Using Apache Spark on IBM

Objectives

After watching this video, you will be able to:

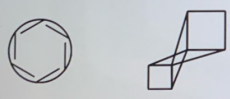
- List the benefits of using Apache Spark ("Spark") on IBM Cloud
- Define AIOps and describe how Spark can be used within AIOps
- Describe how to use Spark with IBM Spectrum Conductor, IBM Watson and the IBM Analytics Engine

- can run spark on local mode but it's difficult to configure

Why Use Spark on IBM Cloud

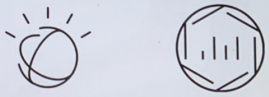
Cloud Benefits

- Streamline deployment with less configuration
- Easily scale up to increase compute power



Plus…IBM Cloud Benefits

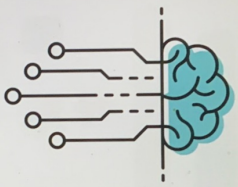
- Enterprise grade security
- Tie into existing IBM big data solutions for AIOps and applications for IBM Watson and IBM Analytics Engine



What is AIOps

AIOps is…

the application of artificial intelligence (AI) to automate or enhance IT operations



AIOps helps you…

- Collect, aggregate and work with large volumes of operations data
- Identify events and patterns in infrastructure systems
- Diagnose root causes of issues and report or fix them automatically

Spark and AIOps

Spark is designed for big data + machine learning because it:

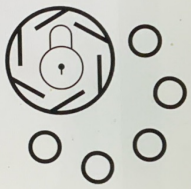
- Processes large amounts of infrastructure data
- Easily applies machine learning to predict or identify operational issues
- Works with IBM Cloud Pak for Watson AIOps to provide real-time insights across your IT operations

Why use Spark with IBM Spectrum Conductor

- multi-tenant platform for deploying and managing Spark and other frameworks on a cluster with shared resources

You can use IBM Spectrum Conductor with Spark to:

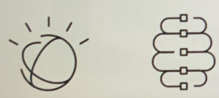
- Run multiple Spark applications and versions together, on a single large cluster
- Manage and share cluster resources as needed
- Provide enterprise grade security



Using Spark with IBM Watson and IBM Analytics Engine

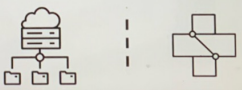
IBM Watson

- Creates production-ready environments for AI and machine learning
- Provides services, support and holistic workflows



IBM Analytics Engine

- Flexible, scalable analytics solution
- Works within Apache Hadoop Cluster framework to separate storage and compute
- Data stored in object storage such as IBM Cloud Object Storage



Setting Apache Spark Configuration

Objectives

After watching this video, you will be able to:

- Describe the configuration types of an Apache Spark ("Spark)" application
- Explain the purpose and common options for each configuration type
- Describe when to use static or dynamic configuration

Spark Configuration Types

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

- Configuration files are located under the `conf/` directory in the Spark installation
- Files are not created by default, however Spark provides template files that can be renamed as shown in the table

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

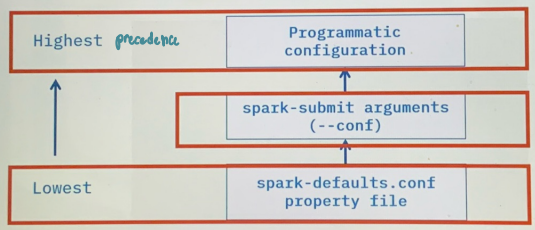
You can set properties:

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()
```

Spark Property Precedence

Spark properties use the following precedence and are merged into a final configuration before running the application



Where to Use Static Configuration

Set static configuration programmatically inside the application for:

- Values that do not change from run to run
- Properties related to the application, not deployment

For example, application name does not change if running in cluster versus local mode

```
spark = SparkSession\
    .builder\
    .appName("MySparkApplication")\
    .config("spark.driver.maxResultSize", "2g")\
    .getOrCreate()
```

When to Use Dynamic Configuration

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

`--master`

`--executor-cores`

`--executor-memory`

Using Environment Variables

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

`conf/spark-env.sh`

`PYSPARK_PYTHON`

Configuring Spark Logging

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>
```

Running Spark on Kubernetes

Objectives

After watching this video, you will be able to:

- Describe when and why Kubernetes is used
- Describe how to run Kubernetes locally versus hosted on the cloud
- Explain how to run Apache Spark ("Spark") on Kubernetes

What is Kubernetes?

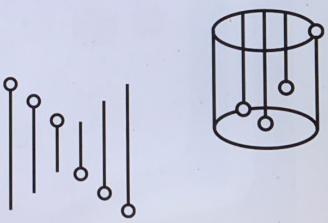
Kubernetes (or "k8s") runs containerized applications on a cluster and is:

- Open-source
- Highly scalable
- Provides flexible, automated deployments
- Portable, so can be run in the same way whether in the cloud or on-premises

How is Kubernetes used?

Kubernetes manages containers that run distributed systems in a more resilient and flexible way, with benefits including:

- Network service discovery
- Cluster load balancing
- Automated scale up and down
- Orchestrating storage



Kubernetes – Locally Hosted

Host Kubernetes locally as a development environment or to try it out:

- Set up a local Kubernetes cluster on a machine using tools such as minikube
- Then apply changes to a cluster in the cloud when ready for production

Cloud-Hosted Kubernetes

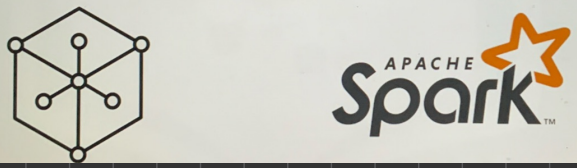
Running a Kubernetes cluster on a private or hybrid cloud:

- Is common for production environments
- Uses existing tools to bootstrap clusters with components or turnkey options from certified Kubernetes providers

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



Submitting Spark Apps on Kubernetes

Use 'spark-submit' to run Spark, setting '–master' to Kubernetes API server and port URL

Spark creates a driver and then executors all running inside Kubernetes pods

Applications can be launched in client or cluster mode, however in Client mode:

- Executors must be able to connect with driver
- Set `spark.kubernetes.driver.pod.name` to the driver pod's name to facilitate all pod cleanup

```
$ ./bin/spark-submit \
    --master k8s://https://<k8s-apiserver-host>:<k8s-apiserver-port> \
    --deploy-mode client \
    --class <application-main-class> \
    --conf spark.kubernetes.container.image=<spark-image> \
    --conf spark.kubernetes.driver.pod.name=<pod-name> \
    local:///path/to/application.jar
```