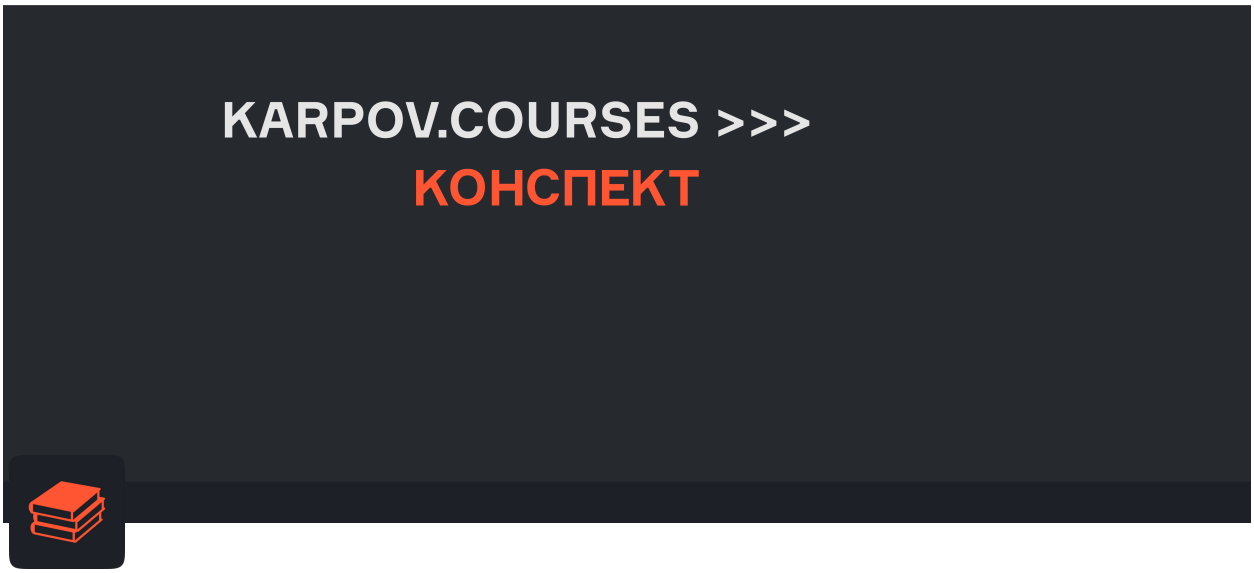


>Конспект > 8 урок > Отладка, профилирование и мониторинг Spark Job

> Оглавление

> Profiling

Профилирование - сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (wikipedia)

Spark написан на Scala -> все средства для профилирования JVM программ нам доступны

Сбор информации о выполнении обычно делают на уровне:

- ОС (top, vmstat, lsof, tcpdump, netstat, iostat и т.д.) - позволяют измерять нагрузку на жесткий диск, сеть.
- JVM (обычно делается с помощью agent)
- конкретной Spark job - Spark имеет UI, где каждая Spark job имеет свой интерфейс для отслеживания состояния.
- кластера (YARN или k8s) - самый высокоуровневый мониторинг

Profiling OS level

Достоинства

- честные метрики, привязанные к конкретному серверу
- родные для Linux утилиты (наличие документации)

Недостатки

- требуется установка всех необходимых утилит на все сервера/контейнеры.
- организация отправки метрик в единое хранилище
- зачастую трудно изолировать конкретный процесс (можно промерить общую нагрузку, включая другие работающие процессы)

Profiling JVM level

Достоинства

- родные для JVM средства (jconsole, visualvm, Java Flight Recorder и т.д.)
- очень подробная детализация (вплоть до вызова native кода)
- можно использовать для любых JVM программ

Недостатки

- требуется подключение к конкретным серверам и процессам (что очень затруднительно в кластерах)

- предварительно желательно локализовать проблему (для подключения к конкретному executor)
- организация сбора метрик и дампов памяти в единое хранилище (зачастую требуется много дискового пространства)
- можно "утонуть" в обилии информации, требуется опыт для эффективного анализа

Profiling Job level

Достоинства

- базовые метрики можно посмотреть как online в процессе выполнения Job, так и по завершении на History Server
- есть графовое представление выполняемых задач
- работает для любых Job вне зависимости от используемого ЯП

Недостатки

- не всегда доступно (закрытый production контур)
- гранулярность (job -> stage -> task)
- требуются дополнительные настройки инфраструктуры (log aggregation, history server и т.д.)

Profiling cluster level

Достоинства

- удобно для высокоуровнего анализа и мониторинга утилизации ресурсов
- централизация

Недостатки

- зачастую трудно отделить проблемы job от проблем инфраструктуры
- при гонке за ресурсы картина может искажаться
- очень трудно оценивать производительность задач при коммуникации с внешними системами (БД, S3/HDFS, Kafka и т.д.)

> Требования к идеальному профилированию

Что хочется?

1. Сбор метрик в едином хранилище

Для избежания проблем, когда, к примеру, на уровне JVM мы можем получить информацию вплоть до нативных вызовов, а на уровне кластера - только информацию о потреблении ресурсов Job.

2. Добавлять свои метрики

Spark имеет собственные метрики, связанные с потреблением ресурсов, характеристиками шафла, но мы можем хотеть наблюдать за бизнес-метриками. Например, вести подсчет обработки физических лиц, исключая юридические.

3. Включать/выключать метрики по необходимости

Желательно иметь простой и гибкий механизм для такого функционала, без привлечения разработчиков.

4. Подробная аналитика (группировка, временной отрезок, скользящие окна и т.д.)

5. Удобные средства визуализации

Удобнее наблюдать за состоянием на графиках, а не таблицах.

6. Минимальные изменения в коде (в идеале - без изменений)

Возможные решения

1. Сбор метрик в едином хранилище

Использовать push/pull доставку в time-series database (InfluxDB, Prometheus)

Обычные реляционные БД и большинство нереляционных БД не позволяют эффективно хранить и обрабатывать информацию, связанную с метриками и мониторингом. Для такого рода задач популярны time-series БД.

2. Добавлять свои метрики

Поддержка с Spark 3.0 (требуется дополнительный код, что логично)

3. Включать/выключать метрики по необходимости

Метрики необходимо группировать по приемнику (sink). Приемники можно включать/отключать на момент запуска Job.

4. Подробная аналитика (группировка, временной отрезок, скользящие окна и т.д.)

Time-Series DB поддерживает широкие аналитические возможности.

5. Удобные средства визуализации

Существуют легковесные средства визуализации, например, Grafana.

6. Минимальные изменения в коде (в идеале - без изменений)

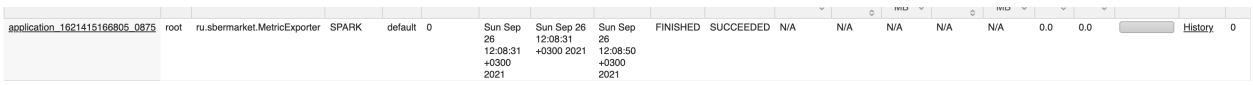
Настройка приемников (sink) доступна как параметр запуска, для отладки можно выводить в консоль, а на production - statsd, InfluxDB и т.д.

Подключение разного рода профайлеров производится через java agent (jar + параметры запуска)

> Обычный flow анализа

1. Смотрим на YARN Resource Manager UI

На уровне кластера можем посмотреть, какие Job и с каким статусом выполнялись.

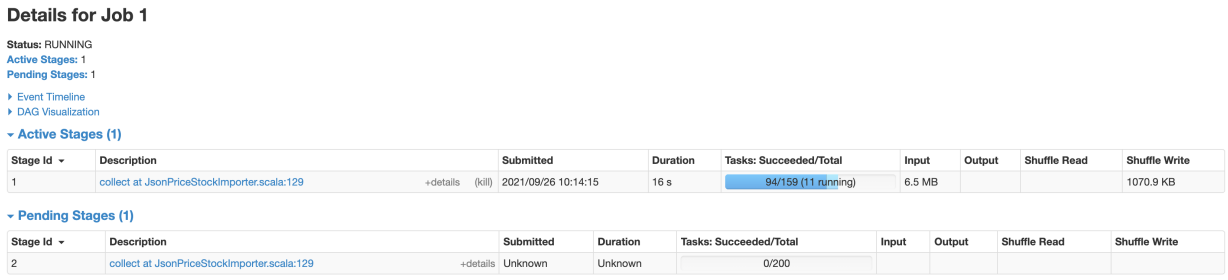


application_1621415166805_0875	root	ru.sbermarket.MetricExporter	SPARK	default	0	Sun Sep 26 12:08:31 +0300 2021	Sun Sep 26 12:08:31 +0300 2021	Sun Sep 26 12:08:50 +0300 2021	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0

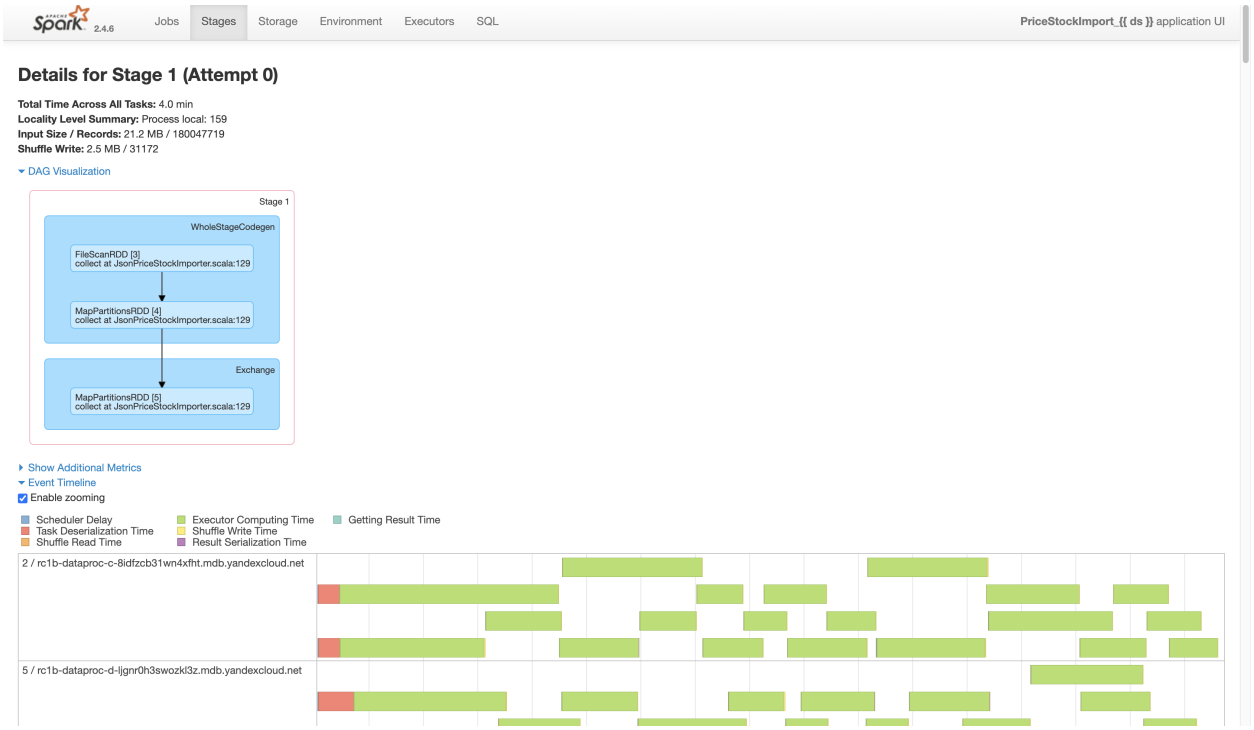
2. Заходим в History (Application Master, если задание еще работает) - для более детального анализа упавшей или подозрительной Job.

3. Смотрим на Jobs, выбираем подозрительную.

Для начала стоит обратить внимание на выделение ресурсов под конкретную Job. Например, открыть Event Timeline и посмотреть, в какой момент добавлялись executor. Могло случиться такое, что в момент запуска, ей не хватало ресурсов и она просто ждала их.



Перейдя на конкретную Job, мы можем смотреть из каких Stage (шагов) состоит выполнение терминального оператора.



По каждому Stage мы можем посмотреть из каких этапов он состоял, а также timeline (что происходит на каждом конкретном executor в конкретный момент времени).

Summary Metrics for 159 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0.3 s	1.0 s	1 s	2 s	5 s
Scheduler Delay	3 ms	5 ms	6 ms	7 ms	28 ms
Task Deserialization Time	0 ms	2 ms	2 ms	3 ms	0.7 s
GC Time	0 ms	0 ms	0 ms	8 ms	0.4 s
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	2 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	64.3 MB	64.3 MB	64.3 MB	64.3 MB	64.3 MB
Input Size / Records	42.2 KB / 225	55.8 KB / 705925	94.1 KB / 942892	127.4 KB / 1128508	1034.9 KB / 4098613
Shuffle Write Size / Records	1284.0 B / 9	6.9 KB / 49	12.6 KB / 101	18.8 KB / 166	55.6 KB / 1317

Aggregated Metrics by Executor

Executor ID ▲		Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Shuffle Write Size / Records	Blacklisted
1	stdout stderr	rc1b-dataproc-c-97cq7ba0wj61569n.mdb.yandexcloud.net:37670	49 s	33	0	0	33	4.3 MB / 38500945	534.1 KB / 6222	false
2	stdout stderr	rc1b-dataproc-c-8idfzcb31wn4xfht.mdb.yandexcloud.net:35373	48 s	32	0	0	32	4.0 MB / 39353280	499.5 KB / 6150	false
3	stdout stderr	rc1b-dataproc-c-e6oft4h554f686b7.mdb.yandexcloud.net:40540	50 s	33	0	0	33	4.4 MB / 34078689	518.8 KB / 6532	false
4	stdout stderr	rc1b-dataproc-d-e366rgs7pfnmjtf0.mdb.yandexcloud.net:33036	54 s	29	0	0	29	4.7 MB / 32331460	490.4 KB / 5976	false
5	stdout stderr	rc1b-dataproc-d-ljgnr0h3swozkl3z.mdb.yandexcloud.net:39406	48 s	32	0	0	32	3.9 MB / 35783345	508.8 KB / 6292	false

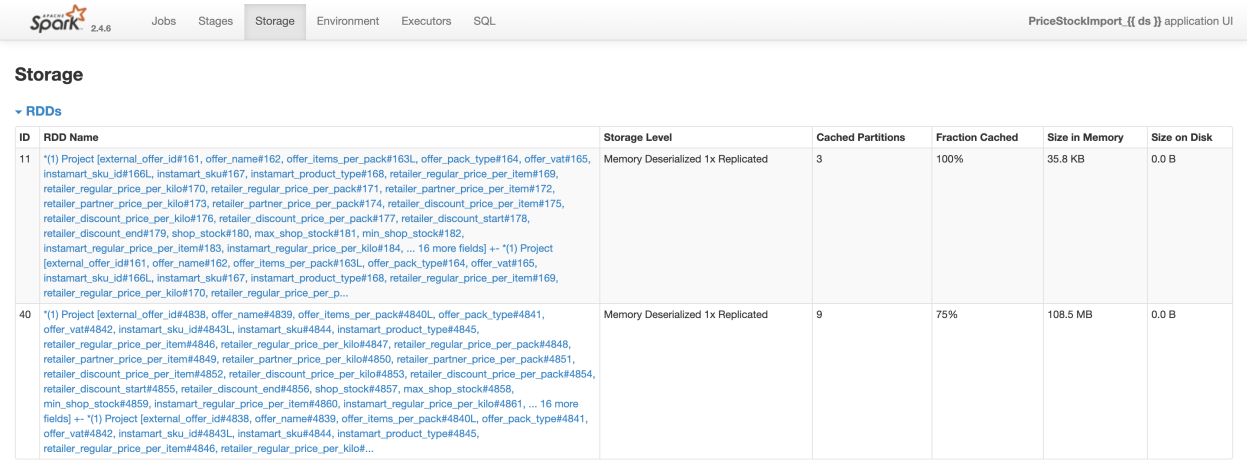
Tasks (159)

Page: 1 2 >

2 Pages. Jump to 1 . Show 100 items in a page. Go

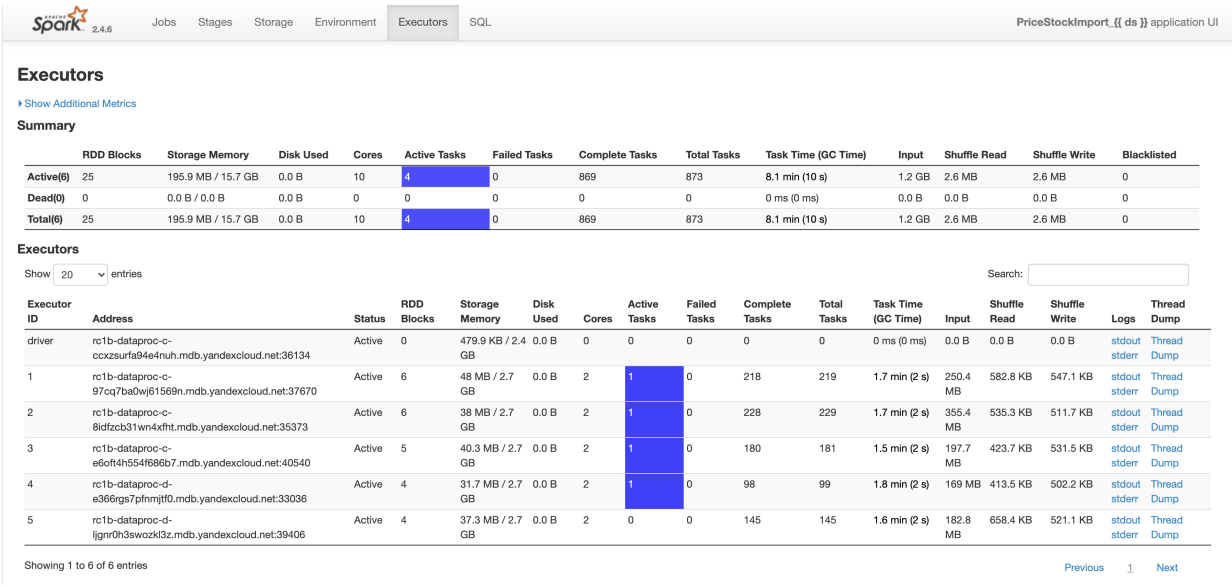
Index ▲	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Peak Execution Memory	Input Size / Records	Write Time	Shuffle Write Size / Records	Errors
0	66	0	SUCCESS	PROCESS_LOCAL	1	rc1b-dataproc-c-97cq7ba0wj61569n.mdb.yandexcloud.net stdout stderr	2021/09/26 10:14:15	4 s	18 ms	0.4 s		0 ms	0 ms	64.3 MB	78.3 KB / 1166024	19 ms	19.8 KB / 167	
1	67	0	SUCCESS	PROCESS_LOCAL	4	rc1b-dataproc-d-e366rgs7pfnmjtf0.mdb.yandexcloud.net	2021/09/26 10:14:15	4 s	17 ms	0.6 s		0 ms	0 ms	64.3 MB	75.8 KB / 1089468	16 ms	18.0 KB / 161	

По каждому Stage можем смотреть Tasks, которые в нем выполнялись. Здесь появляется возможность понять, есть ли выбросы по перцентилям и мониторить равномерное разбиение датафреймов. Также здесь показана агрегированная информация по executor.



ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
11	*(1) Project [external_offer_id#161, offer_name#162, offer_items_per_pack#163L, offer_pack_type#164, offer_vat#165, instamart_sku_id#166L, instamart_sku#167, instamart_product_type#168, retailer_regular_price_per_item#169, retailer_regular_price_per_kilo#170, retailer_regular_price_per_pack#171, retailer_partner_price_per_item#172, retailer_partner_price_per_kilo#173, retailer_partner_price_per_pack#174, retailer_discount_price_per_item#175, retailer_discount_price_per_kilo#176, retailer_discount_price_per_pack#177, retailer_discount_start#178, retailer_discount_end#179, shop_stock#180, max_shop_stock#181, min_shop_stock#182, instamart_regular_price_per_item#183, instamart_regular_price_per_kilo#184, ... 16 more fields] +- *(1) Project [external_offer_id#161, offer_name#162, offer_items_per_pack#163L, offer_pack_type#164, offer_vat#165, instamart_sku_id#166L, instamart_sku#167, instamart_product_type#168, retailer_regular_price_per_item#169, retailer_regular_price_per_kilo#170, retailer_regular_price_per_p...	Memory Deserialized 1x Replicated	3	100%	35.8 KB	0.0 B
40	*(1) Project [external_offer_id#4838, offer_name#4839, offer_items_per_pack#4840L, offer_pack_type#4841, offer_vat#4842, instamart_sku_id#4843L, instamart_sku#4844, instamart_product_type#4845, retailer_regular_price_per_item#4846, retailer_regular_price_per_kilo#4847, retailer_regular_price_per_pack#4848, retailer_partner_price_per_item#4849, retailer_partner_price_per_kilo#4850, retailer_partner_price_per_pack#4851, retailer_discount_price_per_item#4852, retailer_discount_price_per_kilo#4853, retailer_discount_price_per_pack#4854, retailer_discount_start#4855, retailer_discount_end#4856, shop_stock#4857, max_shop_stock#4858, min_shop_stock#4859, instamart_regular_price_per_item#4860, instamart_regular_price_per_kilo#4861, ... 16 more fields] +- *(1) Project [external_offer_id#4838, offer_name#4839, offer_items_per_pack#4840L, offer_pack_type#4841, offer_vat#4842, instamart_sku_id#4843L, instamart_sku#4844, instamart_product_type#4845, retailer_regular_price_per_item#4846, retailer_regular_price_per_kilo#...	Memory Deserialized 1x Replicated	9	75%	108.5 MB	0.0 B

Вкладка Storage нужна для понимания того, что сейчас находится в памяти.



	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(6)	25	195.9 MB / 15.7 GB	0.0 B	10	4	0	869	873	8.1 min (10 s)	1.2 GB	2.6 MB	2.6 MB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(6)	25	195.9 MB / 15.7 GB	0.0 B	10	4	0	869	873	8.1 min (10 s)	1.2 GB	2.6 MB	2.6 MB	0

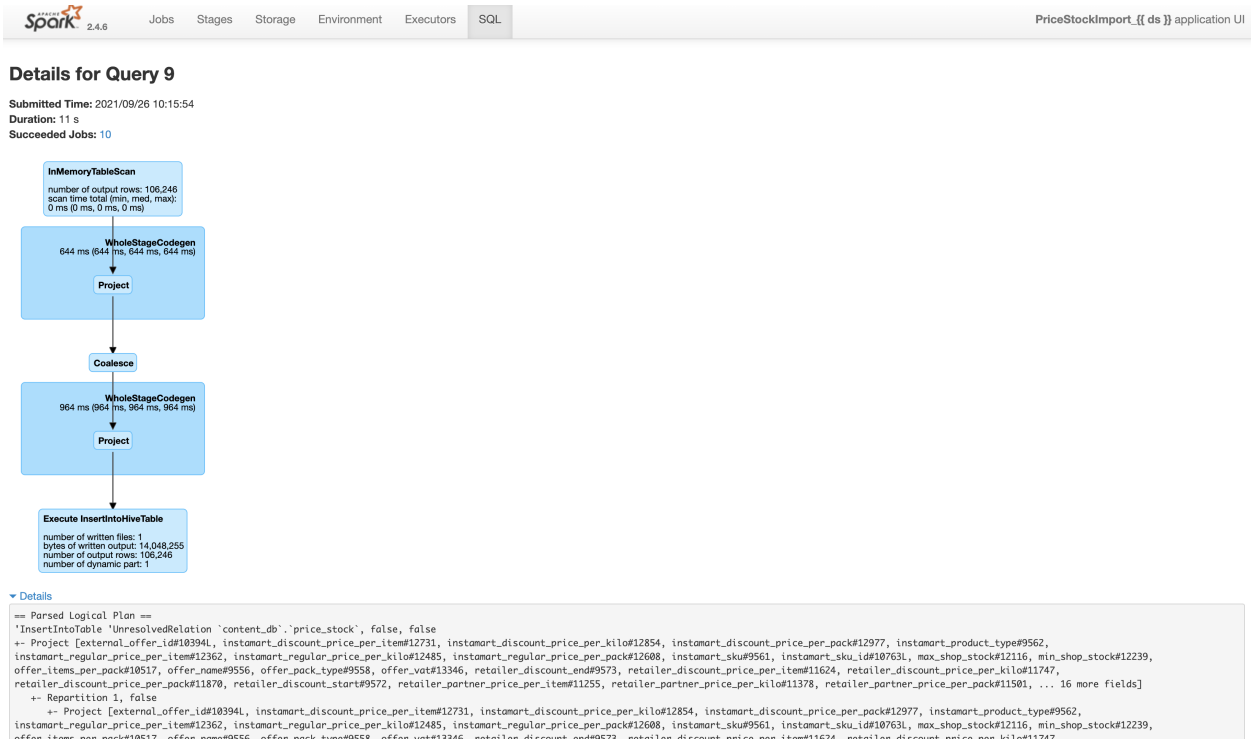
Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	rc1b-dataproc-c-ccxzsurfa94e4nuh.mdb.yandexcloud.net:36134	Active	0	479.9 KB / 2.4 GB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
1	rc1b-dataproc-c-97cq7ba0wj61569n.mdb.yandexcloud.net:37670	Active	6	48 MB / 2.7 GB	0.0 B	2	1	0	218	219	1.7 min (2 s)	250.4 MB	582.8 KB	547.1 KB	stdout stderr	Thread Dump
2	rc1b-dataproc-c-8idfzcb31wn4xfht.mdb.yandexcloud.net:35373	Active	6	38 MB / 2.7 GB	0.0 B	2	1	0	228	229	1.7 min (2 s)	355.4 MB	535.3 KB	511.7 KB	stdout stderr	Thread Dump
3	rc1b-dataproc-c-e6oft4h554f686b7.mdb.yandexcloud.net:40540	Active	5	40.3 MB / 2.7 GB	0.0 B	2	1	0	180	181	1.5 min (2 s)	197.7 MB	423.7 KB	531.5 KB	stdout stderr	Thread Dump
4	rc1b-dataproc-d-e366rgs7pfnmjtf0.mdb.yandexcloud.net:33036	Active	4	31.7 MB / 2.7 GB	0.0 B	2	1	0	98	99	1.8 min (2 s)	169 MB	413.5 KB	502.2 KB	stdout stderr	Thread Dump
5	rc1b-dataproc-d-ljgnr0h3swozkl3z.mdb.yandexcloud.net:39406	Active	4	37.3 MB / 2.7 GB	0.0 B	2	0	0	145	145	1.6 min (2 s)	182.8 MB	658.4 KB	521.1 KB	stdout stderr	Thread Dump

Showing 1 to 6 of 6 entries Previous 1 Next

Вкладка Executors дает информацию по всем executors в Job.



Вкладка SQL показывает план Catalyst (оптимизатора).

Подводя итог, можно сказать, что самым простым и в 90% случаев достаточным средством для мониторинга и профилирования ситуации в Job, является Spark UI, который внутри себя собирает информацию, как на уровне конкретных JVM процессов, так и на уровне заданий.