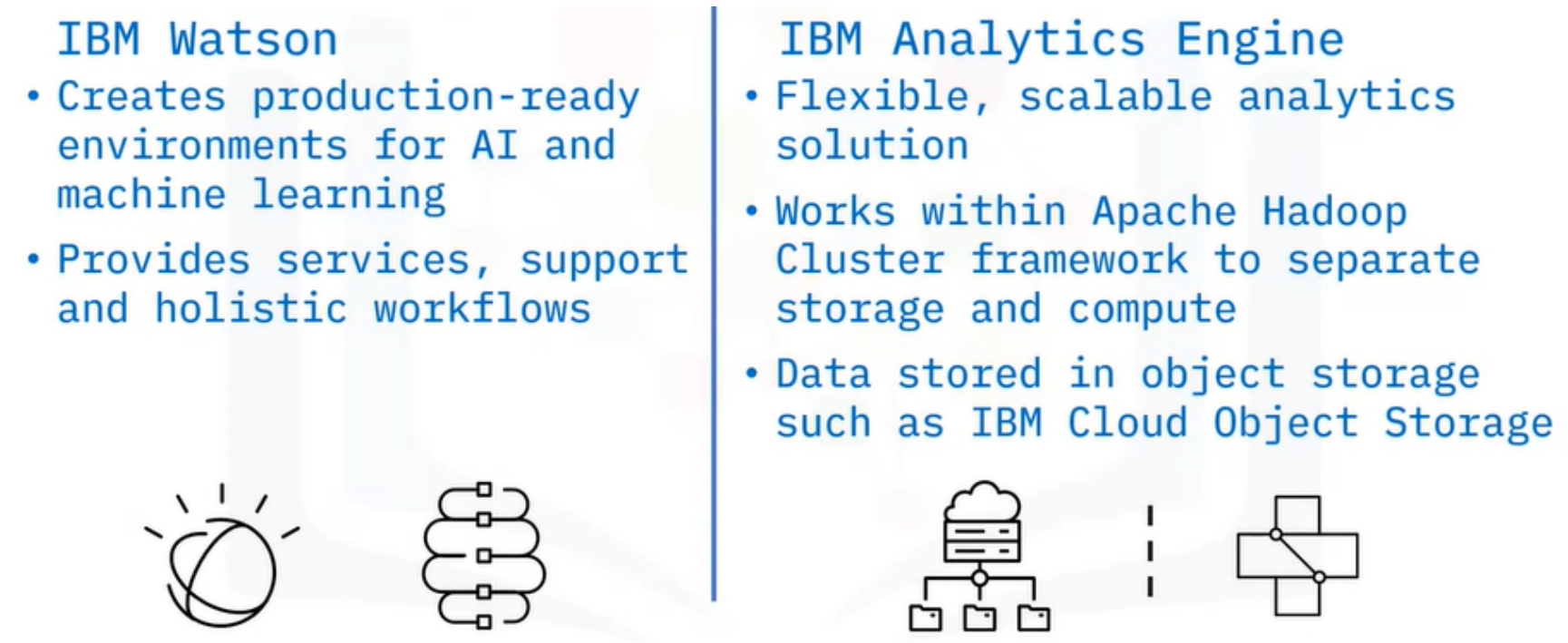
Using Spark with IBM Watson and IBM Analytics Engine

IBM Watson:

- Create production-ready environments for AI and machine learning
- Providing services, support and holistic workflows.
- Reducing setup and maintenance saves time so you can concentrate on training Spark to enhance its machine learning capabilities.

IBM Analytics Engine works with Spark:

- Provide a flexible, scalable analytics solution.
- It uses an Apache Hadoop cluster framework to separate storage and compute by storing data in object storage such as IBM Cloud Object Storage. This means you only need to run compute nodes when needed.



Conclusion

- Running Spark on IBM Cloud provides enterprise security and easily ties in IBM big data solutions for AIOps, IBM Watson, and the IBM Analytics Engine.
- AIOps tools such as IBM Watson use AI to automate or enhance IT operations.
- Spark's big data processing capabilities work well with AIOps tools, using machine learning to identify events or patterns and help report or fix issues.
- IBM Spectrum Conductor manages and deploys Spark resources dynamically on a single cluster and provides enterprise security.
- IBM Watson helps you focus on Spark's machine learning capabilities by creating production-ready environments for AI.
- IBM Analytics Engine separates storage and compute to create a scalable analytics solution alongside Spark's data processing capabilities.

Setting Apache Spark Configuration

Spark configuration types

You can configure a Spark Application using three different methods:

- **Properties**
  - Can set Spark properties to adjust and control most application behaviors, including setting properties with the driver and sharing them with the cluster.
- **Environment variables**
  - Environment variables are loaded on each machine, so they can be adjusted on a per-machine basis if hardware or installed software differs between the cluster nodes.
- **Logging configuration**
  - Spark logging is controlled by the log4j defaults file, which dictates what level of messages, such as info or errors, are logged to file or output to the driver during application execution.

You can configure Spark using three different methods:

Configuration Type	Parameters
Properties	Adjust and control application behavior
Environment variables	Adjust settings on a per-machine basis
Logging	Control how logging is output using `conf/log4j-defaults.properties`

Spark configuration location

Spark configuration files are located under the “conf” directory in the installation.

By default, there are no preexisting files after installation, however Spark provides a template for each configuration type with the filenames shown here.

You can create the appropriate file by removing the ‘.template’ extension.

Inside the template files, are sample configurations for common settings. You can enable them by uncommenting.

Configuration Type	Template File	Actual File
Spark properties	spark-defaults.conf.template	spark-defaults.conf
Environment variables	spark-env.sh.template	spark-env.sh
Logging properties	log4j.properties.template	log4j.properties

Spark property configuration

You can set properties:

1. Programmatically when creating SparkSession or using a SparkConf object
2. In the file `conf/spark-defaults.conf`
3. When launching `spark-submit` with arguments `--master`, or `--conf <key>=<value>`

```
# Set the master and additional conf when creating session
spark = SparkSession\
    .builder\
    .master(“spark://<master-url>:7077”)\
    .config(“<key>”, “<value>”)\
    .getOrCreate()
```

You can configure Spark properties in a few different ways. In the driver program, you can set the configuration

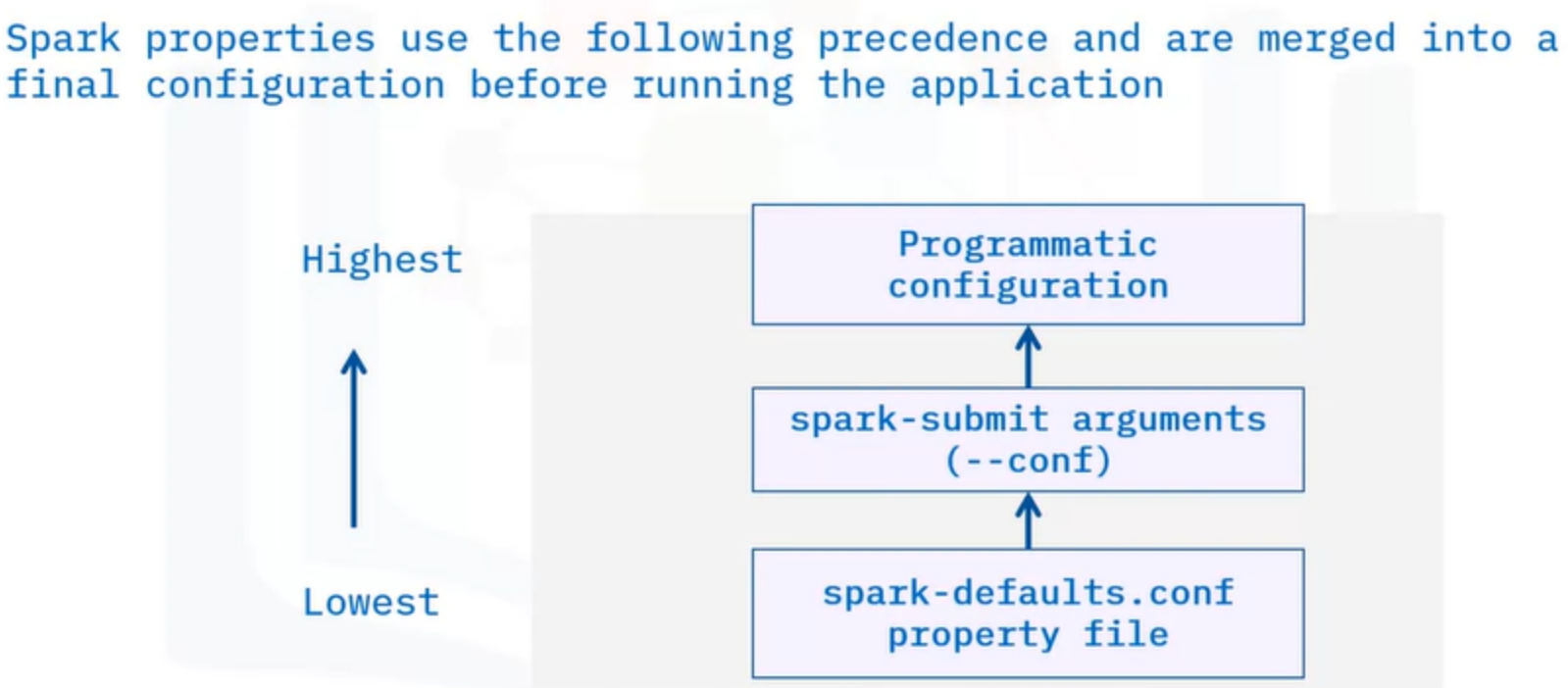
- Programmatically when creating the SparkSession or by using a separate SparkConf object that is then passed into the session constructor.
- In a configuration file found at ‘conf/spark-defaults.conf’
- When launching the application with spark-submit, either by using provided options such as ‘--master’ or using the “--conf” option with a key, value pair.

Spark property precedence

The configuration set programmatically takes the highest precedence. This means that if a configuration has already been set elsewhere or programmed into the application it will be overwritten.

Next is the configuration provided with the spark-submit script.

Last is any configuration set in the spark-defaults.conf file.



Where to use Static configuration?

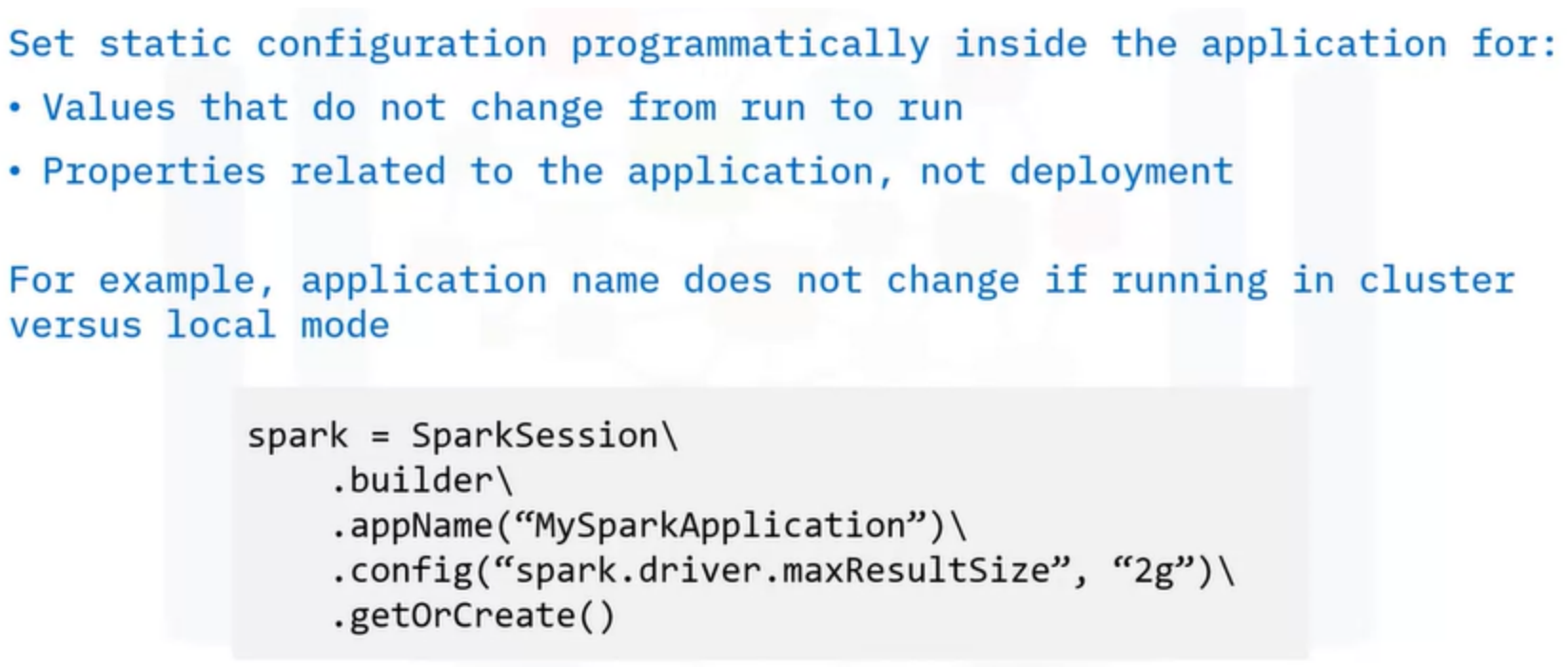
`Static configuration refers to settings that are written programmatically into the application itself`

These settings are not usually changed because it requires modifying the application itself.

Use static configuration for something that:

- Unlikely to be changed or tweaked between application runs, such as application name
- Other properties related to the application only.

For example, you would not use static configuration when it depends on what type of cluster or resource requirements are available. The example here sets the application name property and there is an additional configuration to set the maximum result size in the driver to 2 gigabytes.



Where to use Dynamic configuration?

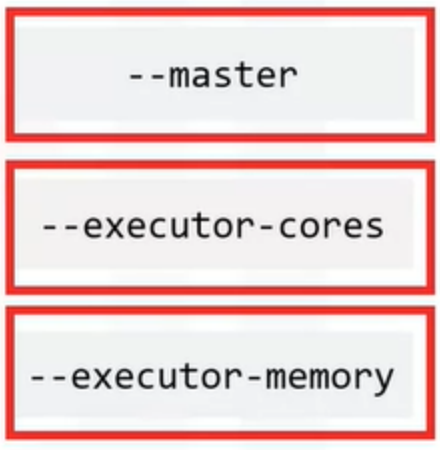
Spark dynamic configuration is useful to avoid hardcoding specific values in the application itself.

This is usually done for configuration such as:

- The master location, so that the application can be launched on a cluster by simply changing the master location.
- How many cores are used
- How much memory is reserved by each executor so that they can be properly tuned for whatever cluster the application is run on.

Setting some configuration dynamically when launching `spark-submit` means avoiding hard-coding values, such as:

- Changing which cluster the app is submitted to
- Adjusting how many cores are used by the executors
- Adjusting how much memory is reserved by each executor



Using Environment variables

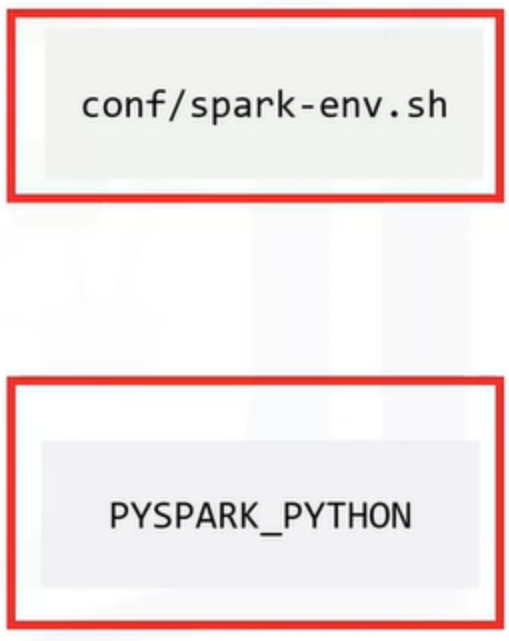
Spark environment variables are loaded from the file ‘conf/spark-env.sh’.

These are loaded from each machine in the cluster when a Spark process is started. Since these can be loaded differently for each machine, it can help configure specifics on a per-machine basis.

A common usage is to ensure each machine in the cluster uses the same Python executable by setting the “PYSPARK_PYTHON” environment variable.

Environment variables are machine specific:

- Spark loads environment variables from `conf/spark-env.sh` if it exists and is executable
- Common example is to set the Python executable used by PySpark driver and workers with `PYSPARK_PYTHON`
- This helps ensure all cluster nodes use the same Python version



Configuring Spark Logging

Spark logging is controlled using log4j and the configuration is read through ‘conf/log4j-properties’.

Here you can adjust:

- A log level to determine which messages (such as debug, info or errors) are shown in the Spark logs.
- Log4j allows the configuration to set where the logs are sent to and adjust specific components of Spark or third-party libraries.

Spark reads logging configuration in the file `conf/log4j.properties`, where you:

- Set a log level to control logging output of driver and executor processes
- Control master and worker logging for Spark Standalone

conf/log4j.properties

```
$ bin/spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
Spark context Web UI available at http://192.168.1.33:4040
Spark context available as 'sc' (master = local[*], app id = local-1614631042181).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_282)
Type in expressions to have them evaluated.
Type :help for more information.
scala>
```

Conclusion

- You can set Spark configuration using properties (to control application behavior), environment variables (to adjust settings on a per-machine basis) or logging properties (to control logging outputs).
- Spark property configuration follows a precedence order, with the highest being configuration set programmatically, then spark-submit configuration, and lastly configuration set in the spark-defaults.conf file.
- Static configuration should be used for values that don't change from run to run or properties related to the application, such as application name.
- Dynamic configuration should be used for values that change or need tuning when deployed, such as master location, executor memory or core settings.

Running Spark on Kubernetes

What is Kubernetes

Kubernetes (also abbreviated as "k8s"), is a popular framework for running containerized applications on a cluster.

- Open-source system
- Highly scalable
- Provides flexible deployments to the cluster.
- Portable, so it can be run in the same way whether you are running it in the cloud or on-premises.

How is Kubernetes used?

Kubernetes is used to manage containers that run distributed systems, such as Spark, in a more flexible and resilient way.
It provides flexibility in the way cluster resources are used and is more resilient if an application errors out or nodes fail.

Kubernetes has many benefits, including:

- Network service discovery
- Cluster load balancing
- Automating scale up and down of systems on the cluster, making the cluster easier to manage with maximum up-time
- Orchestrating storage to make better use of what is available.

Kubernetes - Locally hosted

While Kubernetes is designed to work on a cluster, it also supports being hosted locally, on a single machine.

You can try out a locally-hosted Kubernetes cluster

- Setting it up using tools such as minikube.
  This is commonly used as a `development environment`

- Enabling you to apply changes locally first and test before putting them into production on a cluster in the cloud.

💡 Containerization means that applications run the same way locally as they would on the cluster – hence you can identify and resolve any issues more quickly and easily.

Cloud-hosted Kubernetes

Running a Kubernetes cluster on a cloud is a good choice for `production environments` because of the ease of deployment and flexibility it provides.

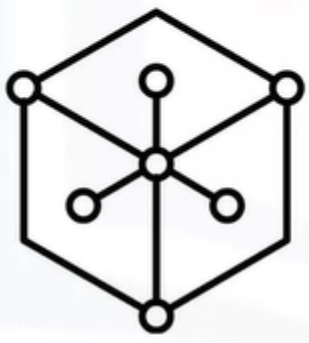
There are two ways to get started with Kubernetes on the cloud.

- Use existing tools to bootstrap the cluster and get the necessary components in place.
- Use a certified Kubernetes provider with a turnkey option to accelerate getting a Kubernetes cluster up.

Running Spark on Kubernetes

Run Spark (v2.3 +) on Kubernetes as an additional deployment mode, with benefits including:

- Containerization – applications are more portable and easier to manage dependencies
- Better resource sharing – multiple Spark applications can run concurrently and in isolation

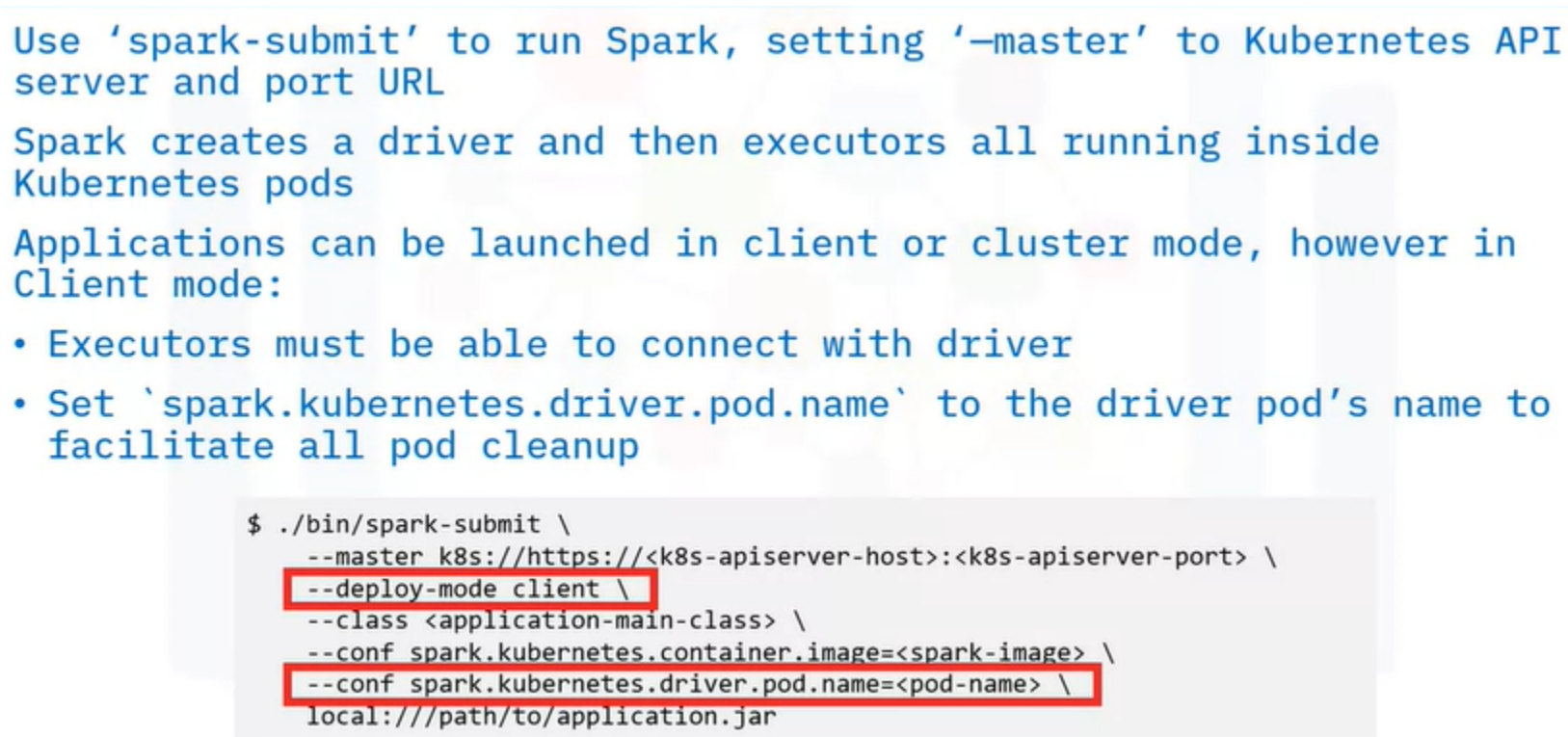


Spark version 2.3 and above can be run on top of Kubernetes as an alternative deployment mode when submitting a Spark application.

Using Spark on Kubernetes comes with some very useful benefits.

- `Containerization` means Spark applications are more portable. It makes it easier to manage dependencies and setup the required environment throughout the cluster.
- Containerization supports `better resource sharing`.
  Multiple Spark applications running concurrently can easily share the available resources and run in isolation of each other.

Submiting Spark apps on Kubernetes



To submit a Spark application to a Kubernetes cluster, set the ‘--master’ option to the URL for the Kubernetes API server and port.

Spark creates a driver which in turn creates executors, with both driver and executors running inside Kubernetes pods.

`A ‘pod’ is a tightly coupled group of containers with shared resources.`

The application can be deployed as client or cluster mode.

However, for client mode some additional considerations are needed.

- The executors must be able to communicate and connect with the driver program.
- Use the driver’s pod name to set ‘spark.kubernetes.driver.pod.name’. This will ensure the executor pods are cleaned up along with the driver pod when the application finishes

Conclusion

- Kubernetes runs containerized applications on a cluster, managing distributed systems such as Spark in a more flexible, resilient way.
- Kubernetes can run locally as a deployment environment, useful for trying out changes before deploying to clusters in the cloud.
- Kubernetes can be hosted on private or hybrid clouds, set up using existing tools to bootstrap clusters, or turnkey options from certified providers.
- While Spark can be launched in client or cluster mode, in Client mode, executors must be able to connect with the driver and pod cleanup settings are required.

Labs

Hands-on Lab: Apache Spark on Kubernetes: https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-BD0225EN-SkillsNetwork/labs/kube_lab.md.html

---

Summary & Highlights

- Running Spark on IBM Cloud provides enterprise security and easily ties in IBM big data solutions for AIOps, IBM Watson and IBM Analytics Engine. Spark’s big data processing capabilities work well with AIOps tools, using machine learning to identify events or patterns and help report or fix issues. IBM Spectrum Conductor manages and deploys Spark resources dynamically on a single cluster and provides enterprise security. IBM Watson helps you focus on Spark’s machine learning capabilities by creating automated production-ready environments for AI. IBM Analytics Engine separates storage and compute to create a scalable analytics solution alongside Spark’s data processing capabilities.
- You can set Spark configuration using properties (to control application behavior), environment variables (to adjust settings on a per-machine basis) or logging properties (to control logging outputs). Spark property configuration follows a precedence order, with the highest being configuration set programmatically, then spark-submit configuration

and lastly configuration set in the spark-defaults.conf file. Use Static configuration options for values that don't change from run to run or properties related to the application, such as the application name. Use Dynamic configuration options for values that change or need tuning when deployed, such as master location, executor memory or core settings.

- Use Kubernetes to run containerized applications on a cluster, to manage distributed systems such as Spark with more flexibility and resilience. You can run Kubernetes as a deployment environment, which is useful for trying out changes before deploying to clusters in the cloud. Kubernetes can be hosted on private or hybrid clouds, and set up using existing tools to bootstrap clusters, or using turnkey options from certified providers. While you can use Kubernetes with Spark launched either in client or cluster mode, when using Client mode, executors must be able to connect with the driver and pod cleanup settings are required.

Quiz

Practice Quiz: Spark Runtime Environments

Question 1

Select the options that correctly fill in the blank. Spark is a good programming option for big data and machine learning because it _____ :

- [ ] Processes infrastructure data on local servers leaving cloud servers free for application processing.
- [x] ~~Works with IBM Cloud Pak for Watson AIOps to provide real-time insights across your IT operations.~~

Yes! Spark works with IBM Cloud Pak for Watson AIOps to provide real-time insights across your IT operations. In addition, Spark is a good programming option for big data and machine learning because it Processes large amounts of infrastructure data and easily applies machine learning to predict or identify operational issues.

- [x] ~~Processes large amounts of infrastructure data and easily applies machine learning to predict or identify operational issues.~~

Correct! Spark is a good programming option for big data and machine learning because it Processes large amounts of infrastructure data and easily applies machine learning to predict or identify operational issues.

- [ ] Enables programmers to hand off coding to AI assistants.

Question 2

Spark enables both static and dynamic configurations. Select the options recommended for static configuration.

- Adjusting Which cluster the app is submitted to
- Adjusting the amount of reserved executor memory
- Adjusting the number of cores assigned.
- **The application name**

Question 3

Select the answers that describe what Kubernetes is and its primary benefits

- [ ] Is only suitable for cloud-based deployments.
- [x] ~~Provides network service discovery, cluster load balancing, automated scaling, orchestrated storage~~

Correct! Kubernetes benefits application performance with features including network service discovery, cluster load balancing, automated scaling, and orchestrated storage.

- [x] ~~Runs the same way whether in the cloud or on-premises~~

Correct! Kubernetes provides developers with a portable, consistent experience whether working with an application on-premises or in the cloud.

- [x] ~~Runs containerized applications on a cluster providing flexible, automated deployments~~

Correct! Kubernetes runs containerized applications on a cluster providing flexible, automated deployments.

Graded Quiz: Spark Runtime Environments

Question 1

What is one of the advantages of using Spark on IBM Cloud? Select all that apply.

- [ ] Better communication for local cluster nodes
- [x] ~~Pre-existing default configuration~~

Correct, running Spark on a cloud streamlines deployment with pre-existing default configuration.

- [ ] Easy to configure local cluster nodes
- [x] ~~Enterprise grade security~~

Question 2

Spark properties have precedence and are merged into a final configuration before running the application. Select the statement that describes the order of precedence.

- Set the spark-submit configuration, set configurations in the spark-defaults.conf file, and lastly perform programmatic configuration.
- Perform programmatic configuration, set configurations in the spark-defaults.conf file, and lastly set the spark-submit configuration.
- Set configurations in the spark-defaults.conf file, set the spark-submit configuration and lastly perform programmatic configuration.
- **Perform programmatic configuration, set spark-submit configuration and lastly set configurations in the spark-defaults.conf file.**

Question 3

What is the first command to run when submitting a Spark application to a Kubernetes cluster?

- 'spark.kubernetes'
- '--deploy-mode client'
- '--conf spark.kubernetes.driver.pod.name'
- **Set the '--master' option to the Kubernetes API server and port**