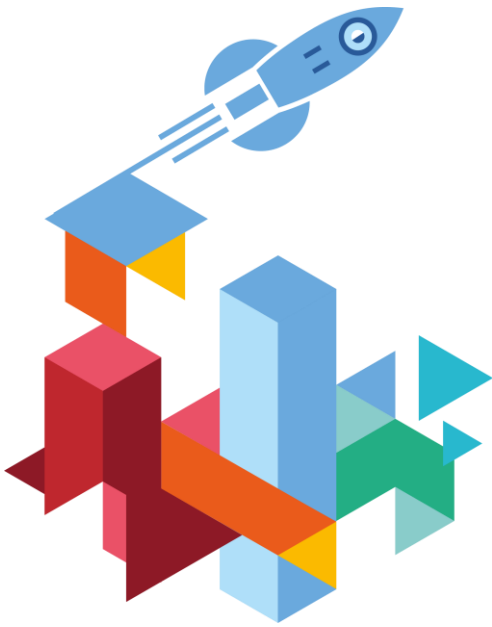




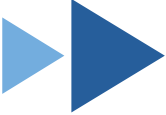
STRATOSCALE




# A HYBRID CLOUD APPROACH TO BIG DATA IN THE ENTERPRISE

[www.stratoscale.com](http://www.stratoscale.com)


US Phone: +1 877 420-3244 | Email: [sales@stratoscale.com](mailto:sales@stratoscale.com)



Deliver true hybrid and consistent environment, leveraging off-prem and on-prem capacity



Innovate fast: deliver easy to consume, on-demand and self-service cloud capabilities



Maximize existing IT investments, significantly reduce IT operation complexity

# TABLE OF CONTENT

▶ Introduction	04
▶ Use case overview	05
▶ Deployment details	07
▶ 1. Deploying an RDS instance on Chorus	07
▶ 2. Deploying a Kubernetes cluster on Chorus	10
▶ 3. Deploying a standalone Spark cluster on Kubernetes	12
▶ 4. Running Spark applications as Kubernetes Jobs	14
▶ Conclusions	16
▶ Appendices	17

# INTRODUCTION

All IT managers understand the critical role they play in ensuring that the enterprise computing infrastructure under their control is effective in meeting the business goals of the organization. Indeed, staying competitive in the rapidly evolving world of today requires that businesses take a proactive mindset towards leveraging their IT investments to deliver key insights through big data initiatives. The adoption of this mentality is often reflected by the inclusion of dedicated data science teams capable of driving these projects. These groups apply advanced analytical algorithms and artificial intelligence to large datasets with a goal of deriving competitive advantages for the organization. The specific implementations vary and may impact a variety of business processes such as product development, marketing initiatives, sales management, etc. However, as the adoption of these practices continues to increase, IT teams need to be prepared to effectively support them.

While on the surface it may seem that big data workloads are simply another application to deploy and manage, in practice they entail multiple challenges. First and foremost, big data use cases typically require the integration of multiple application and storage components. These issues are often exacerbated by the fact that data may be located in a different location than where the key insights need to be derived and stored. Finally, data science teams often employ a combination of frameworks and dataflows, so there's no "one size fits all" deployment strategy. Fortunately, these challenges can be effectively addressed through the adoption of enabling technology solutions.

Over the past decade cloud computing has quickly transitioned from a nascent technology to the de facto standard for many workloads found in modern enterprises. The ability to quickly allocate and manage resources in a self-service manner, reduce costs through efficiency gains, and easily manage fluctuations in capacity needs are just some of the motivating reasons for this shift in the market. Indeed, while early adoption typically occurred in the context of public cloud offerings such as Amazon Web Services (AWS) <sup>[1]</sup>, today there is a clear demand for similar operating models within the on-premises datacenter as well. Stratoscale<sup>[2]</sup> is an example of a solution that allows IT administrators to easily implement an AWS-compatible region within their data center using existing hardware assets. When combined with a public cloud infrastructure, the resulting hybrid cloud capabilities can prove advantageous in supporting big data use cases by allowing analysts to manage their resources and data flows in a self-service manner.

In this paper, we demonstrate the ability to implement a big data use case in practice using a hybrid cloud deployment strategy. We provide an overview of our sample scenario, followed by a detailed deployment walk through that enables users to easily replicate the scenario in their own environments. By the end of this paper, readers should have a clear understanding of how big data use cases can be implemented using the underlying technologies covered and how the adoption of hybrid cloud computing environments enables IT leaders to successfully meet the needs of these initiatives within their organizations.

# USE CASE OVERVIEW

We consider a scenario where a data science team needs to analyze a dataset that resides within an object storage service on a public cloud. For example, this is a common scenario when an externally exposed service is deployed on a public cloud and telemetry data is stored for future analysis. It is common in these cases for the subsequent analysis to happen on-premises where the data processing algorithms can integrate with high-value business databases that reside within the enterprise data center.

In our scenario, we make use of a specific set of technologies which are representative of those that are typically adopted as part of big data and cloud computing deployment paradigms. The technologies are summarized

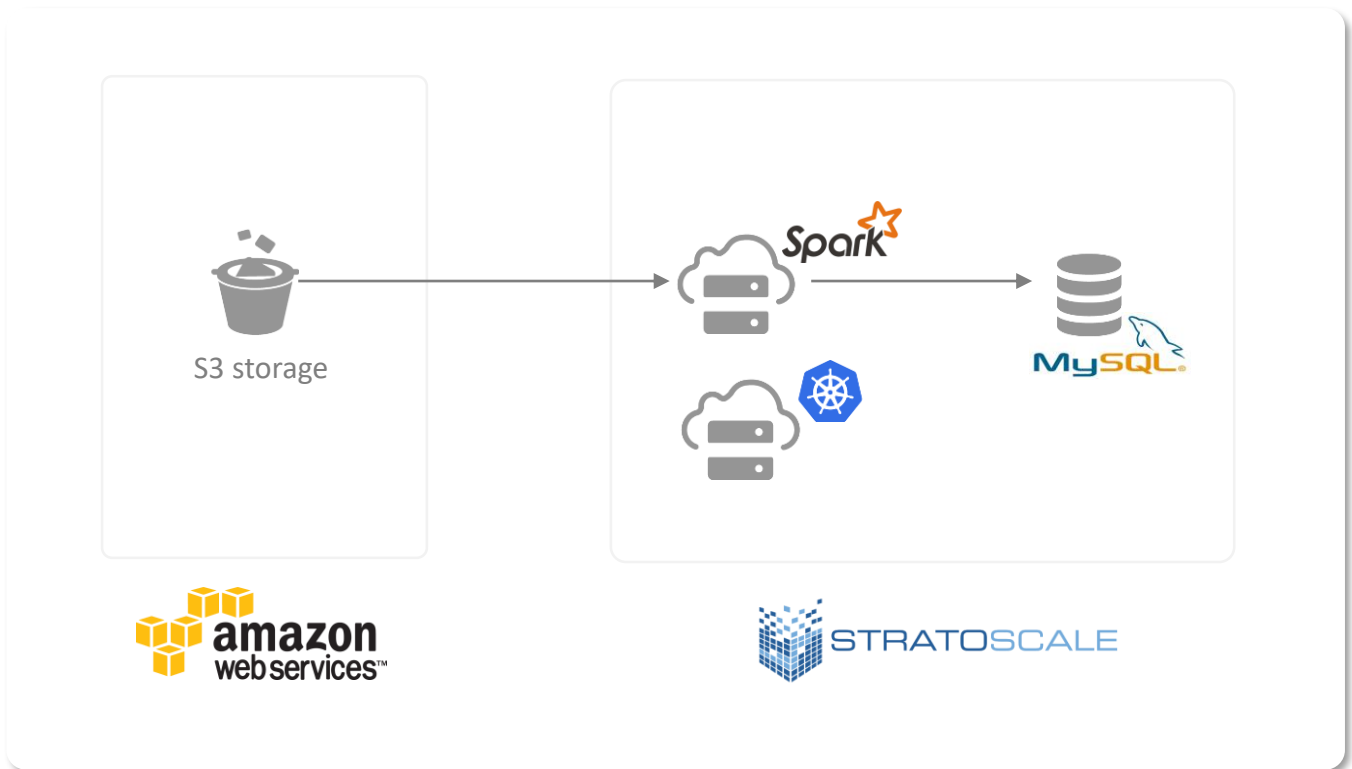
in Table 1 along with their location, where we use AWS and a Stratoscale cluster for public and private cloud environments respectively. S3 is a common choice for unstructured data due to its simple model, low cost, and scalability. We select MySQL as our structured database to store results from analytics applications. Finally, we use Spark<sup>[3]</sup> as the underlying data processing framework, where we deploy the Spark cluster using Kubernetes<sup>[4]</sup>. Spark is a common selection by data science teams due to its flexible programming model as well as support for multiple languages including Python, Java and Scala. Kubernetes allows users to easily deploy containerized application code and is a natural fit for managing and scaling Spark clusters.

**TABLE 1:  
TECHNOLOGIES INCORPORATED INTO THE BIG DATA USE CASE**

Logical component	Technology	Location
Object storage	S3 storage	Public cloud
Structured storage	MySQL database	Private cloud
Orchestration	Kubernetes	Private cloud
Data processing	Spark	Private cloud

Figure 1 illustrates the overall dataflow achieved through the integration of these technologies across the cloud environments. The data stored on AWS S3 is accessed via a Spark application executed via an on-premises Spark cluster. The Spark cluster itself is deployed through a Kubernetes cluster managed by Stratoscale. Upon completion of its analysis, the Spark application stores its result to a MySQL database within the same environment.

**FIGURE 1:  
DATAFLOW ACROSS PUBLIC AND PRIVATE CLOUD ENVIRONMENTS**



# DEPLOYMENT DETAILS

As a specific instantiation of the scenario depicted in Figure 1, our deployment implementation is composed of the following on-premises resources:

- A MySQL instance
- A three node Kubernetes cluster
- A Spark cluster comprised of a single master and two worker instances

We also assume the existence of:

- An S3 bucket on AWS
- A utility host / VM which is able to access on-premise resources

This section outlines the specific steps taken to implement this deployment and execute a sample Spark application to validate the system end-to-end.

## 1. DEPLOYING AN RDS INSTANCE ON STRATOSCALE CHORUS

A MySQL database instance can be instantiated using the AWS-compatible relational database service (RDS) within Stratoscale Chorus. As illustrated in Figure 2, the database menu item can be found under Cloud Services on the Symphony menu, where users can select Instances to open the list of current resources as well as create new databases.

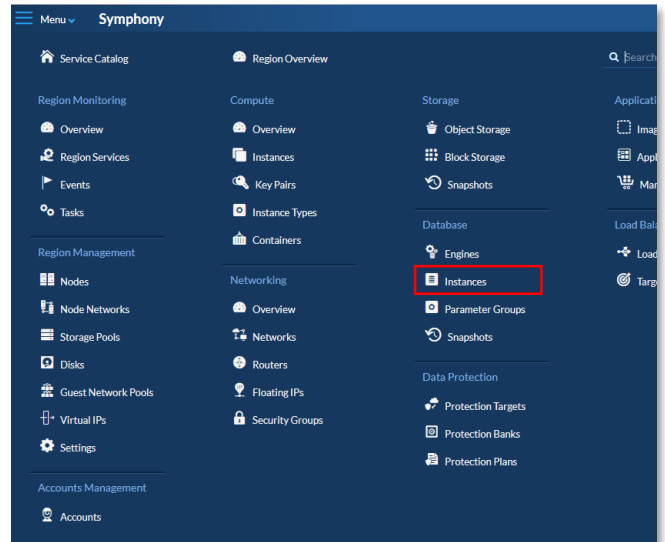
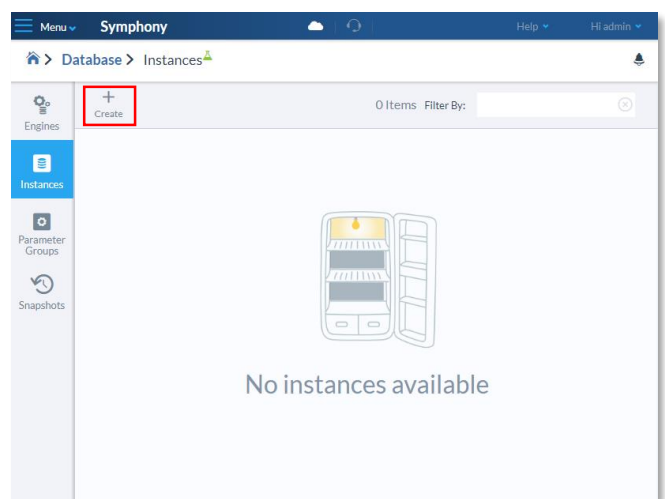


Figure 2: Selecting RDS instance menu option

The following screenshots illustrate the subsequent steps for configuring and deploying a MySQL instance using the Stratoscale Symphony UI:

### Step 1: Select “Create Instance”



### Step 2: Populate instance name and select engine version, instance type, data volume and storage pool (here we use MySQL 5.6)

Create Database Instance
✕

---

1 General
2 Network
3 Credentials

Instance Name\*

Description

Engine Version\*  ✕

Parameter Group  ✕

Instance Type\*  ✕

Data volume\*  GB

Storage Pool\*  ✕

Cancel
Next

### Step 4: Configure database name and user

Create Database Instance
✕

---

General
Network
3 Credentials

Database Name\*

Master User Name\*

Master Password\*

Confirm Password\*

Cancel
Back
Finish

### Step 3: Select a network

Create Database Instance
✕

---

General
2 Network
3 Credentials

Network\*  ✕

Edge Network  ▼

Database Port

Cancel
Back
Next



Once the instance has been launched, the status can be viewed in the dashboard as shown in Figure 3. Once the database state is “Active”, more details regarding the configuration can be obtained by clicking on the name link. The resulting view as shown in Figure 4 provides the IP address where the database can be reached.

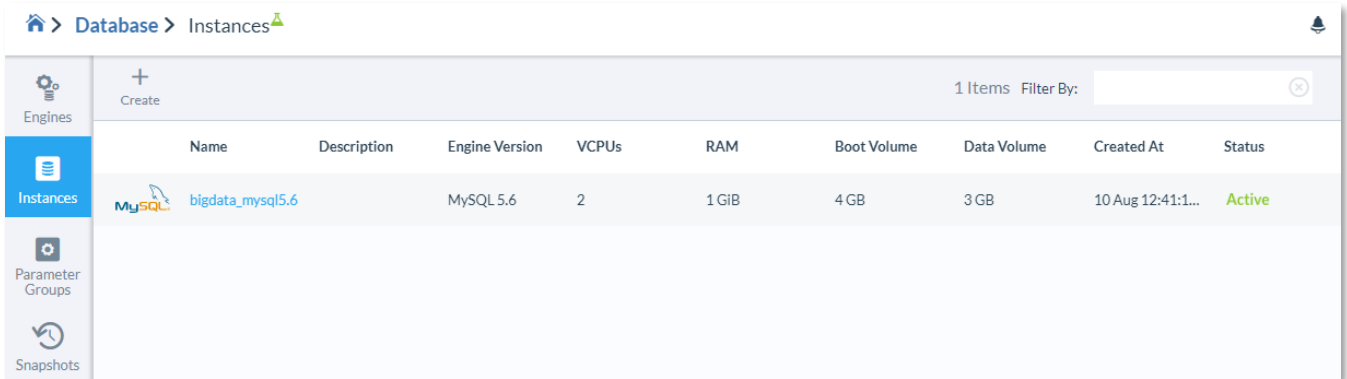


Figure 3: Overview of RDS instances and state

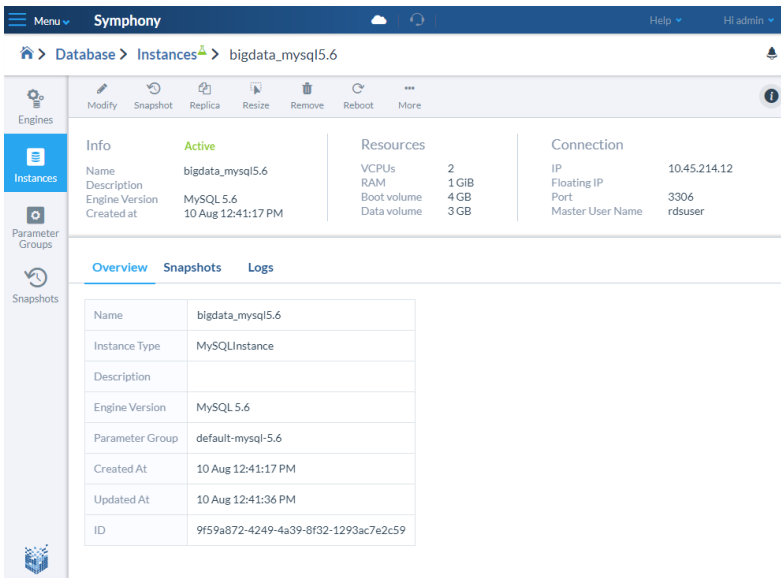


Figure 4: RDS instance detailed view

As a final confirmation step, the database instance can be connected to and configured using the MySQL command line utility. Figure 5 displays a basic data table definition for our sample Spark application to store results in the test database setup as part of the RDS deployment.

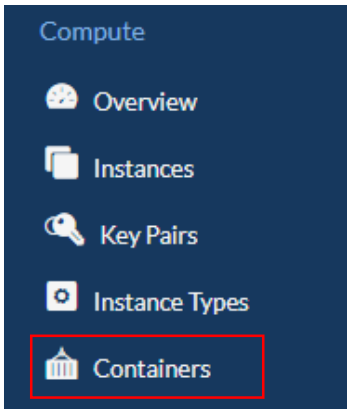
```
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| data            |
+-----+
1 row in set (0.00 sec)

mysql> describe data;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default        | Extra |
+-----+-----+-----+-----+-----+-----+
| value | int(11)| YES  |     | NULL           |       |
| created | timestamp | YES  |     | CURRENT_TIMESTAMP |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Figure 5: Accessing and configuring MySQL tables via command line

## 2. DEPLOYING A KUBERNETES CLUSTER ON SYMPHONY

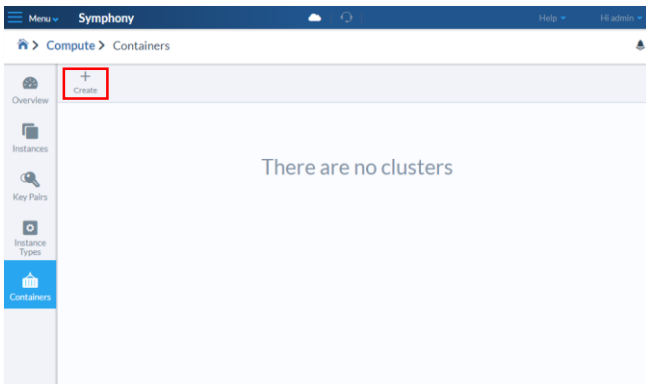
Similar to RDS, Stratoscale Symphony simplifies the deployment of Kubernetes clusters using a native container service. Figure 6 highlights the corresponding menu item which, again, can be found under “Compute Services”.



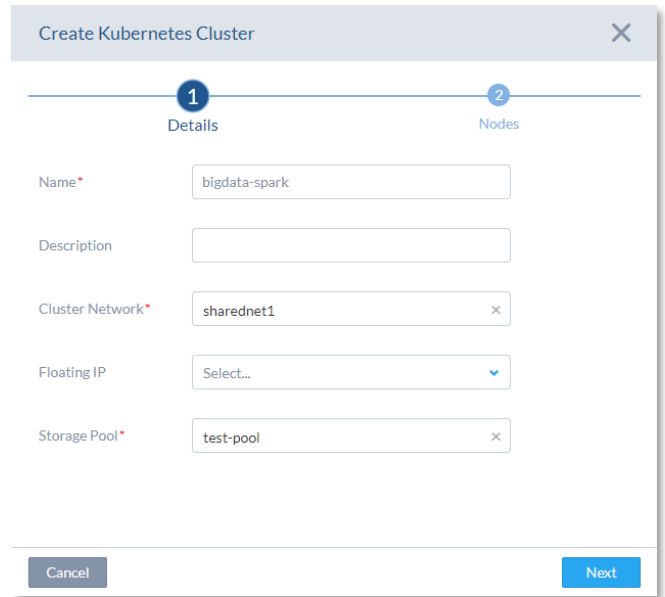
**Figure 6:** Selecting Containers service

The following screenshots illustrate the subsequent steps for configuring and deploying a Kubernetes cluster using the Stratoscale Symphony UI:

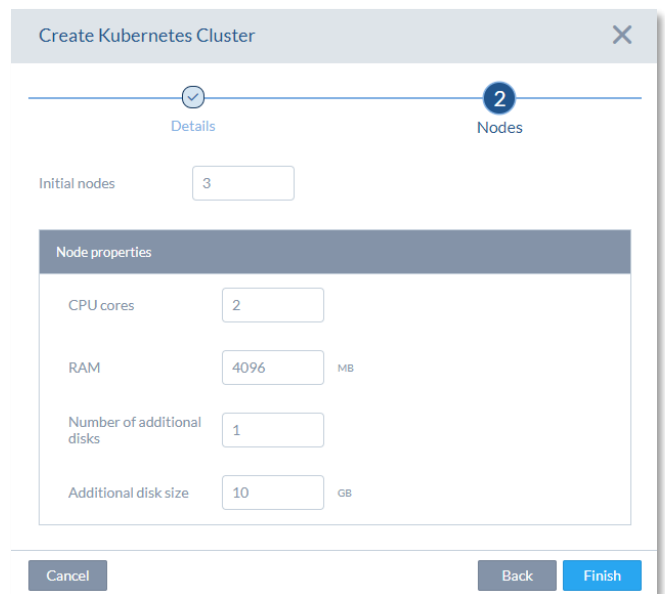
### Step 1: Select “Create Cluster”



### Step 2: Specify a cluster name, network, and storage pool



### Step 3: Specify the cluster size and resource requirements



Once the cluster has been launched, the status is updated live in the dashboard as nodes are allocated, provisioned, and configured to form a Kubernetes cluster. Once the state is “Running” as shown in Figure 7, additional details regarding the configuration can be obtained by clicking on the name link. The corresponding view as shown in Figure 8 provides the IP address for the cluster as well as a convenient link to open the native Kubernetes UI dashboard as shown in Figure 9.

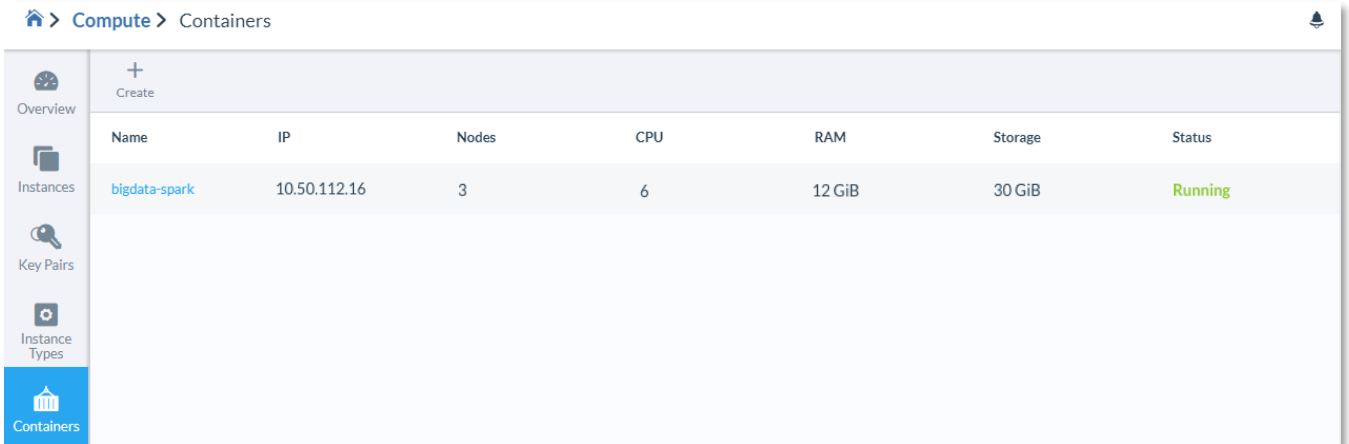


Figure 7: Overview of Kubernetes clusters and status

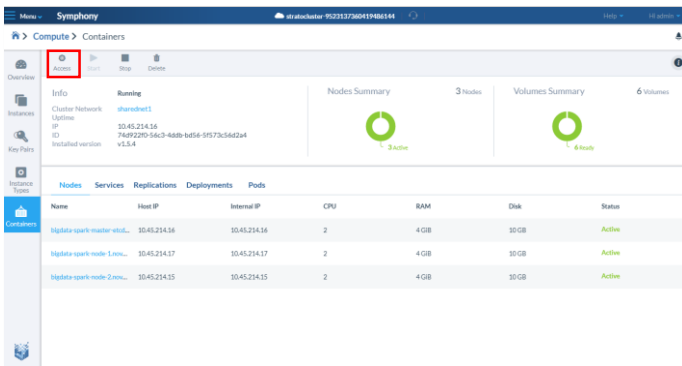


Figure 8: Kubernetes cluster detailed view and link to open native dashboard

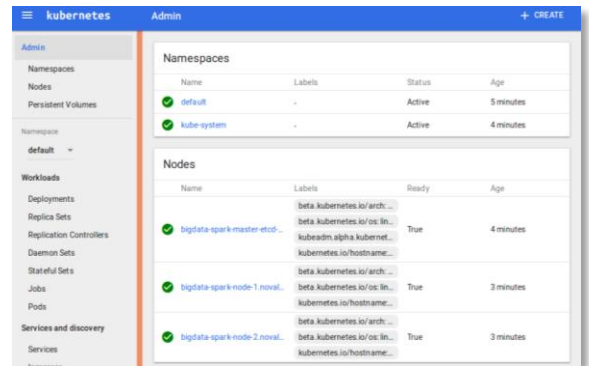


Figure 9: Kubernetes native dashboard

The Kubernetes command line tool KubeCTL can be installed <sup>[5]</sup> on the utility host and configured to point at the deployed cluster using the following commands:

```
kubectl config set-cluster bigdata --server=http://<CLUSTER_IP>:8080
kubectl config set-context default-context --cluster=bigdata
kubectl config use-context default-context
```

### 3. DEPLOYING A STANDALONE SPARK CLUSTER ON KUBERNETES

In our example, we deploy Spark 2.1.1 using the standalone deployment model supported by Spark [6]. In order to deploy Spark via Kubernetes, we define a Docker image as outlined in Appendix A.1. The image incorporates precompiled binaries for Spark as well as dependencies for AWS connectors. Our sample analyzer application is written in Python, and we therefore also include dependencies to support PySpark.

The Spark deployment on Kubernetes consists of the following components:

- A Spark master deployment (outlined in spark-master-deployment.yaml in Appendix A.2)
- A service definition so workers can reach the master (outlined in spark-master-service.yaml in Appendix A.2)
- A Spark worker deployment (outlined in spark-worker-deployment.yaml in Appendix A.2)

The components can be deployed using the following commands:

```
kubectl create -f spark-master-deployment.yaml
kubectl create -f spark-master-service.yaml
kubectl create -f spark-worker-deployment.yaml
```

The status of deployments and services can be confirmed using either kubectl or the Kubernetes UI as shown in Figures 10 and 11. In addition, the Spark master logs can be inspected to confirm that it is running successfully and receives connections from the workers as highlighted in Figure 12. At this point a functional Spark environment is available to execute Spark data processing applications.

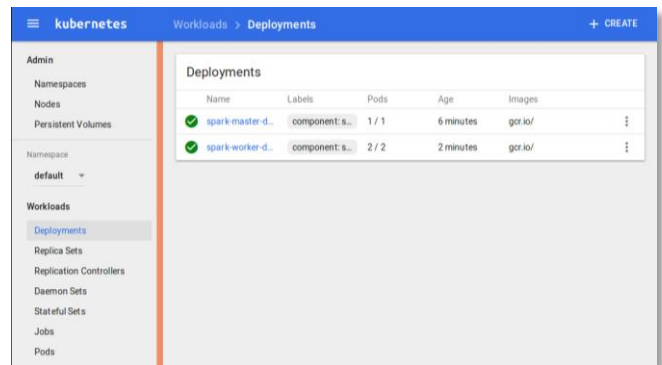


Figure 10: Kubernetes Spark master and slave deployments view

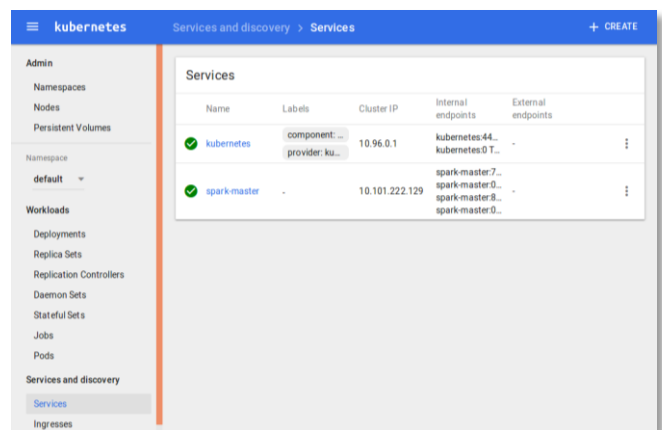
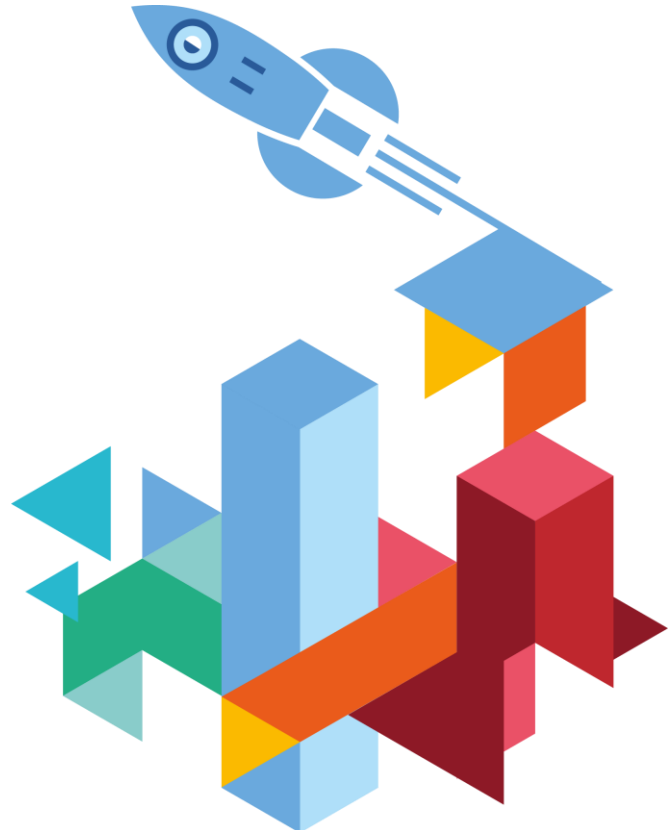


Figure 11: Kubernetes Spark master service view

Logs from spark-master in spark-master-deployment-2547274957-5m8jm

```
2017-07-20T16:13:22.323713000Z 17/07/20 16:13:22 INFO SecurityManager: Changing view
acls groups to:
2017-07-20T16:13:22.324549000Z 17/07/20 16:13:22 INFO SecurityManager: Changing modify
acls groups to:
2017-07-20T16:13:22.325489000Z 17/07/20 16:13:22 INFO SecurityManager:
SecurityManager: authentication disabled; ui acls disabled; users with view
permissions: Set(root); groups with view permissions: Set(); users with modify
permissions: Set(root); groups with modify permissions: Set()
2017-07-20T16:13:22.794693000Z 17/07/20 16:13:22 INFO Utils: Successfully started
service 'sparkMaster' on port 7077.
2017-07-20T16:13:23.119253000Z 17/07/20 16:13:23 INFO Master: Starting Spark master at
spark://spark-master-deployment-2547274957-5m8jm:7077
2017-07-20T16:13:23.123608000Z 17/07/20 16:13:23 INFO Master: Running Spark version
2.1.1
2017-07-20T16:13:23.504124000Z 17/07/20 16:13:23 INFO Utils: Successfully started
service 'MasterUI' on port 8080.
2017-07-20T16:13:23.507376000Z 17/07/20 16:13:23 INFO MasterWebUI: Bound MasterWebUI
to 0.0.0.0, and started at http://10.244.2.2:8080
2017-07-20T16:13:23.525531000Z 17/07/20 16:13:23 INFO Utils: Successfully started
service on port 6066.
2017-07-20T16:13:23.526705000Z 17/07/20 16:13:23 INFO StandaloneRestServer: Started
REST server for submitting applications on port 6066
2017-07-20T16:13:23.722709000Z 17/07/20 16:13:23 INFO Master: I have been elected
leader! New state: ALIVE
2017-07-20T16:15:51.583492000Z 17/07/20 16:15:51 INFO Master: Registering worker
10.244.2.3:38620 with 2 cores, 2.9 GB RAM
2017-07-20T16:17:45.080451000Z 17/07/20 16:17:45 INFO Master: Registering worker
10.244.1.2:34776 with 2 cores, 2.9 GB RAM
```

Figure 12: Spark master logs confirming launch and connection with workers



## 4. DEPLOYING A STANDALONE SPARK CLUSTER ON KUBERNETES

As an illustrative example, we define a simple Spark application that reads all text files from an S3 bucket and stores the number of total words to a MySQL table. The following code outlines this functionality as a PySpark application.

### spark-analyzer.py

```
from pyspark.sql import SparkSession
import MySQLdb
import os

SPARK_APP_NAME = "Spark analyzer"
DATABASE_HOST = os.environ.get('DATABASE_HOST')
DATABASE_USER = os.environ.get('DATABASE_USER')
DATABASE_PASSWORD = os.environ.get('DATABASE_PASSWORD')
DATABASE = 'test'

def store_result(val):
    """Store value to data table"""
    db = MySQLdb.connect(
        host=DATABASE_HOST,
        user=DATABASE_USER,
        passwd=DATABASE_PASSWORD,
        db=DATABASE
    )

    cursor = db.cursor()
    sql = "insert into data (value) VALUES(%s)" % (val)
    cursor.execute(sql)
    db.commit()
    db.close()

def main():
    """Main application method"""
    # Configure SparkSession
    spark = SparkSession \
        .builder \
        .appName(SPARK_APP_NAME) \
        .getOrCreate()
    s3Rdd = spark.sparkContext.textFile("s3a://<BUCKET_NAME>/*")
    answer = s3Rdd.flatMap(lambda line: line.split(" ")).count()
    store_result(answer)

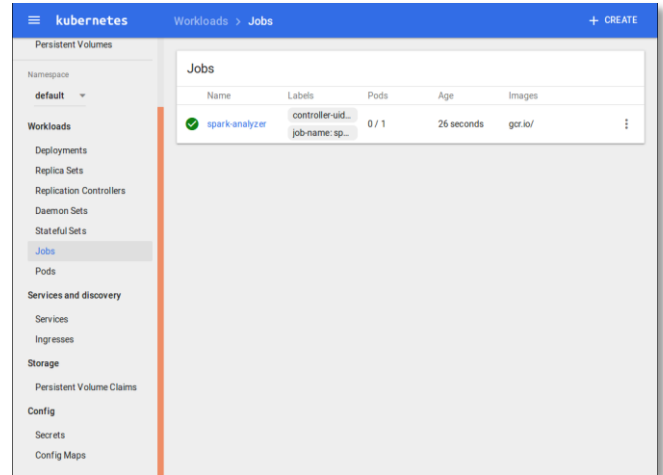
if __name__ == "__main__":
    main()
```

We execute this Spark application by defining it as a Kubernetes batch job so that the Spark driver can access the cluster and also to streamline the overall flow. The corresponding job configuration definition is outlined in Appendix A.3.

While some configuration values are hard-coded in this sample code, the application does expect certain information, including database details and AWS credentials for accessing S3, to be passed as environmental variables. These are defined as part of the job configuration which, in turn, makes use of Kubernetes support for secrets to securely store and pass credential data within the cluster. Representative secret configuration files can be found in Appendix A.4. The secret configuration and job execution can be invoked using the following commands:

```
kubectl create -f aws-secrets.yaml
kubectl create -f db-secrets.yaml
kubectl create -f spark-job-analyzer.yaml
```

The job status can be tracked in the Kubernetes dashboard as shown in Figure 13. The logs can also be inspected via UI or kubectl as needed. In our case, the successful run of the job results in the insertion of a value in our database as depicted in Figure 14.



**Figure 13:** Kubernetes view showing execution of Spark job

```
mysql> select value from data;
+-----+
| value |
+-----+
| 48    |
+-----+
1 row in set (0.01 sec)
```

**Figure 14:** Database query confirming successful execution of Spark application

# CONCLUSIONS

With the increasing emphasis on deriving business value from accumulated datasets, the demand to support big data initiatives in the enterprise is on a trajectory to grow rapidly over the coming years. The ability of an IT team to address the corresponding requirements of data science teams within the organization will be critical in serving this demand effectively. As demonstrated in this paper, the incorporation of key technologies can be a tremendous asset for IT managers by allowing them to enable developers and users to implement their use cases in a self-service manner while adhering to operational concerns and policies. Specifically, we've outlined how:

A hybrid cloud implementation comprised of AWS public cloud and AWS-compatible on-premises environments allows developers to

easily access and deploy resources as needed with tools they're already familiar with.

The use of resource managers such as Kubernetes allow data scientists to configure and execute analytics code written in their processing framework of choice in a composable manner.

Enabling dataflows that span public and private clouds can be an effective way to implement analytics while maintaining the data security needs of the organization.

These approaches demonstrate the benefits that proactive IT leaders can deliver while supporting big data projects, and thereby expand the role of IT as a strategic asset to their organizations.

# REFERENCES

[1] [aws.amazon.com](https://aws.amazon.com)

[2] [www.stratoscale.com](https://www.stratoscale.com)

[3] [spark.apache.org](https://spark.apache.org)

[4] [kubernetes.io](https://kubernetes.io)

[5] [kubernetes.io/docs/tasks/tools/install-kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl)

[6] [spark.apache.org/docs/latest/spark-standalone.html](https://spark.apache.org/docs/latest/spark-standalone.html)



# APPENDICES

## 1. SPARK CONTAINER IMAGE DOCKERFILE

```
FROM ubuntu:16.04

# Install base dependencies
RUN apt-get update && apt-get install -y \
    openjdk-8-jre-headless \
    curl \
    python-minimal \
    python-setuptools \
    python-dev \
    build-essential \
    libmysqlclient-dev

RUN easy_install pip
RUN pip install MySQL-python

# Get Spark from US Apache mirror
RUN mkdir -p /opt && \
    cd /opt && \
    curl http://www.us.apache.org/dist/spark/spark-2.1.1/spark-2.1.1-bin-
hadoop2.7.tgz | \
    tar -zx && \
    ln -s spark-2.1.1-bin-hadoop2.7 spark

# Download dependencies for AWS
RUN cd /opt/spark/jars && \
    curl -O http://central.maven.org/maven2/com/amazonaws/aws-java-
sdk/1.7.4/aws-java-sdk-1.7.4.jar && \
    curl -O http://central.maven.org/maven2/org/apache/hadoop/hadoop-
aws/2.7.3/hadoop-aws-2.7.3.jar

ENV PATH $PATH:/opt/spark/bin
```

## 2. SPARK DEPLOYMENT CONFIGURATION FILES

### spark-analyzer.py

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: spark-master-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      component: spark-master
  template:
    metadata:
      labels:
        component: spark-master
    spec:
      containers:
        - name: spark-master
          image: #INSERT IMAGE LOCATION
          command: ["/opt/spark/sbin/start-master.sh"]
          env:
            - name: SPARK_NO_DAEMONIZE
          ports:
            - containerPort: 7077
            - containerPort: 8080
          resources:
            requests:
              cpu: 100m
```

### spark-analyzer.py

```
kind: Service
apiVersion: v1
metadata:
  name: spark-master
spec:
  ports:
    - port: 7077
      targetPort: 7077
      name: spark
    - port: 8080
      targetPort: 8080
      name: http
  selector:
    component: spark-master
```

### spark-worker-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: spark-worker-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      component: spark-worker
  template:
    metadata:
      labels:
        component: spark-worker
    spec:
      containers:
        - name: spark-worker
          image: #INSERT IMAGE LOCATION
          command: ["/opt/spark/sbin/start-slave.sh"]
          args: ["spark://spark-master:7077"]
          env:
            - name: SPARK_NO_DAEMONIZE
          ports:
            - containerPort: 8081
          resources:
            requests:
              cpu: 100m
```

### 3. SPARK APPLICATION JOB CONFIGURATION FILE

```
apiVersion: batch/v1
kind: Job
metadata:
  name: spark-analyzer
spec:
  template:
    metadata:
      name: spark-analyzer
    spec:
      containers:
        - name: spark-analyzer
          image: #INSERT IMAGE LOCATION
          command: ["/opt/spark/bin/spark-submit"]
          args: [
            "--master",
            "spark://spark-master:7077",
            "/opt/app/spark-analyzer.py"
          ]
      env:
        - name: AWS_ACCESS_KEY_ID
          valueFrom:
            secretKeyRef:
              name: aws-credentials
              key: awsAccessKeyID
        - name: AWS_SECRET_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              name: aws-credentials
              key: awsSecretAccessKey
        - name: DATABASE_USER
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: username
        - name: DATABASE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: password
        - name: DATABASE_HOST
          value: #INSERT SQL hostname / IP
      restartPolicy: Never
```

## 4. SECRET CONFIGURATION FILES

### aws-secrets.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: aws-credentials
type: Opaque
data:
  awsAccessKeyID: #base64 encoded value
  awsSecretAccessKey: #base64 encoded value
```

### db-secrets.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: db-credentials
type: Opaque
data:
  username: #base64 encoded value
  password: #base64 encoded value
```



# Support Modern Applications with Cloud-Native Services



## About Stratoscale

Stratoscale delivers robust cloud building blocks to modernize and future-proof the enterprise on-prem environment, aligning it with the public cloud. Stratoscale's comprehensive software solution, enables enterprises to run and scale applications anywhere by transforming the data center into an agile, flexible and scalable environment. Driven by innovation and advanced cloud practices, Stratoscale provides development teams and IT the building blocks, managed services and tools to automate and simplify the entire life-cycle, ensure fast time-to-market and meet evolving business needs. Stratoscale raised over \$70M from leading investors including: Battery Ventures, Bessemer Venture Partners, Cisco, Intel, Qualcomm Ventures, SanDisk and Leslie Ventures.

For more information visit:

<http://www.stratoscale.com>

+1 877 420-3244 | [sales@stratoscale.com](mailto:sales@stratoscale.com)



AWS (Amazon Web Services) is a trademark of Amazon.com, Inc.