Working with Data at Scale in PySpark

Introduction

Presenters

- Sahil Jhangiani
  - Lead Architect at Nuna Inc
  - Head of Analytics at Team Liquid
- Sev Leonard
  - Senior Software Engineer at Fletch

Agenda

- Section 1: Cloud services overview
- Section 2: Building ingestion processes
- Section 3: Scaling and optimizing workloads
- Shutting down resources & billing console

Cloud Services

- Big three:
  - Amazon Web Services (AWS)
  - Google Cloud Platform (GCP)
  - Microsoft Azure
- Pros:
  - “Infinitely” scalable
  - Managed services reduce overall maintenance burden
  - Plug and play
- Cons:
  - “Infinitely” expensive
  - Less flexible (dependent on a few factors)
  - Subject to consistent change

Picking the Right Service

- Areas to evaluate:
  - Amount of data throughput/complexity of workloads
  - Future scaling/expected rate of change
  - Logic/ETL mobility
  - Existing workflow/needed interconnections/integrations

Amazon Web Services And Spark

- Spark
  - Open source
  - Premiere standard for big data processing
  - Allows for much more flexibility than most out of the box managed services
- Cluster computing
  - Horizontal scaling infrastructure
  - Challenges of distributed computing
- AWS
  - IAM roles and security groups
  - EMR
  - S3

Resource Setup

NOTE - Cloud compute costs ahead

- clone https://github.com/sjsloreilly/datascalepyspark
- https://console.aws.amazon.com/
- Select S3
- Navigate to data-scale-oreilly-{your name}/notebooks
- Upload bootstrap.sh

Resource Setup Continued

- Click the Services link in the upper left
- Under “Find Services” type EMR to go to the EMR console
- Clone the cluster you set up for class
- Add the bootstrap action:
    - s3://data-scale-oreilly-{your name}/notebooks/bootstrap.sh
- Click on “Notebooks” in the left sidebar
- Click on the notebook you set up and “Change cluster” to point to the cluster you just started. Start the notebook.
- When Open in JupyterLab is enabled, click to launch

Pre Class Poll

- Any issues with the pre class setup? (Group chat)
- Poll:
    - Familiarity with (1 to 5, 1 being the least familiar):
        - Python
        - Spark
        - AWS
    - Were you able to connect to JupyterHub? (True/False)

Introduction

Q&A

Working with Data at Scale in PySpark

Building a Big Data Pipeline

Section 2 Notebook

- Ingesting from an S3 endpoint
- Lab 2.1 - Ingesting taxi data
- Testing and Data Modeling
- Lab 2.2 - Expand taxi data ingest
- Handling corrupt data
- Lab 2.3 - Taxi zone lookup ingest
- Break
- Lab 2.4 - Case study 1
- Lab 2.5 - Case study 2
- Lab 2.6 - Full pipeline

Working with Data at Scale in PySpark

Break

Core Differences Between Pandas and PySpark

- APIs
- Backend processing
- Out of the box functionality
- Scalability
- Flexibility
- Koalas

Building a Big Data Pipeline

Q&A

Working with Data at Scale in PySpark

Break

Working with Data at Scale in PySpark

Scaling/ Optimizing Workloads

Section 3 Notebook

- Lab 3.1 - Leveraging file types
- Lab 3.2 - Partitioning
- Schema management
- Lab 3.3 - Case study 3
- Lab 3.4 - Writing data out to long term storage

File Types Overview

- Delimited (CSV, PSV, TSV, et al)
- JSON
- Sequence
- ORC
- AVRO
- Parquet
  - Footer metadata information
  - Run length encoding
  - Columnar store

Partitioning Methodologies

- Can be applied at multiple levels
- Repartition vs coalesce
- Field selection
- Workloads requirements/dealing with skew

Identifying Bottlenecks

- Monitoring Cluster Health
- Using the Spark UI
- Avoiding red herrings/effectively allocating your time optimizing

Writing Out Data One Last Time

- Long term storage vs intermediate workload
- EBS vs S3

Working with Data at Scale in PySpark

Break

Building a Big Data Pipeline

Q&A

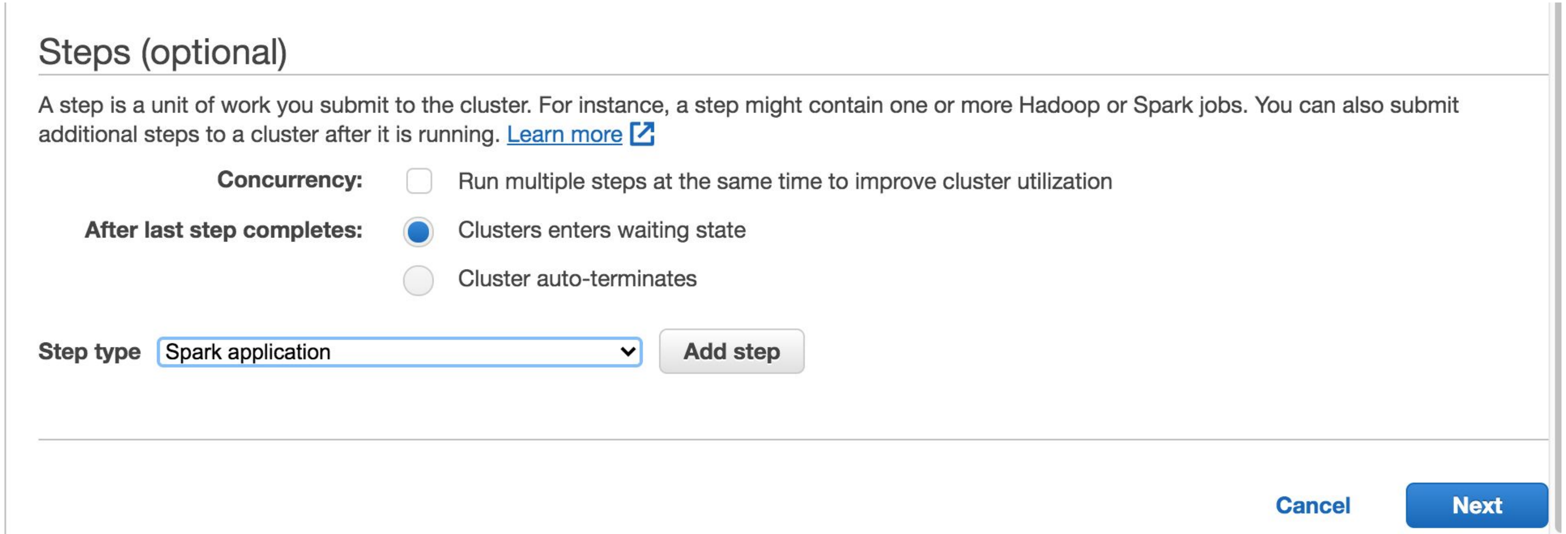
Creating an EMR Job

- Exporting code from Jupyter Hub (ingest.py)
- Creating EMR steps

Note: additional cost

Cluster setup

- Clone your Notebook cluster, including steps
- Under Step 1 “Software and Steps” scroll to the “Steps” area and add a step, choosing “Spark Application” for step type
- Click “Add step”



Steps (optional)

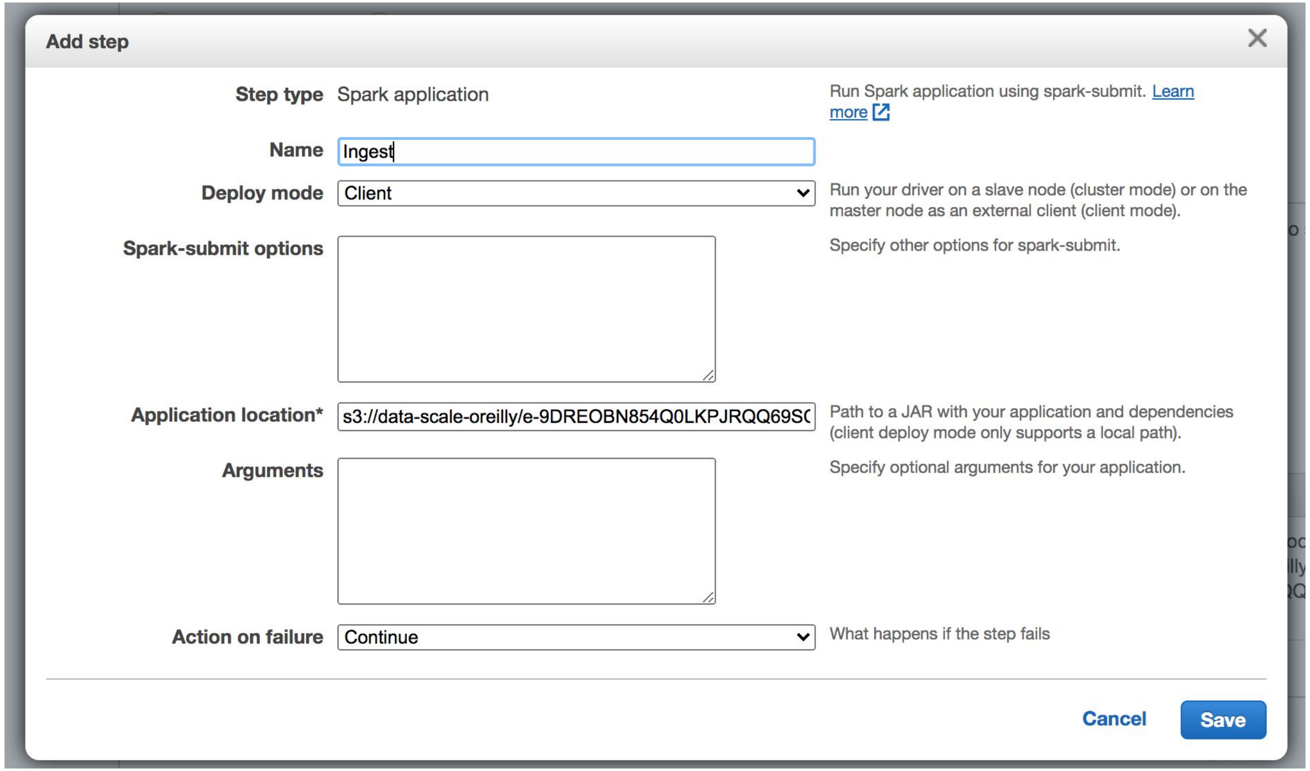
A step is a unit of work you submit to the cluster. For instance, a step might contain one or more Hadoop or Spark jobs. You can also submit additional steps to a cluster after it is running. Learn more

Concurrency: ☐ Run multiple steps at the same time to improve cluster utilization

After last step completes: [✓ Clusters enters waiting state] Cluster auto-terminates

Step type Spark application

Step setup - Ingest



Add step

Step type	Spark application	Run Spark application using spark-submit. Learn more
Name	Ingest	
Deploy mode	Client	Run your driver on a slave node (cluster mode) or on the master node as an external client (client mode).
Spark-submit options		Specify other options for spark-submit.
Application location*	s3://data-scale-oreilly/e-9DREOBN854Q0LKPJRQQ69SC	Path to a JAR with your application and dependencies (client deploy mode only supports a local path).
Arguments		Specify optional arguments for your application.
Action on failure	Continue	What happens if the step fails

Step setup - Case study

- As an exercise, export a case study to a .py file and create another step after the ingest

Hardware setup

Optional: This may be needed if you try to launch the cluster and receive an error due to vCPU limits

- Click “Next”
- Under Step 2 “Hardware” scroll to the “Cluster Nodes and Instances” area
- Change the Core instance type to m5.large and reduce the instance count to 1

Bootstrap setup

- If additional libraries are required, modify the bootstrap.sh file
- Click “Create Cluster”. The steps will run when the cluster spins up.

Scaling/ Optimizing Workloads

Q&A

Working with Data at Scale in PySpark

Spinning Down Resources/ Closing Remarks

Resource spin down walkthrough

- Saving work
- Terminating notebooks and cluster
- Navigating the AWS Billing console to monitor cost

Don’t forget to spin down your resources!