BIG DATA
CLOUD ANALYSIS

READ MORE

PRESENTATION BY

Jacop Antony L
Manikandan N
Saravanaperumal S
Vishwa P
Jeevanantham SS

What is Cloud Foundry?in Big Data

Cloud Foundry is the premier industry standard Platform-as-a-Service (PaaS) that ensures the fastest, easiest, and most reliable deployment of cloud-native apps. Cloud Foundry ensures that the build and deploy aspects of coding remain carefully coordinated with any attached services; resulting in quick, consistent, and reliable iterating of apps.

Benefits of Cloud Foundry

Choose your own language - IBM Cloud® Foundry includes runtimes for Java™, Node.js, PHP, Python, Ruby, Swift, and Go. Cloud Foundry community build packs are also available. Combined with DevOps services, the app runtimes enable a delivery pipeline that automates much of the iterative development process.

Fault tolerant

Runtimes facilitate developing apps as stateless processes that quickly: start and stop, replicate if an instance fails, and duplicate if sustained or increased performance requires.

About Cloud Foundry in Big Data

Cloud Foundry is an open source, multicloud platform as a service (PaaS) for developing and deploying enterprise cloud applications. It is governed by the Cloud Foundry Foundation, an independent non-profit organization, and hosted by The Linux Foundation.

Extend apps with services

Runtimes link IBM Cloud services to apps as endpoints, giving any instance of an app embedded knowledge of how to manage relevant calls and data. In fact, runtimes manage all linked resources this way: SDKs, APIs (whether made available as cloud services or exposed from within a traditional enterprise as custom services), and also apps themselves when used as resources by other apps.

Big Data Cloud Foundry

```
from pyspark import SparkContext, SparkConf

# Initialize Spark
conf = SparkConf().setAppName("WordCount")
sc = SparkContext(conf=conf)

# Read data
text_file = sc.textFile("data.txt")

# Word count
word_count = text_file.flatMap(lambda line: line.split(" ")) \
                      .map(lambda word: (word, 1)) \
                      .reduceByKey(lambda a, b: a + b)

# Print results
for (word, count) in word_count.collect():
    print(f"{word}: {count}")

# Stop Spark
sc.stop()
```

OUTPUT

WORD1: 5
WORD2: 10
WORD3: 3
...

Thank you