パフォーマンス チューニングのための監視パターンとメトリック

Databricks Log Analytics Monitor

開発チームは、可観測性パターンとメトリックを使用して、ボトルネックを見つけ、ビッグ データ システムのパフォーマンスを向上できます。 大規模なアプリケーションでは、大量のメトリック ストリームのロード テストを行う必要があります。

このシナリオでは、パフォーマンス チューニングに関するガイダンスを提供します。このシナリオでは、顧客ごとのログ記録に関するパフォーマンスの問題があるため、Azure Databricks を使用します。これにより、これらの項目を堅牢に監視できます。

- カスタム アプリケーション メトリック
- ストリーミング クエリ イベント
- アプリケーション ログ メッセージ

Azure Databricks では、この監視データをさまざまなログ記録サービス (Azure Log Analytics など) に送信できます。

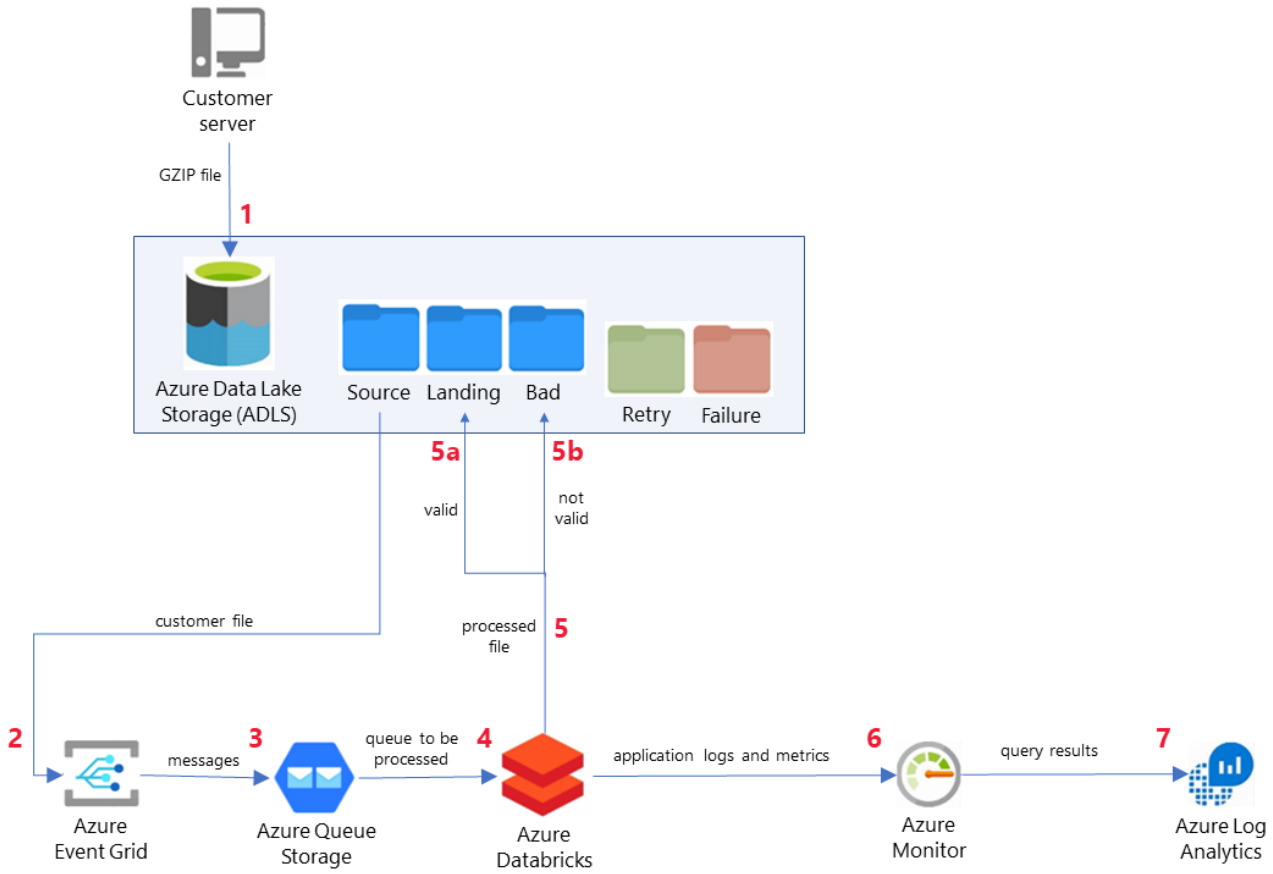
このシナリオでは、顧客ごとにグループ化され、GZIP アーカイブ ファイルに格納されている大量のデータを取り込む方法を説明します。 詳細ログは、リアルタイムの Apache Spark™ ユーザー インターフェイスの外部で Azure Databricks からは使用できません。そのため、チームは、顧客ごとにすべてのデータを格納し、ベンチマークと比較を行う方法を必要とします。 大規模なデータのシナリオでは、最速の処理時間に最適な組み合わせ Executor プールと仮想マシン (VM) のサイズを見つけることが重要です。 このビジネス シナリオでは、アプリケーション全体が取り込みとクエリの要件の速度に依存しているため、システムのスループットは、作業量が増加しても予期せず低下することはありません。 このシナリオでは、システムが顧客と共に確立されたサービス レベル アグリーメント (SLA) を満たしていることを保証する必要があります。

考えられるユース ケース

このソリューションによってメリットのあるシナリオは次のとおりです。

- システム正常性の監視。
- パフォーマンス メンテナンス。
- 日常的なシステムの使用状況の監視。
- 対処できない場合に、将来の問題の原因となる傾向を発見。

アーキテクチャ



このソリューションでは、次の手順を実行します。

1. サーバーは、顧客別にグループ化された大きな GZIP ファイルを Azure Data Lake Storage (ADLS) の **ソース** フォルダーに送信します。

2. その後、ADLS は、正常に抽出された顧客ファイルを Azure Event Grid に送信します。これにより、顧客のファイル データが複数のメッセージに変換されます。

3. Azure Event Grid は、メッセージを Azure Queue Storage サービスに送信し、キューに格納します。

4. Azure Queue Storage は、キューを Azure Databricks Data Analytics プラットフォームに送信して処理します。

5. Azure Databricks は、キュー データを、ADLS に送信して戻す処理済みのファイルにアンパックし、処理します。

   a. 処理されたファイルが有効な場合、**Landing** フォルダーに移動します。

   b. そうしないと、ファイルは **Bad** フォルダー ツリーに移動します。 最初に、ファイルは **Retry** サブフォルダーに移動し、ADLS は顧客のファイル処理を再度試みます (手順 2)。 再試行を 2 回実行しても有効でない処理済みのファイルを Azure Databricks が返す場合、処理されたファイルは **Failure** サブフォルダーに入ります。

6. Azure Databricks は、前の手順でデータをアンパックして処理するとき、アプリケーション ログとメトリックもストレージ用に Azure Monitor に送信します。

7. Azure Log Analytics ワークスペースは、アプリケーション ログに関する Kusto クエリと、トラブルシューティングおよびディープ診断のための Azure Monitor のメトリックを適用します。

コンポーネント

- Azure Data Lake Storage は、ビッグ データ分析専用の一連の機能です。
- Azure Event Grid を使うと、イベントベースのアーキテクチャを備えたアプリケーションを開発者が簡単にビルドできます。
- Azure Queue Storage は、多数のメッセージを格納するためのサービスです。 メッセージには、HTTP または HTTPS を使用して、認証された呼び出しを介して世界中のどこからでもアクセスできます。 キューを使用して、非同期的な処理用に作業のバックログを作成できます。
- Azure Databricks は、Azure クラウド サービス用に最適化された Data Analytics プラットフォームです。 データを集中的に使用するアプリケーションを開発するために Azure Databricks が提供する 2 つの環境のうちの 1 つは、大規模なデータ処理を行うための Apache Spark ベースの統合分析エンジンである Azure Databricks ワークスペースです。
- Azure Monitor によって、パフォーマンス メトリックやアクティビティ ログなどのアプリ テレメトリが収集され、分析されます。
- Azure Log Analytics は、データでログ クエリを編集および実行するツールです。

考慮事項

このアーキテクチャを検討するときは、以下の点に留意してください。

- Azure Databricks は、大規模なジョブに必要なコンピューティング リソースを自動的に割り当てることができるため、他のソリューションによって生じる問題を回避できます。 たとえば、Apache Spark で Databricks で最適化された自動スケールを使用すると、過剰なプロビジョニングによってリソースが最適に使用されなくなる可能性があります。 または、ジョブに必要な Executor の数がわからない場合もあります。

- Azure Queue Storage のキュー メッセージの許容される最大サイズは 64 KB です。 キューには、ストレージ アカウントの総容量の上限を超えない限り、数百万のキュー メッセージを含めることができます。

このシナリオのデプロイ

注意

ここで説明するデプロイ手順は、Azure Databricks、Azure Monitor、Azure Log Analytics にのみ適用されます。 この記事では、他のコンポーネントのデプロイについては説明しません。

プロセスのすべてのログと情報を取得するには、Azure Log Analytics と Azure Databricks 監視ライブラリを設定します。 監視ライブラリは、Apache Spark レベル イベントと Spark Structured Streaming メトリックをジョブから Azure Monitor にストリーミングします。 これらのイベントとメトリックのアプリケーション コードに変更を加える必要はありません。

ビッグ データ システムのパフォーマンス チューニングを設定する手順は次のとおりです。

1. Azure portal で Azure Databricks ワークスペースを作成します。 後で使用するために、Azure サブスクリプション ID (GUID)、リソース グループ名、Databricks ワークスペース名、およびワークスペース ポータル URL をコピーして保存します。

2. Web ブラウザーで、Databricks ワークスペースの URL にアクセスし、Databricks 個人用アクセス トークンを生成します。 表示されるトークン文字列を後で使用するためにコピーして保存します (文字列は `dapi` で始まる 32 文字の 16 進値です)。

3. mspnp/spark-monitoring GitHub リポジトリをローカル コンピューターに複製します。 このリポジトリには、次のコンポーネントのソース コードが含まれています。

    - Azure Log Analytics ワークスペースを作成するための Azure Resource Manager (ARM) テンプレート。Spark メトリックを収集するための事前に作成されたクエリもインストールされます。
    - Azure Databricks 監視ライブラリ
    - アプリケーション メトリックとアプリケーション ログを Azure Databricks から Azure Monitor に送信するサンプル アプリケーション

4. ARM テンプレートをデプロイするための Azure CLI コマンドを使用して、あらかじめ構築された Spark メトリック クエリで Azure Log Analytics ワークスペースを作成します。 コマンドの出力から、新しい Log Analytics ワークスペースの生成された名前をコピーして保存します (*spark-monitoring-<randomized-string>* の形式)。

5. Azure portal で Log Analytics ワークスペース ID とキーをコピーして保存し、後で使用できるようにします。

6. IntelliJ IDEA の Community Edition をインストールします。これは、Java 開発キット (JDK) と Apache Maven の組み込みサポートを備えた統合開発環境 (IDE) です。 Scala プラグインを追加します。

7. IntelliJ IDEA を使用して、Azure Databricks 監視ライブラリを構築します。 実際の構築手順を実行するには、 **[表示]** > **[ツール ウィンドウ]** > **Maven** を選択して Maven ツール ウィンドウを表示し、 **[Maven 目標を実行する]** > **[mvn パッケージ]** を選択します。

8. Python パッケージ インストール ツールを使用して Azure Databricks CLI をインストールし、前の手順でコピーした Databricks 個人用アクセス トークンを使用して認証を設定します。

9. 前の手順でコピーした Databricks と Log Analytics の値を使用して Databricks init スクリプトを変更して、Azure Databricks ワークスペースを構成し、Azure Databricks CLI を使用して

init スクリプトおよび Azure Databricks 監視ライブラリを Databricks ワークスペースにコピーします。

10. Databricks ワークスペース ポータルで、Azure Databricks クラスターを作成して構成します。

11. IntelliJ IDEA で、Maven を使用して サンプル アプリケーションを構築します。次に、Databricks ワークスペース ポータルでサンプル アプリケーションを実行して、Azure Monitor のサンプル ログとメトリックを生成します。

12. サンプル ジョブが Azure Databricks で実行されている間、Azure portal にアクセスして、Log Analytics インターフェイスでイベントの種類 (アプリケーション ログとメトリック) を表示およびクエリします。
    a. **[テーブル]** > **[カスタム ログ]** を選択して Spark リスナー イベント (**SparkListenerEvent_CL**) のテーブル スキーマ、Spark ログ イベント (**SparkLoggingEvent_CL**)、Spark メトリック (**SparkMetric_CL**) を表示します
    b. **[クエリ エクスプローラー]** > **[保存されたクエリ]** > **[Spark メトリック]** を選択し、Log Analytics ワークスペースを作成したときに追加されたクエリを表示して実行します。

    事前構築済みおよびカスタムのクエリを表示して実行する方法の詳細については、次のセクションをご覧ください。

Azure Log Analytics でログとメトリックをクエリする

事前構築済みクエリへのアクセス

Spark メトリックを取得するための事前構築済みクエリ名を次に示します。

- Executor ごとの CPU 時間 (%)
- Executor ごとの逆シリアル化時間 (%)
- Executor ごとの JVM 時間 (%)
- Executor ごとのシリアル化時間 (%)
- スピルされたディスク バイト数
- エラー トレース (不良レコードまたは不良ファイル)
- Executor ごとのファイル システム読み取りバイト数
- Executor ごとのファイル システム書き込みバイト数
- ジョブあたりのジョブ エラー数
- ジョブあたりのジョブ待機時間 (バッチ期間)
- ジョブ スループット
- 実行中の Executor
- シャッフル読み取りバイト数
- Executor ごとのシャッフル読み取りバイト数
- Executor ごとのディスクへのシャッフル読み取りバイト数
- シャッフル クライアント向けメモリ

- Executor ごとのシャッフル クライアント メモリ
- Executor ごとのスピルされたディスク バイト数
- Executor ごとのシャッフル ヒープ メモリ
- Executor ごとのスピルされたメモリ バイト数
- ステージごとのステージ待機時間 (ステージ期間)
- ステージごとのスループットステージ
- ストリームごとのストリーミング エラー
- ストリームごとのストリーミング待機時間
- ストリーミング スループット入力行数/秒
- ストリーミング スループット処理済み行数/秒
- ホストごとのタスクの合計実行数
- タスクの逆シリアル化時間
- ステージごとのタスク エラー数
- タスクの Executor コンピューティング時間 (データ スキュー時間)
- タスクの入力読み取りバイト数
- ステージごとのタスク待機時間 (タスクの期間)
- タスクの結果のシリアル化時間
- タスク スケジューラの遅延時間
- タスクのシャッフル読み取りバイト数
- タスクのシャッフル書き込みバイト数
- タスクのシャッフル読み取り時間
- タスクのシャッフル書き込み時間
- タスクのスループット (ステージごとのタスクの合計数)
- Executor ごとのタスク (Executor ごとのタスクの合計数)
- ステージごとのタスク数

カスタム クエリを書き込む

Kusto クエリ言語 (KQL) で独自のクエリを作成することもできます。 編集可能な上部中央のウィンドウを選択し、ニーズに合わせてクエリをカスタマイズするだけです。

次の 2 つのクエリは、Spark ログ イベントからデータを取得します。

Kusto コピー

```
SparkLoggingEvent_CL | where logger_name_s contains "com.microsoft.pnp"
```

Kusto コピー

```
SparkLoggingEvent_CL
| where TimeGenerated > ago(7d)
| project TimeGenerated, clusterName_s, logger_name_s
| summarize Count=count() by clusterName_s, logger_name_s, bin(TimeGenerated, 1h)
```

次の 2 つの例は、Spark メトリック ログに対するクエリです。

Kusto コピー

```
SparkMetric_CL
| where name_s contains "executor.cpuTime"
| extend sname = split(name_s, ".")
| extend executor=strcat(sname[0], ".", sname[1])
| project TimeGenerated, cpuTime=count_d / 100000
```

Kusto コピー

```
SparkMetric_CL
| where name_s contains "driver.jvm.total."
| where executorId_s == "driver"
| extend memUsed_GB = value_d / 1000000000
| project TimeGenerated, name_s, memUsed_GB
| summarize max(memUsed_GB) by tostring(name_s), bin(TimeGenerated, 1m)
```

クエリの用語

次の表では、アプリケーション ログとメトリックのクエリを作成するときに使用される用語について説明します。

期間	id	注釈
Cluster_init	Application ID	
キュー	Run ID (実行 ID)	1 つの実行 ID が複数のバッチに相当します。
Batch	Batch ID	1 つのバッチが 2 つのジョブに相当します。
ジョブ	[ジョブ ID]	1 つのジョブが 2 つのステージに相当します。
段階	ステージ ID	1 つの段階には、タスク (読み取り、シャッフル、または書き込み) に応じて 100-200 のタスク ID があります。
タスク	タスク ID	1 つのタスクが 1 つの Executor に割り当てられます。 1 つのタスクが 1 つのパーティションに対して `partitionBy` を実行するために割り当てられます。 約 200 人の顧客については、200 のタスクが必要です。

次のセクションには、システムのスループット、Spark ジョブの実行状態、システム リソースの使用状況を監視するために、このシナリオで使用される一般的なメトリックが含まれています。

システムのスループット

名前	Measurement	ユニット
ストリームのスループット	1 分あたりの平均処理レートでの平均入力レート	1 分ごとに 1 秒
ジョブ期間	1 分ごとの平均終了 Spark ジョブの期間	1 分ごとの期間
ジョブ数	1 分ごとの終了した Spark ジョブの平均数	1 分ごとのジョブ件数
ステージ期間	1 分ごとの完了した平均ステージ期間	1 分ごとの期間
ステージ数	1 分ごとの完了したステージの平均数	1 分ごとのステージ数
タスクの所要時間	1 分ごとの完了したタスクの平均期間	1 分ごとの期間
タスク数	1 分あたりの完了したタスクの平均数	1 分ごとのタスク数

Spark ジョブの実行状態

名前	Measurement	ユニット
スケジューラのプール数	1 分ごとのスケジューラ プールの個数 (動作しているキューの数)	スケジューラ プールの数
実行中の Executor の数	1 分ごとの実行中の Executor の数	実行中の Executor の数
エラー トレース	`Error` レベルおよび対応するタスク/ステージ ID (`thread_name_s` を参照) を持つすべてのエラー ログ	

システム リソースの使用状況

名前	Measurement	ユニット

名前	Measurement	ユニット
Executor ごとの平均 CPU 使用率/全体	1 分ごとの Executor ごとに使用された CPU の割合	1 分ごとの %
ホストごとの使用された平均ダイレクト メモリ (MB)	1 分ごとの各 Executor の使用された平均ダイレクト メモリ	1 分ごとの MB
ホストごとにスピルされたメモリ	Executor ごとのスピルされた平均メモリ	1 分ごとの MB
期間のデータ スキューの影響を監視する	タスクの期間に 70 - 90 パーセンタイルおよび 90 - 100 パーセンタイル の範囲と相違を測定する	100%、90%、70% の差分差。100%、90%、70% の差の割合

Azure Databricks は、バッチ操作全体を 1 ユニットとして処理するため、GZIP アーカイブファイルに結合された顧客の入力を特定の Azure Databricks 出力ファイルに関連付ける方法を決定します。 ここでは、トレースに粒度を適用します。 また、カスタム メトリックを使用して、1 つの出力ファイルを元の入力ファイルにトレースすることもできます。

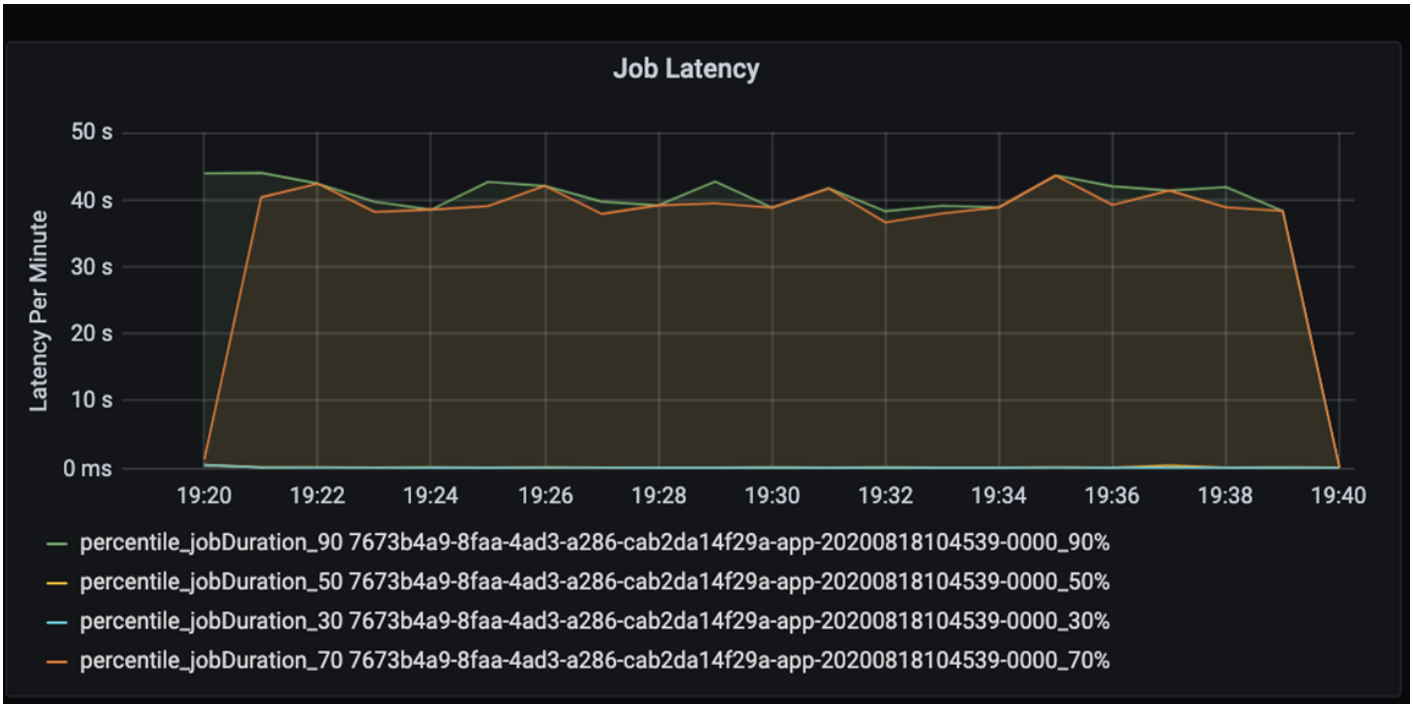
各メトリックの詳細な定義については、この Web サイトの「ダッシュ ボードでの視覚化」または Apache Spark のドキュメントの「メトリック」セクションをご覧ください。

パフォーマンス チューニング オプションの評価

ベースライン定義

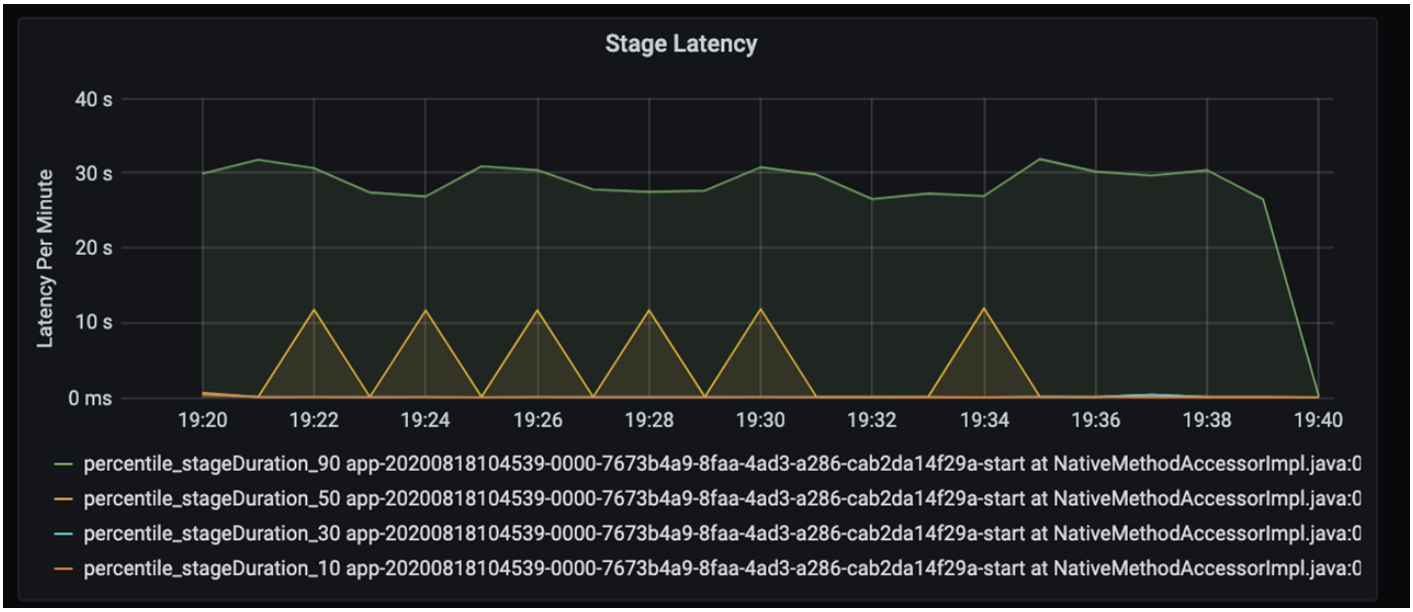
開発チームは、アプリケーションの将来の状態を比較できるように、ベースラインを確立する必要があります。

アプリケーションのパフォーマンスを定量的に測定します。 このシナリオでは、主要なメトリックはジョブの待機時間です。これは、ほとんどのデータの前処理とインジェストで一般的です。 次のグラフのように、データ処理時間を短縮し、待機時間の測定に専念します。



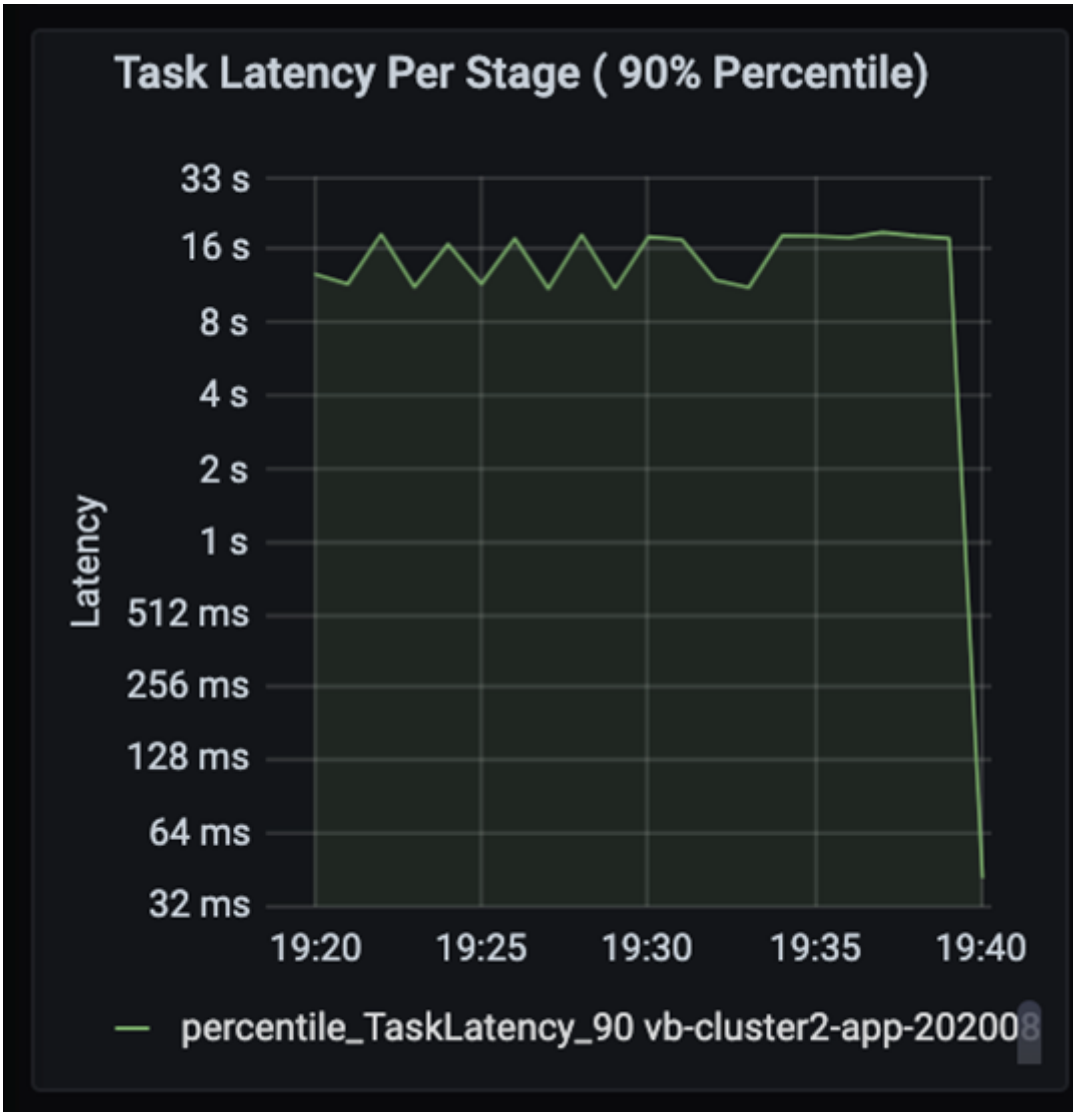
ジョブの実行待機時間の測定: ジョブの全体的なパフォーマンスに関する大まかなビューと、開始から完了 (マイクロバッチ時間) までのジョブ実行時間。 上の図では、19:30 マークでは、ジョブの処理に約 40 秒かかっています。

その 40 秒を詳しく確認する場合、ステージの下記データを表示します。



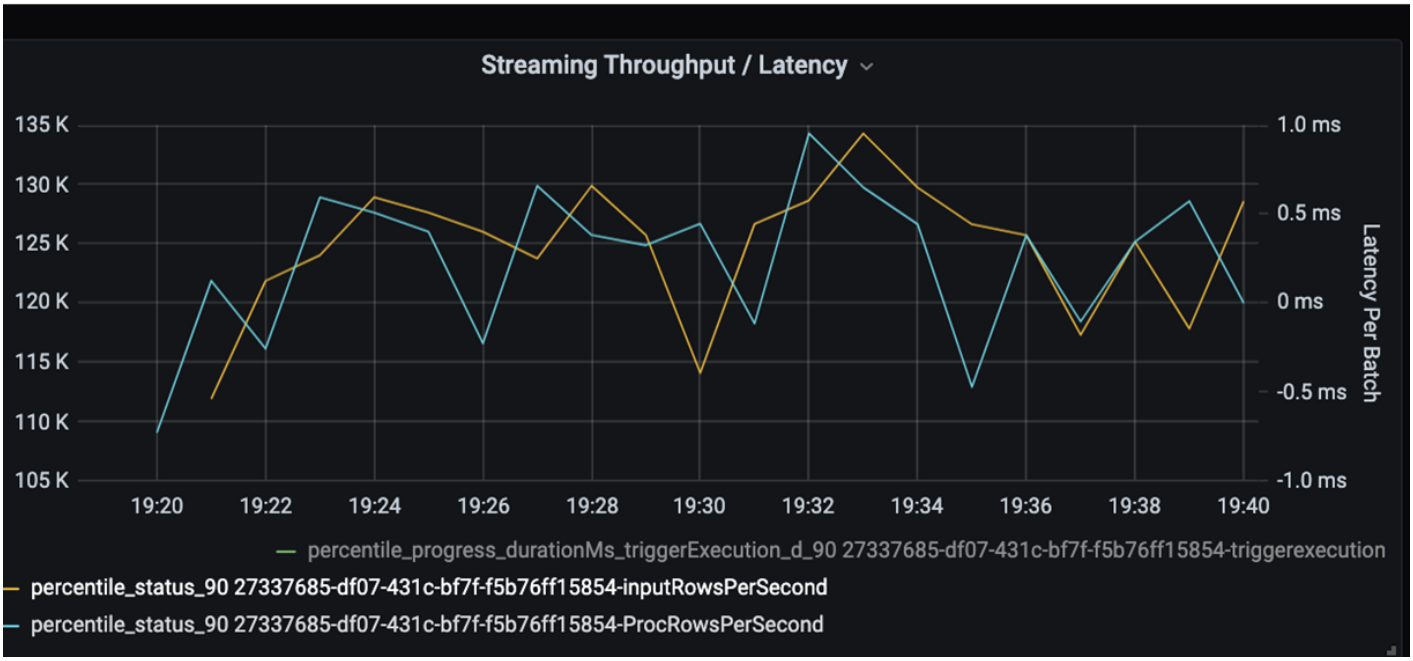
19:30 マークには、2 つのステージがあります。10 秒のオレンジのステージと 30 秒の緑のステージです。 スパイクはステージの遅延を示すため、ステージが急増しているかどうかを監視します。

特定のステージの実行速度がいつ遅いかを調査します。 パーティション分割シナリオでは、通常、少なくとも 2 つのステージがあります。1 つはファイルを読み取るステージ、もう 1 つはファイルのシャッフル、パーティション分割、書き込みを行うステージです。 待機時間が長いステージ (ほとんどの場合は書き込みステージ) がある場合、パーティション分割中にボトルネックの問題が発生した可能性があります。



前のステージが後のステージをブロックしている状態で、ジョブに含まれるステージが順次実行されているときにタスクを観察します。 ステージ内で 1 つのタスクが他のタスクよりも低速のシャッフル パーティションを実行する場合、ステージを完了するためには、クラスター内のすべてのタスクは、低速のタスクが完了するまで待機する必要があります。 タスクは、データ スキューや潜在的なボトルネックを監視する方法になります。 上のグラフでは、すべてのタスクが均等に分散されていることがわかります。

ここで、処理時間を監視します。 ストリーミング シナリオがあるため、ストリーミングのスループットを確認します。



上のストリーミングのスループット/バッチ待機時間のグラフでは、オレンジ色の線は入力速度 (1 秒あたりの入力行数) を表します。 青い線は処理速度 (1 秒あたりに処理された行数) を表します。 一部の時点では、処理速度が入力速度に追いついていません。潜在的な問題は、入力ファイルがキューに累積しています。

処理速度がグラフの入力速度と一致しないため、入力速度が完全にカバーされるように処理速度を上げることを検討してください。 考えられる原因の 1 つとして、ボトルネックにつながる各パーティション キーの顧客データの不均衡が考えられます。 次の手順と考えられる解決策として、Azure Databricks のスケーラビリティを活用します。

パーティション分割の調査

まず、Azure Databricks で必要な、適切な数のスケーリング Executor を特定します。実行中の Executor に専用の CPU を使用して各パーティションを割り当てるというルールを適用します。 たとえば、200 のパーティション キーがある場合、CPU の数と、Executor の数を乗算した値は 200 になります (たとえば、8 個の CPU と 25 個の Executor の組み合わせは適切です)。200 のパーティション キーでは、各 Executor は 1 つのタスクでのみ動作し、ボトルネックの機会を低減できます。

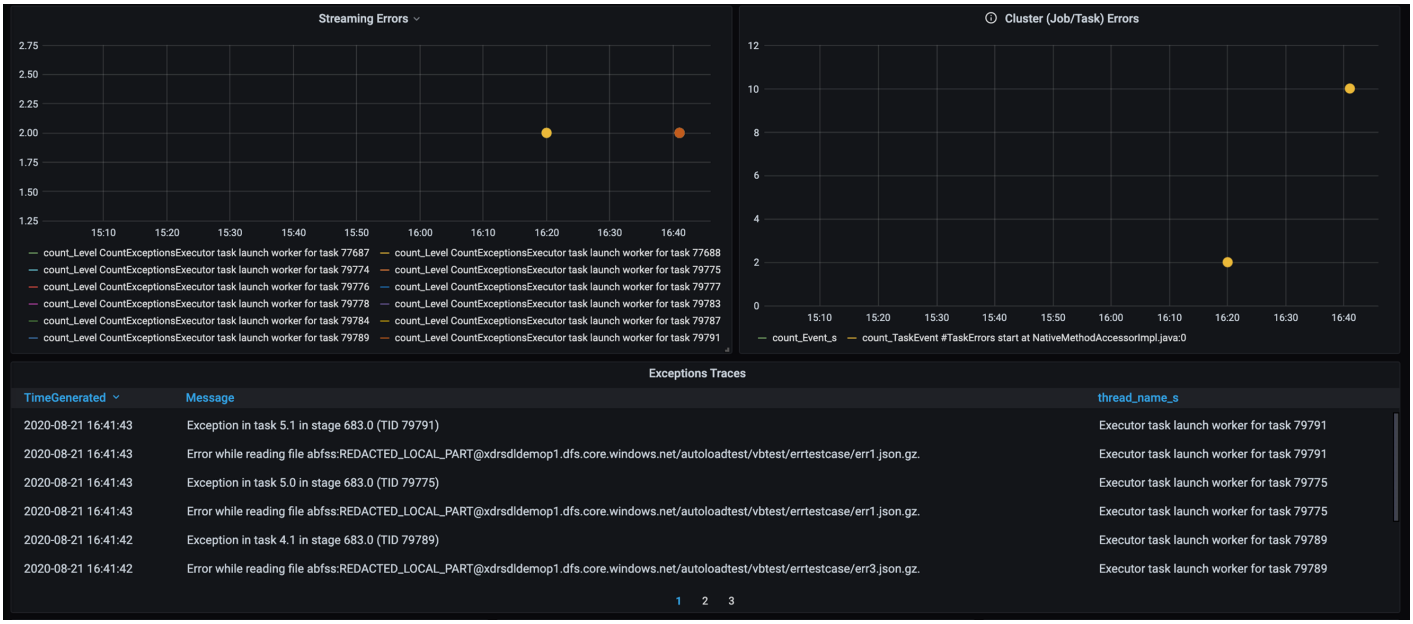
このシナリオでは、低速なパーティションがいくつかあるため、タスクの実行時間の高差異を調査します。 タスクの実行時間が急増していないか確認します。 1 つのタスクが 1 つのパーティションを処理します。 タスクの実行に時間がかかる場合、パーティションが大きすぎてボトルネックになる可能性があります。

Check Skew

TimeGenerated	slice	max_TaskLatency	avg_TaskLatency	min_TaskLatency	max_Stage_Info_Number_of_Tasks_d
2020-08-21 16:41:50	vb-cluster2-app-20200821054920-0...	2.8	0.71	0.012	200
2020-08-21 16:41:35	vb-cluster2-app-20200821054920-0...	15	6.6	0.16	13
2020-08-21 16:23:46	vb-cluster2-app-20200821054920-0...	28	5.8	0.012	200
2020-08-21 16:23:35	vb-cluster2-app-20200821054920-0...	11	10	9.4	60
2020-08-21 16:23:34	vb-cluster2-app-20200821054920-0...	0.057	0.030	0.014	60
2020-08-21 16:23:34	vb-cluster2-app-20200821054920-0...	0.019	0.015	0.010	60
2020-08-21 16:23:03	vb-cluster2-app-20200821054920-0...	30	7.8	0.0090	200
2020-08-21 16:22:52	vb-cluster2-app-20200821054920-0...	12	11	9.7	100
2020-08-21 16:22:52	vb-cluster2-app-20200821054920-0...	0.086	0.036	0.016	100

エラー トレース

顧客固有のデータ エラーを特定できるように、エラー トレースのダッシュボードを追加します。 データの前処理では、ファイルが破損していて、ファイル内のレコードがデータ スキーマと一致しない場合があります。 次のダッシュボードでは、多くの不良ファイルと不良レコードをキャッチします。



このダッシュボードには、デバッグのためにエラー数、エラー メッセージ、タスク ID が表示されます。 このメッセージでは、エラー ファイルにエラーを簡単にトレースできます。 読み取り中にエラーが発生したファイルがいくつかあります。 上のタイムラインを確認し、グラフ内の特定のポイント (16:20 と 16:40) を調査します。

その他のボトルネック

その他の例とガイダンスについては、「Azure Databricks でのパフォーマンスのボトルネックのトラブルシューティング」をご覧ください。

パフォーマンス チューニング評価の概要

このシナリオでは、これらのメトリックは次の観測値を識別しました。

- ステージ待機時間グラフでは、書き込みステージがほとんどの処理時間を占めています。
- タスクの待ち時間グラフでは、タスクの待機時間は安定しています。
- ストリーミング スループット グラフでは、いくつかの時点で出力速度が入力速度よりも遅くなっています。
- タスクの期間テーブルでは、顧客データの不均衡が原因でタスクの差異が発生しています。
- パーティション分割ステージで最適なパフォーマンスを得るには、スケーリング Executor 数がパーティションの数と一致している必要があります。
- 不良ファイルや不良レコードなどのトレース エラーが発生しています。

これらの問題を診断するために、次のメトリックを使用しました。

- ジョブの待機時間
- ステージの待機時間
- タスクの待機時間
- ストリーミングのスループット
- ステージあたりのタスクの実行時間 (最大、平均、最小)
- エラー トレース (数、メッセージ、タスク ID)

価格

このソリューションの実装コストを見積もるには、Azure 料金計算ツールを使用します。

次のステップ

Log Analytics のチュートリアルをご覧ください。