Confluent Kafka のベストプラクティス

NetApp artificial intelligence solutions

NetApp
February 12, 2026

目次

Confluent Kafka のベストプラクティス

TR-4912: NetAppを使用した Confluent Kafka 階層型ストレージのベストプラクティスガイドライン

Karthikeyan Nagalingam、JosephKantilparambil、 NetApp Rankesh Kumar、Confluent

Apache Kafka は、1 日に数兆件のイベントを処理できるコミュニティ分散型のイベント ストリーミング プラットフォームです。当初はメッセージング キューとして考案された Kafka は、分散コミット ログの抽象化に基づいています。 Kafka は 2011 年に LinkedIn によって作成され、オープンソース化されて以来、メッセージ キューから本格的なイベント ストリーミング プラットフォームへと進化してきました。 Confluent は、Confluent Platform を通じて Apache Kafka のディストリビューションを提供します。 Confluent Platform は、大規模な運用におけるオペレーターと開発者の両方のストリーミング エクスペリエンスを強化するように設計された追加のコミュニティ機能と商用機能を Kafka に補完します。

このドキュメントでは、次の内容を提供して、NetApp のオブジェクト ストレージ サービスで Confluent Tiered Storage を使用するためのベスト プラクティス ガイドラインについて説明します。

- NetAppオブジェクトストレージによる合流性検証 – NetApp StorageGRID
- 階層型ストレージのパフォーマンステスト
- NetAppストレージ システム上の Confluent に関するベスト プラクティス ガイドライン

Confluent 階層型ストレージを選ぶ理由

Confluent は、特にビッグ データ、分析、ストリーミング ワークロードなど、多くのアプリケーションのデフォルトのリアルタイム ストリーミング プラットフォームになっています。階層型ストレージを使用すると、ユーザーは Confluent プラットフォームでコンピューティングとストレージを分離できます。これにより、データの保存にかかるコスト効率が向上し、事実上無制限の量のデータを保存して、オンデマンドでワークロードをスケールアップ (またはスケールダウン) できるようになり、データやテナントの再調整などの管理タスクが容易になります。 S3 互換ストレージ システムは、これらすべての機能を活用して、すべてのイベントを 1 か所に集めてデータを民主化し、複雑なデータ エンジニアリングの必要性を排除できます。 Kafka に階層型ストレージを使用する理由の詳細については、以下を参照してください。"Confluentによるこの記事" 。

NetApp instaclustr は、3.8.1 から階層型ストレージを備えた Kafka もサポートしています。詳細はこちらをご覧ください "Kafka 階層型ストレージを使用した Instaclust"

階層型ストレージにNetApp StorageGRID を選ぶ理由

StorageGRIDは、 NetAppによる業界をリードするオブジェクト ストレージ プラットフォームです。 StorageGRID は、Amazon Simple Storage Service (S3) API を含む業界標準のオブジェクト API をサポートする、ソフトウェア定義のオブジェクトベースのストレージ ソリューションです。 StorageGRID は、大規模に非構造化データを保存および管理し、安全で耐久性のあるオブジェクト ストレージを提供します。コンテンツは適切な場所、適切な時間、適切なストレージ層に配置され、ワークフローが最適化され、グローバルに分散されたリッチ メディアのコストが削減されます。

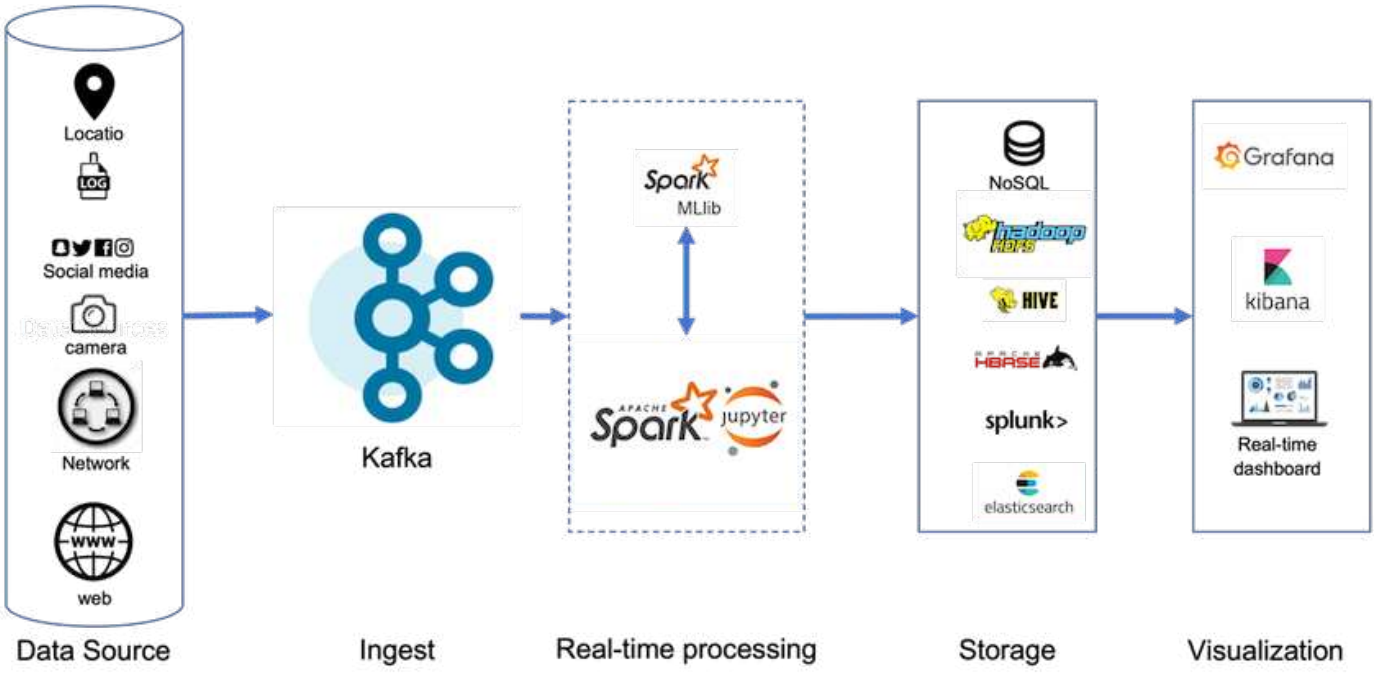
StorageGRIDの最大の差別化要因は、ポリシー主導のデータ ライフサイクル管理を可能にする情報ライフサイクル管理 (ILM) ポリシー エンジンです。ポリシー エンジンはメタデータを使用して、データの存続期間全体にわたってデータの保存方法を管理し、最初にパフォーマンスを最適化し、データが古くなるにつれてコストと耐久性を自動的に最適化します。

Confluent階層化ストレージの有効化

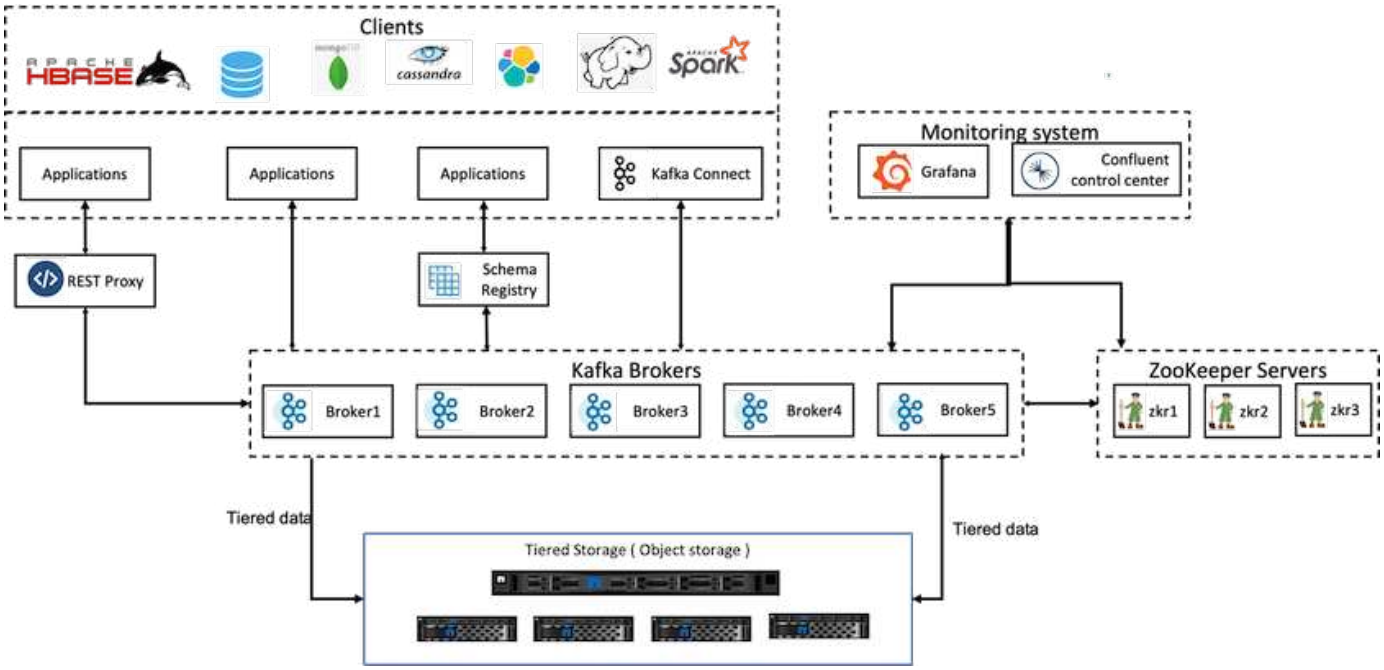
階層型ストレージの基本的な考え方は、データ保存のタスクとデータ処理のタスクを分離することです。この分離により、データ ストレージ層とデータ処理層を個別に拡張することがはるかに容易になります。

Confluent の階層型ストレージ ソリューションは、2 つの要素に対処する必要があります。まず、LIST 操作の不整合や、時々発生するオブジェクトの利用不可など、一般的なオブジェクト ストアの一貫性と可用性の特性を回避または回避する必要があります。次に、ゾンビ リーダーがオフセット範囲を階層化し続ける可能性を含め、階層化ストレージと Kafka のレプリケーションおよびフォールト トレランス モデル間のやり取りを正しく処理する必要があります。 NetAppオブジェクト ストレージは、一貫したオブジェクト可用性と HA モデルの両方を提供し、疲れたストレージを階層オフセット範囲で利用できるようにします。 NetAppオブジェクト ストレージは、一貫したオブジェクト可用性と HA モデルを提供し、疲れたストレージを階層オフセット範囲で利用できるようにします。

階層型ストレージを使用すると、ストリーミング データの末尾付近の低レイテンシの読み取りと書き込みに高性能プラットフォームを使用できるほか、高スループットの履歴読み取りにNetApp StorageGRIDなどの安価でスケーラブルなオブジェクト ストアを使用することもできます。また、NetApp ストレージ コントローラを使用した Spark 向けの技術ソリューションも用意しており、詳細はこちらをご覧ください。次の図は、Kafka がリアルタイム分析パイプラインにどのように適合するかを示しています。



次の図は、 NetApp StorageGRID がConfluent Kafka のオブジェクト ストレージ層としてどのように適合するかを示しています。



ソリューションアーキテクチャの詳細

このセクションでは、Confluent 検証に使用されるハードウェアとソフトウェアについて説明します。この情報は、 NetAppストレージを使用した Confluent Platform の展開に適用されます。次の表は、テストされたソリューション アーキテクチャと基本コンポーネントを示しています。

ソリューションコンポーネント	詳細
Confluent Kafka バージョン 6.2	• 3人の動物園飼育員 • 5つのブローカーサーバー • 5つのツールサーバー • グラファナ1個 • 1つのコントロールセンター
Linux（Ubuntu 18.04）	すべてのサーバー
階層型ストレージ向けNetApp StorageGRID	• StorageGRIDソフトウェア • 1 x SG1000（ロードバランサー） • 4 x SGF6024 • 4 x 24 x 800 SSD • S3プロトコル • 4 x 100GbE (ブローカーとStorageGRIDインスタンス間のネットワーク接続)

ソリューションコンポーネント	詳細
富士通 PRIMERGY RX2540 サーバ 15 台	それぞれに以下の機能が搭載されています: * 2つのCPU、合計16個の物理コア * Intel Xeon * 256GBの物理メモリ * 100GbEデュアルポート

技術概要

このセクションでは、このソリューションで使用されるテクノロジについて説明します。

NetAppStorageGRID

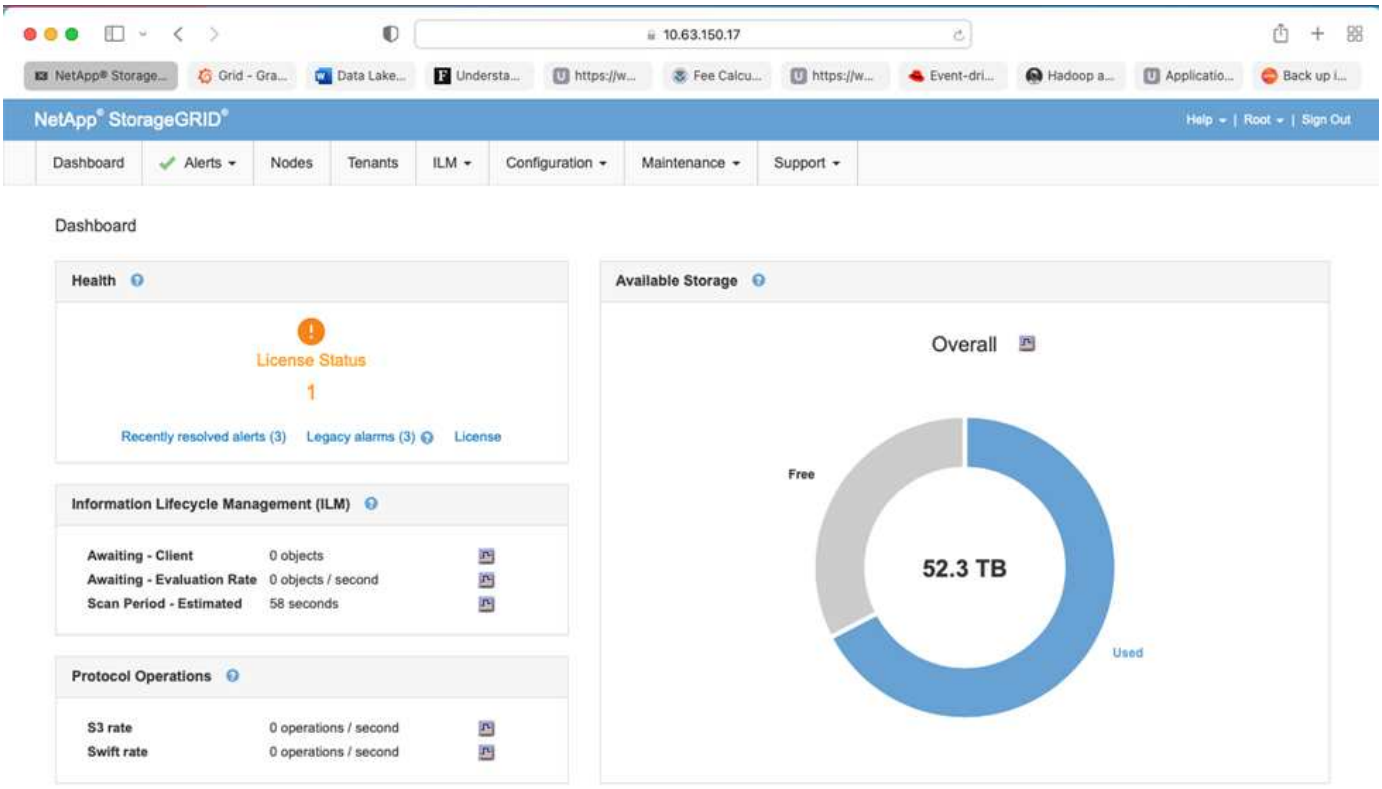
NetApp StorageGRIDは、高性能でコスト効率に優れたオブジェクト ストレージ プラットフォームです。階層型ストレージを使用すると、ローカル ストレージまたはブローカーの SAN ストレージに保存されている Confluent Kafka 上のデータの大部分が、リモート オブジェクト ストアにオフロードされます。この構成により、クラスターの再調整、拡張、縮小、または障害が発生したブローカーの交換にかかる時間とコストが削減され、運用が大幅に改善されます。オブジェクト ストレージは、オブジェクト ストア層に存在するデータの管理において重要な役割を果たすため、適切なオブジェクト ストレージを選択することが重要です。

StorageGRID は、分散型のノードベースのグリッド アーキテクチャを使用して、インテリジェントなポリシー主導のグローバル データ管理を提供します。ユビキタスなグローバル オブジェクト名前空間と洗練されたデータ管理機能を組み合わせることで、ペタバイト単位の非構造化データと数十億個のオブジェクトの管理を簡素化します。シングルコールのオブジェクト アクセスはサイト全体に拡張され、高可用性アーキテクチャを簡素化するとともに、サイトまたはインフラストラクチャの停止に関係なく継続的なオブジェクト アクセスを保証します。

マルチテナンシーにより、複数の非構造化クラウドおよびエンタープライズ データ アプリケーションを同じグリッド内で安全に処理できるようになり、 NetApp StorageGRIDの ROI と使用事例が増加します。メタデータ駆動型のオブジェクト ライフサイクル ポリシーを使用して複数のサービス レベルを作成し、複数の地域にわたって耐久性、保護、パフォーマンス、および局所性を最適化できます。ユーザーは、常に変化する IT 環境で要件が変わったときに、データ管理ポリシーを調整し、トラフィック制限を監視および適用して、中断なくデータ ランドスケープに再調整することができます。

グリッドマネージャーによるシンプルな管理

StorageGRID Grid Manager は、ブラウザベースのグラフィカル インターフェイスであり、世界中に分散された場所にあるStorageGRIDシステムを単一の画面で構成、管理、監視できます。



StorageGRID Grid Manager インターフェイスを使用して、次のタスクを実行できます。

- 画像、ビデオ、レコードなどのオブジェクトの、グローバルに分散されたペタバイト規模のリポジトリを管理します。
- オブジェクトの可用性を確保するためにグリッド ノードとサービスを監視します。
- 情報ライフサイクル管理 (ILM) ルールを使用して、時間の経過に伴うオブジェクト データの配置を管理します。これらのルールは、オブジェクトのデータが取り込まれた後に何が起こるか、どのように損失から保護されるか、オブジェクト データがどこに保存されるか、どのくらいの期間保存されるかを制御します。
- システム内のトランザクション、パフォーマンス、および操作を監視します。

情報ライフサイクル管理ポリシー

StorageGRIDには、特定のパフォーマンスとデータ保護の要件に応じて、オブジェクトのレプリカ コピーを保持したり、2+1 や 4+2 などの EC (消去コーディング) スキームを使用してオブジェクトを保存したりするなど、柔軟なデータ管理ポリシーがあります。ワークロードと要件は時間の経過とともに変化するため、ILM ポリシーも時間の経過とともに変更する必要があるのが一般的です。 ILM ポリシーの変更は中核機能であり、これによりStorageGRID のお客様は変化し続ける環境に迅速かつ簡単に適応できます。

パフォーマンス

StorageGRIDは、VM、ベアメタル、または専用アプライアンスなどのストレージノードを追加することでパフォーマンスを拡張します。"SG5712、SG5760、SG6060、またはSGF6024" 。当社のテストでは、SGF6024 アプライアンスを使用した最小サイズの 3 ノード グリッドで Apache Kafka の主要なパフォーマンス要件を超えました。顧客が追加のブローカーを使用して Kafka クラスターを拡張すると、ストレージノードを追加してパフォーマンスと容量を向上させることができます。

ロードバランサとエンドポイントの構成

StorageGRIDの管理ノードは、 StorageGRIDシステムを表示、構成、管理するための Grid Manager UI (ユーザー インターフェイス) と REST API エンドポイント、およびシステム アクティビティを追跡するための監査ログを提供します。 Confluent Kafka 階層型ストレージに高可用性の S3 エンドポイントを提供するために、管理ノードとゲートウェイ ノードでサービスとして実行されるStorageGRIDロード バランサーを実装しました。さらに、ロード バランサはローカル トラフィックも管理し、GSLB (グローバル サーバー負荷分散) と通信して災害復旧を支援します。

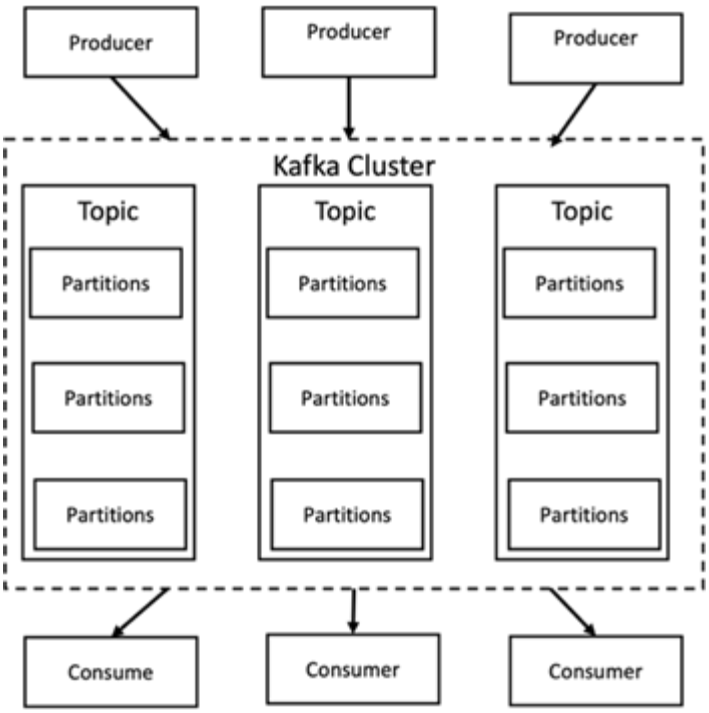
エンドポイント構成をさらに強化するために、 StorageGRID は管理ノードに組み込まれたトラフィック分類ポリシーを提供し、ワークロード トラフィックを監視し、ワークロードにさまざまなサービス品質 (QoS) 制限を適用できるようにします。トラフィック分類ポリシーは、ゲートウェイ ノードと管理ノードのStorageGRIDロード バランサ サービスのエンドポイントに適用されます。これらのポリシーは、トラフィックのシェーピングと監視に役立ちます。

StorageGRIDにおけるトラフィック分類

StorageGRIDには QoS 機能が組み込まれています。トラフィック分類ポリシーは、クライアント アプリケーションから送信されるさまざまな種類の S3 トラフィックを監視するのに役立ちます。次に、入力/出力帯域幅、読み取り/書き込み同時要求の数、または読み取り/書き込み要求レートに基づいてこのトラフィックに制限を設定するポリシーを作成して適用できます。

Apache Kafka

Apache Kafka は、Java と Scala で記述されたストリーム処理を使用したソフトウェア バスのフレームワーク実装です。リアルタイムのデータフィードを処理するための、統合された高スループット、低レイテンシのプラットフォームを提供することを目的としています。 Kafka は、Kafka Connect を介してデータのエクスポートとインポートのために外部システムに接続でき、Java ストリーム処理ライブラリである Kafka ストリームを提供します。 Kafka は、効率性を重視して最適化されたバイナリ TCP ベースのプロトコルを使用し、メッセージを自然にグループ化してネットワーク ラウンドトリップのオーバーヘッドを削減する「メッセージセット」抽象化に依存しています。これにより、大規模な順次ディスク操作、大規模なネットワーク パケット、連続したメモリ ブロックが可能になり、Kafka はランダム メッセージ書き込みのバースト ストリームを線形書き込みに変換できるようになります。次の図は、Apache Kafka の基本的なデータ フローを示しています。



Kafka は、プロデューサーと呼ばれる任意の数のプロセスから送信されるキーと値のメッセージを保存します。データは、異なるトピック内の異なるパーティションに分割できます。パーティション内では、メッセージはオフセット (パーティション内のメッセージの位置) によって厳密に順序付けられ、タイムスタンプとともにインデックスが付けられて保存されます。コンシューマーと呼ばれる他のプロセスは、パーティションからメッセージを読み取ることができます。ストリーム処理の場合、Kafka は、Kafka からデータを消費し、結果を Kafka に書き戻す Java アプリケーションを作成できる Streams API を提供します。 Apache Kafka は、Apache Apex、Apache Flink、Apache Spark、Apache Storm、Apache NiFi などの外部ストリーム処理システムとも連携します。

Kafka は 1 つ以上のサーバー (ブローカーと呼ばれる) のクラスター上で実行され、すべてのトピックのパーティションはクラスター ノード全体に分散されます。さらに、パーティションは複数のブローカーに複製されます。このアーキテクチャにより、Kafka はフォールト トレラントな方法で大量のメッセージ ストリームを配信できるようになり、Java Message Service (JMS)、Advanced Message Queuing Protocol (AMQP) などの従来のメッセージング システムの一部を置き換えることが可能になりました。 0.11.0.0 リリース以降、Kafka は、Streams API を使用して 1 回だけのストリーム処理を提供するトランザクション書き込みを提供します。

Kafka は、通常のトピックと圧縮されたトピックの 2 種類のトピックをサポートしています。通常のトピックは、保持時間またはスペースの制限付きで構成できます。指定された保持期間よりも古いレコードがある場合、またはパーティションのスペース制限を超えた場合、Kafka は古いデータを削除してストレージスペースを解放できます。デフォルトでは、トピックの保持期間は 7 日間に設定されていますが、データを無期限に保存することもできます。圧縮されたトピックの場合、レコードは時間または領域の境界に基づいて期限切れになることはありません。代わりに、Kafka は後続のメッセージを同じキーを持つ古いメッセージの更新として扱い、キーごとに最新のメッセージを削除しないことを保証します。ユーザーは、特定のキーに null 値を設定したいわゆるトゥームストーン メッセージを書き込むことで、メッセージを完全に削除できます。

Kafka には 5 つの主要な API があります。

- *プロデューサー API。*アプリケーションがレコードのストリームを公開することを許可します。
- *コンシューマー API*アプリケーションがトピックをサブスクライブし、レコードのストリームを処理することを許可します。

- *コネクタ API。*トピックを既存のアプリケーションにリンクできる再利用可能なプロデューサー API とコンシューマー API を実行します。
- *ストリーム API。*この API は入力ストリームを出力に変換し、結果を生成します。
- 管理 **API**。 Kafka トピック、ブローカー、およびその他の Kafka オブジェクトを管理するために使用されます。

コンシューマー API とプロデューサー API は、Kafka メッセージング プロトコルの上に構築され、Java での Kafka コンシューマー クライアントとプロデューサー クライアントのリファレンス実装を提供します。基礎となるメッセージング プロトコルはバイナリ プロトコルであり、開発者はこれを使用して任意のプログラミング言語で独自のコンシューマー クライアントまたはプロデューサー クライアントを作成できます。これにより、Kafka は Java 仮想マシン (JVM) エコシステムから解放されます。利用可能な非 Java クライアントのリストは、Apache Kafka wiki で管理されています。

Apache Kafka のユースケース

Apache Kafka は、メッセージング、Web サイトのアクティビティ追跡、メトリック、ログ集約、ストリーム処理、イベント ソーシング、コミット ログで最も人気があります。

- Kafka はスループット、組み込みのパーティショニング、レプリケーション、フォールト トレランスが向上しており、大規模なメッセージ処理アプリケーションに適したソリューションとなっています。
- Kafka は、追跡パイプライン内のユーザーのアクティビティ (ページ ビュー、検索) を、リアルタイムのパブリッシュ/サブスクライブ フィードのセットとして再構築できます。
- Kafka は運用監視データによく使用されます。これには、分散アプリケーションからの統計を集約して、運用データの集中フィードを生成することが含まれます。
- 多くの人が、ログ集約ソリューションの代わりとして Kafka を使用しています。ログ集約では通常、サーバーから物理ログ ファイルが収集され、処理のために中央の場所 (ファイル サーバーや HDFS など) に配置されます。 Kafka はファイルの詳細を抽象化し、ログまたはイベント データをメッセージ ストリームとしてよりクリーンに抽象化します。これにより、処理のレイテンシが低減され、複数のデータ ソースと分散データ消費のサポートが容易になります。
- Kafka の多くのユーザーは、複数のステージで構成される処理パイプラインでデータを処理します。このパイプラインでは、生の入力データが Kafka トピックから消費され、その後、さらなる消費や後続処理のために、集約、拡充、またはその他の方法で新しいトピックに変換されます。たとえば、ニュース記事を推奨するための処理パイプラインでは、RSS フィードから記事のコンテンツをクロールし、「記事」トピックに公開する場合があります。さらに処理を進めると、このコンテンツが正規化または重複排除され、クリーンアップされた記事コンテンツが新しいトピックに公開され、最終処理段階でこのコンテンツをユーザーに推奨しようとする可能性があります。このような処理パイプラインは、個々のトピックに基づいてリアルタイムのデータフローのグラフを作成します。
- イベント サウシングは、状態の変化が時間順のレコードのシーケンスとして記録されるアプリケーション設計のスタイルです。 Kafka は非常に大きなログ データの保存をサポートしているため、このスタイルで構築されたアプリケーションにとって優れたバックエンドになります。
- Kafka は、分散システムの一種の外部コミット ログとして機能します。ログはノード間でデータを複製するのに役立ち、障害が発生したノードがデータを復元するための再同期メカニズムとして機能します。 Kafka のログ圧縮機能は、このユースケースのサポートに役立ちます。

合流

Confluent Platform は、アプリケーション開発と接続の高速化、ストリーム処理による変換の実現、大規模なエンタープライズ運用の簡素化、厳格なアーキテクチャ要件への対応を支援するように設計された高度な機能で Kafka を補完する、エンタープライズ対応のプラットフォームです。 Apache Kafka のオリジナル作成者に

よって構築された Confluent は、エンタープライズ グレードの機能によって Kafka の利点を拡大するとともに、Kafka の管理や監視の負担を軽減します。現在、Fortune 100 企業の 80% 以上がデータ ストリーミング テクノロジーを活用しており、そのほとんどが Confluent を使用しています。

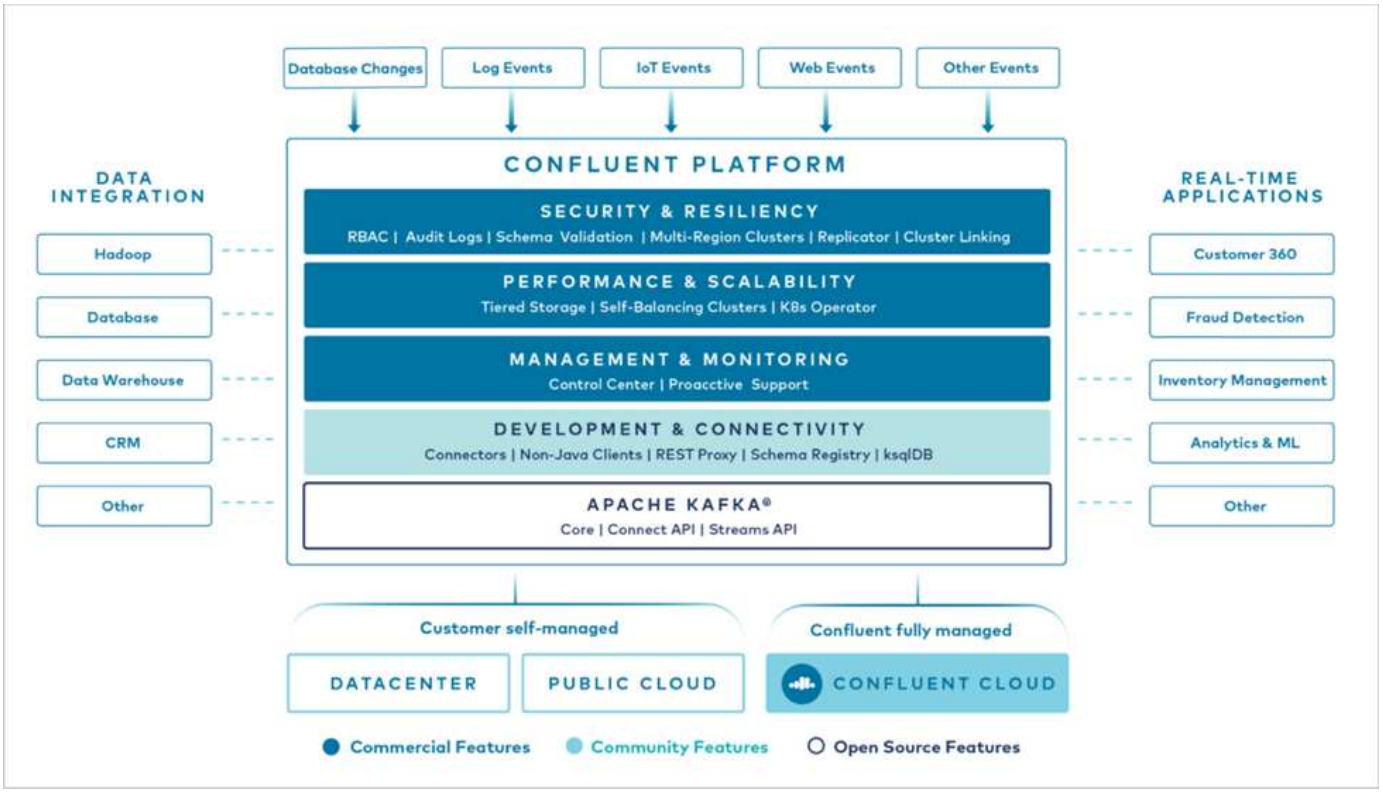
Confluentを選ぶ理由

Confluent は、履歴データとリアルタイム データを単一の信頼できる中央ソースに統合することで、まったく新しいカテゴリの最新のイベント駆動型アプリケーションを簡単に構築し、ユニバーサル データ パイプラインを実現し、完全なスケーラビリティ、パフォーマンス、信頼性を備えた強力な新しいユースケースを実現します。

Confluent は何に使用されますか**?**

Confluent Platform を使用すると、異なるシステム間でデータがどのように転送されるか、または統合されるかといった基礎となる仕組みを心配するのではなく、データからビジネス価値を引き出す方法に集中できます。具体的には、Confluent Platform は、データ ソースを Kafka に接続し、ストリーミング アプリケーションを構築するだけでなく、Kafka インフラストラクチャのセキュリティ保護、監視、管理も簡素化します。現在、Confluent Platform は、金融サービス、オムニチャネル小売、自律走行車から不正検出、マイクロサービス、IoT まで、さまざまな業界の幅広いユースケースに使用されています。

次の図は、Confluent Kafka プラットフォームのコンポーネントを示しています。



Confluentのイベントストリーミングテクノロジーの概要

Confluent Platformの中核は "Apache Kafka"最も人気のあるオープンソースの分散ストリーミング プラットフォームです。 Kafka の主な機能は次のとおりです。

- レコードのストリームを公開およびサブスクライブします。
- フォールト トレラントな方法でレコードのストリームを保存します。

- レコードのストリームを処理します。

Confluent Platform には、すぐに使用できる Schema Registry、REST Proxy、合計 100 個以上の構築済み Kafka コネクタ、ksqlDB も含まれています。

Confluent プラットフォームのエンタープライズ機能の概要

- **Confluent** コントロール センター Kafka を管理および監視するための GUI ベースのシステム。 Kafka Connect を簡単に管理し、他のシステムへの接続を作成、編集、管理できるようになります。
- **Kubernetes** 用の **Confluent**。 Confluent for Kubernetes は Kubernetes オペレーターです。 Kubernetes オペレーターは、特定のプラットフォーム アプリケーションに固有の機能と要件を提供することで、Kubernetes のオーケストレーション機能を拡張します。 Confluent Platform の場合、これには Kubernetes 上の Kafka のデプロイメント プロセスを大幅に簡素化し、一般的なインフラストラクチャ ライフサイクル タスクを自動化することが含まれます。
- *Kafka への Confluent コネクタ。*コネクタは Kafka Connect API を使用して、Kafka をデータベース、キー値ストア、検索インデックス、ファイルシステムなどの他のシステムに接続します。 Confluent Hub には、最も人気のあるデータ ソースとシンク用のダウンロード可能なコネクタがあり、Confluent Platform で完全にテストされサポートされているバージョンのコネクタも含まれています。詳細は以下をご覧ください "ここをクリックしてください。"。
- *自己バランス型クラスター。*自動化された負荷分散、障害検出、自己修復を提供します。手動で調整することなく、必要に応じてブローカーを追加または廃止するためのサポートを提供します。
- *合流クラスターのリンク。*クラスターを直接接続し、リンク ブリッジを介して 1 つのクラスターから別のクラスターにトピックをミラーリングします。クラスター リンクにより、マルチデータセンター、マルチクラスター、ハイブリッド クラウドの展開のセットアップが簡素化されます。
- *Confluent 自動データバランサー*クラスター内のブローカーの数、パーティションのサイズ、パーティションの数、およびリーダーの数を監視します。これにより、データをシフトしてクラスター全体で均一なワークロードを作成しながら、再バランスのトラフィックを調整して、再バランス中の本番ワークロードへの影響を最小限に抑えることができます。
- *合流型複製子。*複数のデータ センターで複数の Kafka クラスターを管理することがこれまで以上に簡単になります。
- *階層型ストレージ*お気に入りのクラウド プロバイダーを使用して大量の Kafka データを保存するオプションを提供し、運用上の負担とコストを削減します。階層型ストレージを使用すると、コスト効率の高いオブジェクト ストレージにデータを保存し、コンピューティング リソースが必要な場合にのみブローカーを拡張できます。
- **Confluent JMS** クライアント。 Confluent Platform には、Kafka 用の JMS 互換クライアントが含まれています。この Kafka クライアントは、バックエンドとして Kafka ブローカーを使用して、JMS 1.1 標準 API を実装します。これは、JMS を使用するレガシー アプリケーションがあり、既存の JMS メッセージ ブローカーを Kafka に置き換えたい場合に役立ちます。
- *Confluent MQTT プロキシ。*中間に MQTT ブローカーを必要とせずに、MQTT デバイスおよびゲートウェイから Kafka に直接データを公開する方法を提供します。
- **Confluent** セキュリティ プラグイン Confluent セキュリティ プラグインは、さまざまな Confluent Platform ツールおよび製品にセキュリティ機能を追加するために使用されます。現在、Confluent REST プロキシには、受信リクエストを認証し、認証されたプリンシパルを Kafka へのリクエストに伝播するのに役立つプラグインが用意されています。これにより、Confluent REST プロキシ クライアントは Kafka ブローカーのマルチテナント セキュリティ機能を利用できるようになります。

合流検証

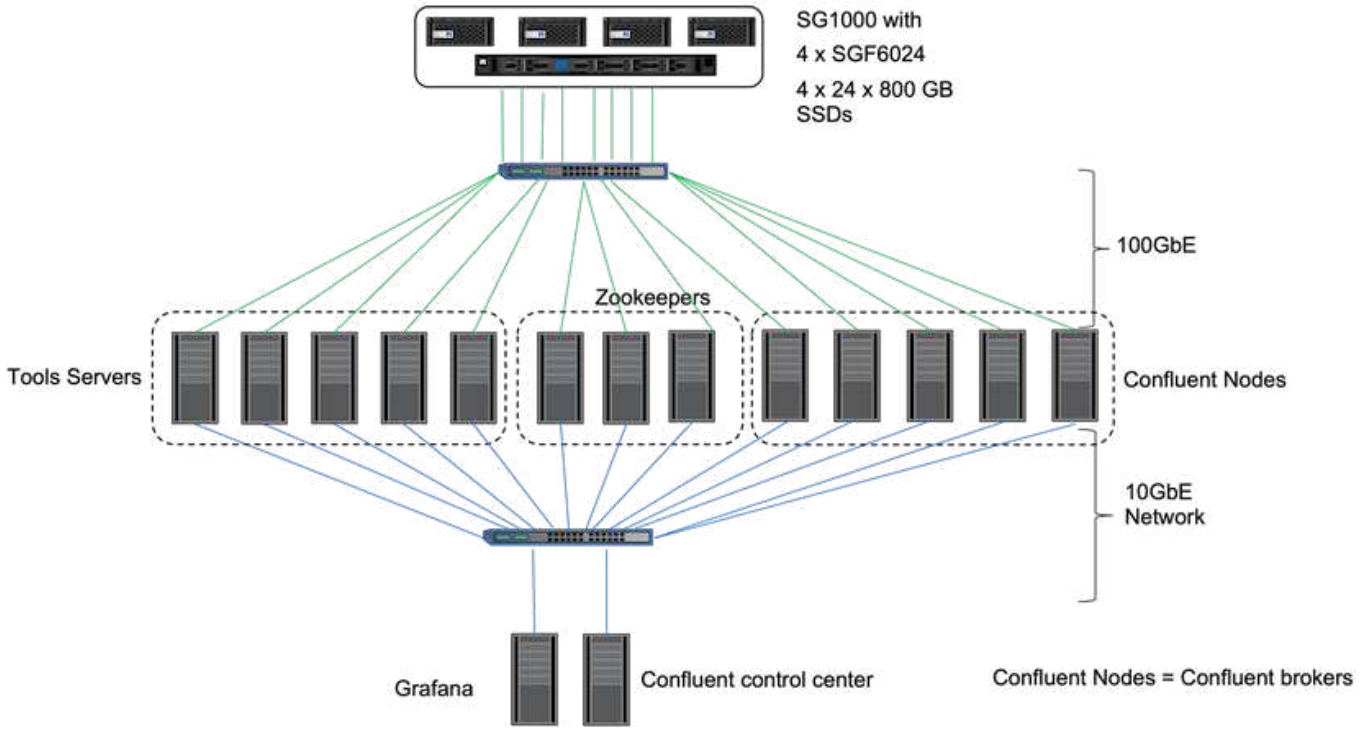
NetApp StorageGRIDの Confluent Platform 6.2 Tiered Storage を使用して検証を実施しました。 NetAppチームと Confluent チームが協力してこの検証に取り組み、検証に必要なテスト ケースを実行しました。

Confluent Platform のセットアップ

検証には次の設定を使用しました。

検証には、3 つの動物園管理人、5 つのブローカー、5 つのテスト スクリプト実行サーバー、256 GB の RAM と 16 個の CPU を備えた名前付きツール サーバーを使用しました。 NetAppストレージには、4 つの SGF6024 を搭載した SG1000 ロード バランサーを備えたStorageGRIDを使用しました。ストレージとブローカーは 100GbE 接続を介して接続されました。

次の図は、Confluent 検証に使用される構成のネットワーク トポロジを示しています。



ツール サーバーは、Confluent ノードに要求を送信するアプリケーション クライアントとして機能します。

Confluent階層型ストレージ構成

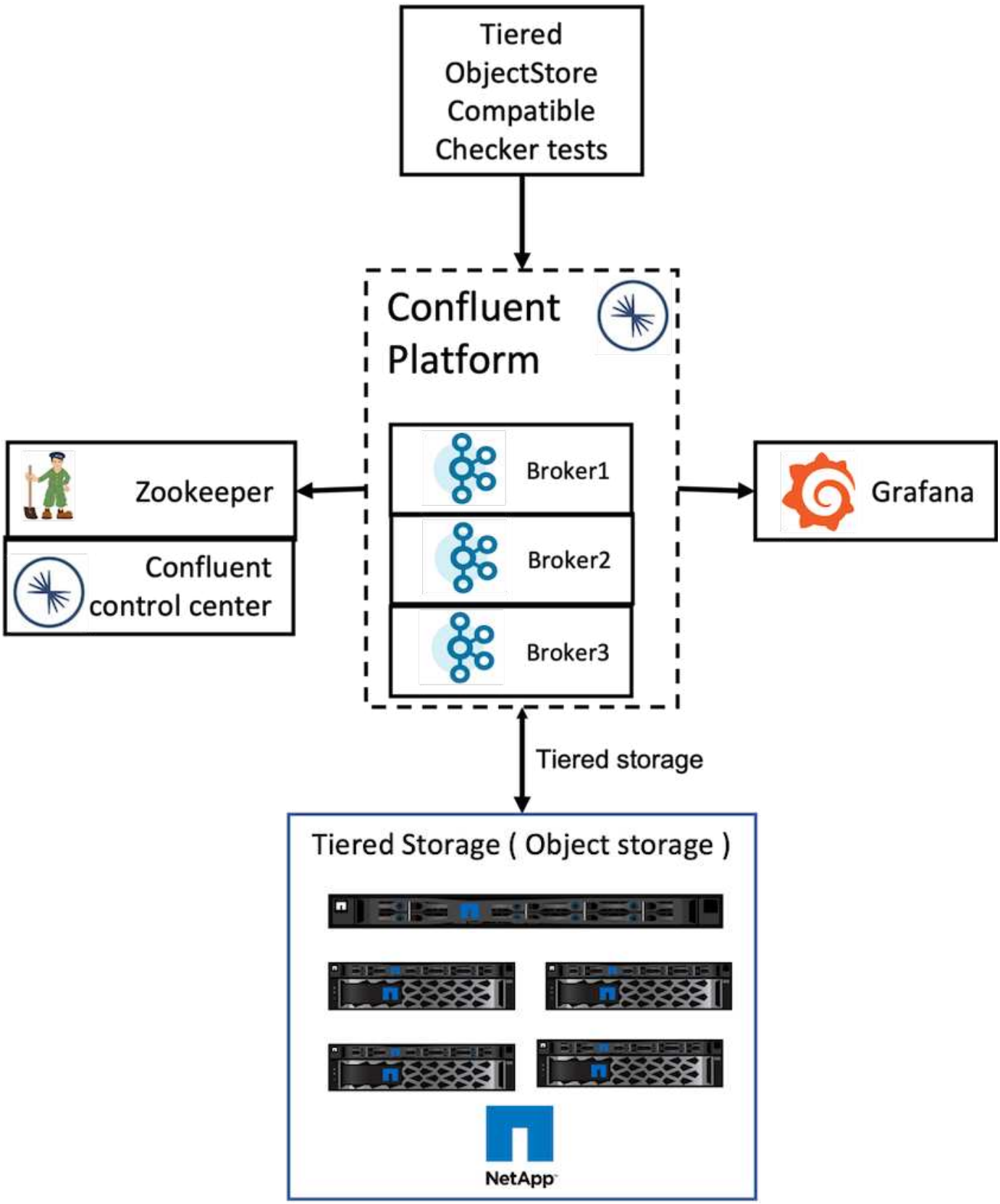
階層化ストレージ構成には、Kafka で次のパラメータが必要です。

```
Confluent.tier.archiver.num.threads=16
confluent.tier.fetcher.num.threads=32
confluent.tier.enable=true
confluent.tier.feature=true
confluent.tier.backend=S3
confluent.tier.s3.bucket=kafkasgdbucket1-2
confluent.tier.s3.region=us-west-2
confluent.tier.s3.cred.file.path=/data/kafka/.ssh/credentials
confluent.tier.s3.aws.endpoint.override=http://kafkasgd.rtpppe.netapp.com:
10444/
confluent.tier.s3.force.path.style.access=true
```

検証には、HTTP プロトコルを使用したStorageGRIDを使用しましたが、HTTPS も機能します。アクセスキーと秘密鍵は、 `confluent.tier.s3.cred.file.path`パラメータ。

NetAppオブジェクトストレージ - StorageGRID

検証のために、 StorageGRIDで単一サイト構成を構成しました。



検証テスト

検証のために以下の 5 つのテストケースを完了しました。これらのテストは Trogdor フレームワークで実行されます。最初の 2 つは機能テストであり、残りの 3 つはパフォーマンス テストでした。

オブジェクトストアの正確性テスト

このテストでは、オブジェクト ストア API のすべての基本操作 (get/put/delete など) が階層化ストレージのニーズに応じて適切に機能するかどうかを判定します。これは、すべてのオブジェクト ストア サービスが次のテストに先立って合格することが期待される基本テストです。合格か不合格かを判断する断定的なテストです。

階層化機能の正確性テスト

このテストでは、合格または不合格のいずれかになるアサーション テストを使用して、エンドツーエンドの階層型ストレージ機能が適切に動作するかどうかを判断します。このテストでは、デフォルトで階層化が有効にされ、ホットセット サイズが大幅に削減されたテスト トピックが作成されます。新しく作成されたテスト トピックにイベント ストリームを生成し、ブローカーがセグメントをオブジェクト ストアにアーカイブするのを待機し、イベント ストリームを消費して、消費されたストリームが生成されたストリームと一致することを検証します。イベント ストリームに生成されるメッセージの数は構成可能であり、ユーザーはテストのニーズに応じて十分な大きさのワークロードを生成できます。ホットセットのサイズが縮小されたことで、アクティブ セグメント外のコンシューマー フェッチがオブジェクト ストアからのみ提供されるようになり、読み取りに対するオブジェクト ストアの正確性をテストするのに役立ちます。このテストは、オブジェクト ストア障害注入ありとなしの状態で実行しました。 StorageGRIDのノードの 1 つでサービス マネージャー サービスを停止し、エンドツーエンドの機能がオブジェクト ストレージで動作することを検証することで、ノード障害をシミュレートしました。

階層フェッチベンチマーク

このテストでは、階層化オブジェクト ストレージの読み取りパフォーマンスを検証し、ベンチマークによって生成されたセグメントからの高負荷状態での範囲フェッチ読み取り要求をチェックしました。このベンチマークでは、Confluent は階層フェッチ要求に対応するカスタム クライアントを開発しました。

生産・消費ワークロードベンチマーク

このテストでは、セグメントのアーカイブを通じてオブジェクト ストアへの書き込みワークロードを間接的に生成しました。読み取りワークロード (読み取られたセグメント) は、コンシューマー グループがセグメントを取得したときにオブジェクト ストレージから生成されました。このワークロードはテスト スクリプトによって生成されました。このテストでは、並列スレッドでのオブジェクト ストレージの読み取りと書き込みのパフォーマンスをチェックしました。階層化機能の正確性テストと同様に、オブジェクト ストア障害注入の有無でテストを行いました。

保持ワークロードベンチマーク

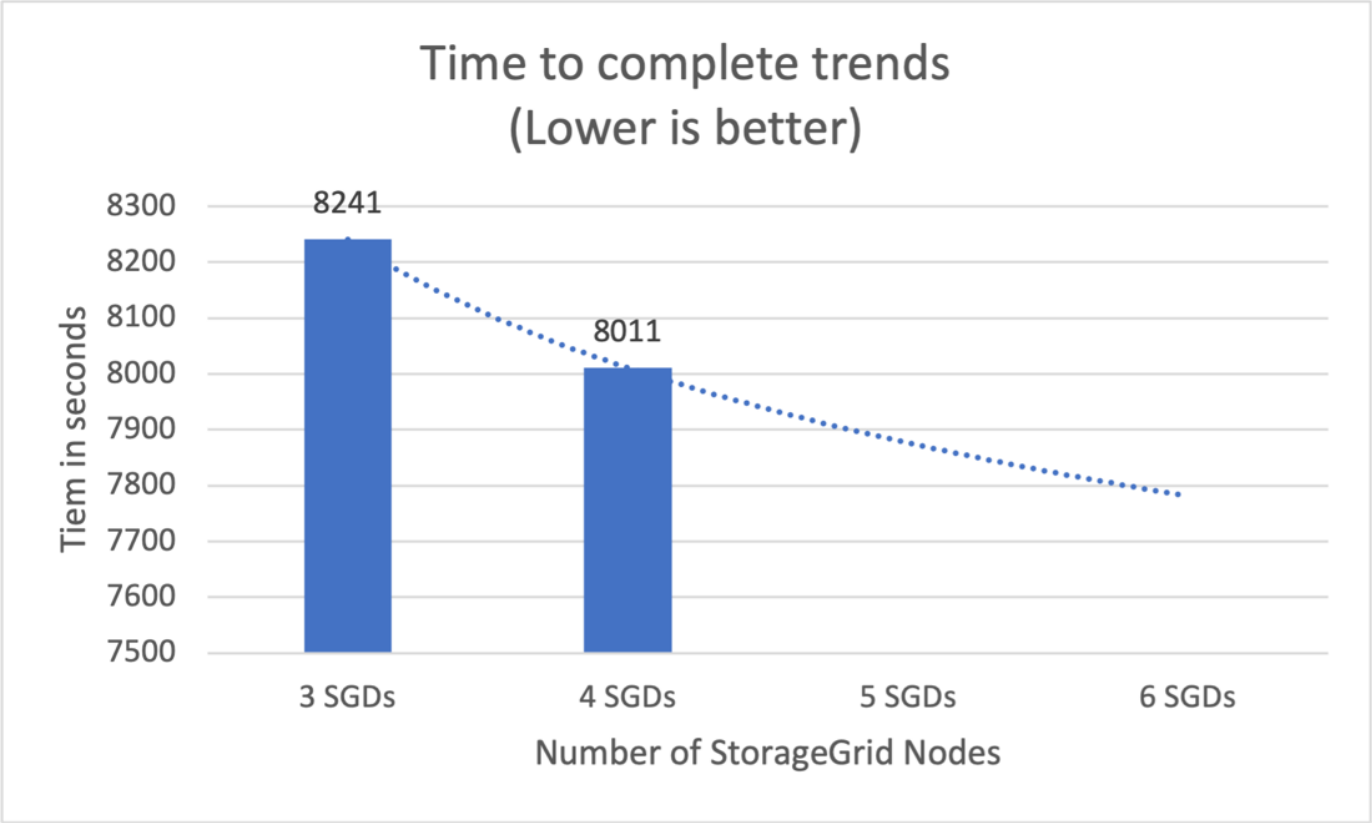
このテストでは、トピック保持の負荷が高い状態でのオブジェクト ストアの削除パフォーマンスをチェックしました。保持ワークロードは、テスト トピックに並行して多数のメッセージを生成するテスト スクリプトを使用して生成されました。テスト トピックでは、サイズ ベースおよび時間ベースの積極的な保持設定が構成されていたため、イベント ストリームがオブジェクト ストアから継続的に消去されていました。その後、セグメントはアーカイブされました。これにより、ブローカーによるオブジェクト ストレージ内の大量の削除と、オブジェクト ストア削除操作のパフォーマンスの収集が行われました。

スケーラビリティを考慮したパフォーマンステスト

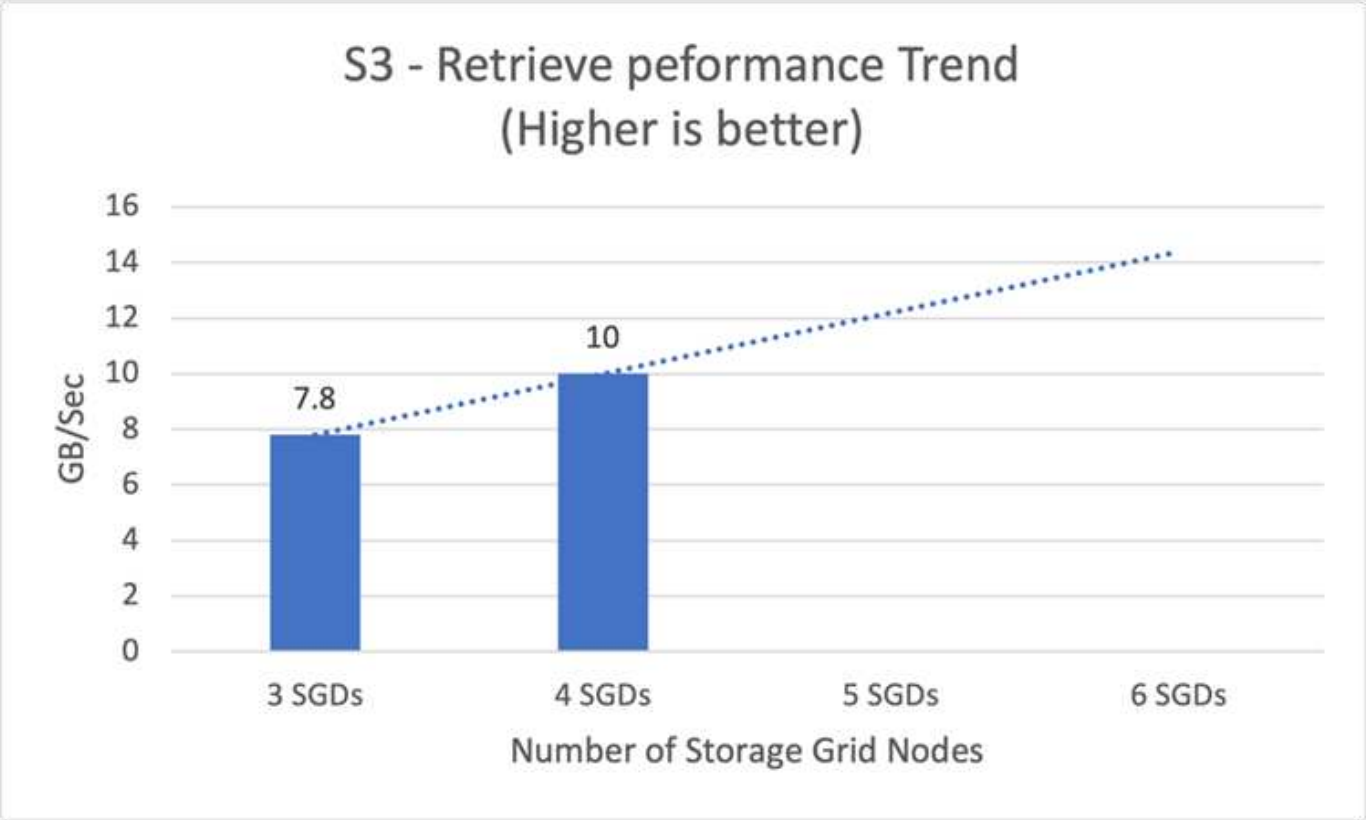
NetApp StorageGRIDセットアップを使用して、プロデューサー ワークロードとコンシューマー ワークロード用に 3 ～ 4 ノードの階層型ストレージ テストを実行しました。当社のテストによると、完了までの時間とパフォーマンス結果は、 StorageGRIDノードの数に正比例しました。 StorageGRID のセットアップには少なくとも 3 つのノードが

必要でした。

- ストレージ ノードの数が増加すると、生成と消費の操作を完了する時間は直線的に減少しました。



- s3 取得操作のパフォーマンスは、 StorageGRIDノードの数に基づいて直線的に増加しました。StorageGRID は最大 200 個の StorageGRID ノードをサポートします。



Confluent S3コネクタ

Amazon S3 Sink コネクタは、Apache Kafka トピックから S3 オブジェクトに Avro 、JSON、または Bytes 形式でデータをエクスポートします。 Amazon S3 シンクコネクタは、Kafka からデータを定期的にポーリングし、それを S3 にアップロードします。パーティショナーは、すべての Kafka パーティションのデータをチャンクに分割するために使用されます。各データ チャンクは S3 オブジェクトとして表されます。キー名は、トピック、Kafka パーティション、およびこのデータ チャンクの開始オフセットをエンコードします。

このセットアップでは、Kafka s3 シンク コネクタを使用して、Kafka からオブジェクト ストレージ内のトピックを直接読み書きする方法を示します。このテストではスタンドアロンの Confluent クラスターを使用しましたが、このセットアップは分散クラスターにも適用できます。

1. Confluent Web サイトから Confluent Kafka をダウンロードします。
2. パッケージをサーバー上のフォルダーに解凍します。
3. 2 つの変数をエクスポートします。

```
Export CONFLUENT_HOME=/data/confluent/confluent-6.2.0
export PATH=$PATH:/data/confluent/confluent-6.2.0/bin
```

4. スタンドアロンのConfluent Kafkaセットアップの場合、クラスタは一時的なルートフォルダを作成します。 `/tmp`また、`Zookeeper`、`Kafka`、スキーマレジストリ、`connect`、`ksql-server`、`control-`

centerフォルダを作成し、それぞれの設定ファイルをコピーします。 `$CONFLUENT_HOME 。次の例を参照してください。

```
root@stlrx2540m1-108:~# ls -ltr /tmp/confluent.406980/
total 28
drwxr-xr-x 4 root root 4096 Oct 29 19:01 zookeeper
drwxr-xr-x 4 root root 4096 Oct 29 19:37 kafka
drwxr-xr-x 4 root root 4096 Oct 29 19:40 schema-registry
drwxr-xr-x 4 root root 4096 Oct 29 19:45 kafka-rest
drwxr-xr-x 4 root root 4096 Oct 29 19:47 connect
drwxr-xr-x 4 root root 4096 Oct 29 19:48 ksql-server
drwxr-xr-x 4 root root 4096 Oct 29 19:53 control-center
root@stlrx2540m1-108:~#
```

5. Zookeeper を設定します。デフォルトのパラメータを使用する場合は、何も変更する必要はありません。

```
root@stlrx2540m1-108:~# cat
/tmp/confluent.406980/zookeeper/zookeeper.properties  | grep -iv ^#
dataDir=/tmp/confluent.406980/zookeeper/data
clientPort=2181
maxClientCnxns=0
admin.enableServer=false
tickTime=2000
initLimit=5
syncLimit=2
server.179=controlcenter:2888:3888
root@stlrx2540m1-108:~#
```

上記の構成では、 `server. xxx`財産。デフォルトでは、Kafka リーダーの選択には 3 つの Zookeeper が必要です。

6. myidファイルを作成した `/tmp/confluent.406980/zookeeper/data`一意のIDを持つ:

```
root@stlrx2540m1-108:~# cat /tmp/confluent.406980/zookeeper/data/myid
179
root@stlrx2540m1-108:~#
```

myid ファイルには、最後の IP アドレス番号を使用しました。 Kafka、connect、control-center、Kafka 、Kafka-rest、ksql-server、および schema-registry 構成にはデフォルト値を使用しました。

7. Kafka サービスを開始します。

```
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# confluent
local services  start
The local commands are intended for a single-node development
environment only,
NOT for production usage.

Using CONFLUENT_CURRENT: /tmp/confluent.406980
ZooKeeper is [UP]
Kafka is [UP]
Schema Registry is [UP]
Kafka REST is [UP]
Connect is [UP]
ksqlDB Server is [UP]
Control Center is [UP]
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

各構成にはログ フォルダーがあり、問題のトラブルシューティングに役立ちます。場合によっては、サービスの起動に時間がかかることがあります。すべてのサービスが稼働していることを確認してください。

8. Kafka connectをインストールするには `confluent-hub`。

```
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# ./confluent-
hub install confluentinc/kafka-connect-s3:latest
The component can be installed in any of the following Confluent
Platform installations:
  1. /data/confluent/confluent-6.2.0 (based on $CONFLUENT_HOME)
  2. /data/confluent/confluent-6.2.0 (where this tool is installed)
Choose one of these to continue the installation (1-2): 1
Do you want to install this into /data/confluent/confluent-
6.2.0/share/confluent-hub-components? (yN) y

Component's license:
Confluent Community License
http://www.confluent.io/confluent-community-license
I agree to the software license agreement (yN) y
Downloading component Kafka Connect S3 10.0.3, provided by Confluent,
Inc. from Confluent Hub and installing into /data/confluent/confluent-
6.2.0/share/confluent-hub-components
Do you want to uninstall existing version 10.0.3? (yN) y
Detected Worker's configs:
  1. Standard: /data/confluent/confluent-6.2.0/etc/kafka/connect-
distributed.properties
  2. Standard: /data/confluent/confluent-6.2.0/etc/kafka/connect-
standalone.properties
  3. Standard: /data/confluent/confluent-6.2.0/etc/schema-
```

```
registry/connect-avro-distributed.properties
  4. Standard: /data/confluent/confluent-6.2.0/etc/schema-
registry/connect-avro-standalone.properties
  5. Based on CONFLUENT_CURRENT:
/tmp/confluent.406980/connect/connect.properties
  6. Used by Connect process with PID 15904:
/tmp/confluent.406980/connect/connect.properties
Do you want to update all detected configs? (yN) y
Adding installation directory to plugin path in the following files:
  /data/confluent/confluent-6.2.0/etc/kafka/connect-
distributed.properties
  /data/confluent/confluent-6.2.0/etc/kafka/connect-
standalone.properties
  /data/confluent/confluent-6.2.0/etc/schema-registry/connect-avro-
distributed.properties
  /data/confluent/confluent-6.2.0/etc/schema-registry/connect-avro-
standalone.properties
  /tmp/confluent.406980/connect/connect.properties
  /tmp/confluent.406980/connect/connect.properties

Completed
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

特定のバージョンをインストールするには、 `confluent-hub install confluentinc/kafka-connect-s3:10.0.3` 。

9. デフォルトでは、 `confluentinc-kafka-connect-s3`インストールされている `/data/confluent/confluent-6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-s3`。
10. プラグインのパスを新しいものに更新します `confluentinc-kafka-connect-s3`。

```
root@stlrx2540m1-108:~# cat /data/confluent/confluent-
6.2.0/etc/kafka/connect-distributed.properties | grep plugin.path
#
plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/co
nnectors,
plugin.path=/usr/share/java,/data/zookeeper/confluent/confluent-
6.2.0/share/confluent-hub-components,/data/confluent/confluent-
6.2.0/share/confluent-hub-components,/data/confluent/confluent-
6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-s3
root@stlrx2540m1-108:~#
```

11. Confluent サービスを停止して再起動します。

```
confluent local services  stop
confluent local services  start
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin# confluent
local services  status
The local commands are intended for a single-node development
environment only,
NOT for production usage.

Using CONFLUENT_CURRENT: /tmp/confluent.406980
Connect is [UP]
Control Center is [UP]
Kafka is [UP]
Kafka REST is [UP]
ksqlDB Server is [UP]
Schema Registry is [UP]
ZooKeeper is [UP]
root@stlrx2540m1-108:/data/confluent/confluent-6.2.0/bin#
```

12. アクセスIDと秘密鍵を `/root/.aws/credentials`ファイル。

```
root@stlrx2540m1-108:~# cat /root/.aws/credentials
[default]
aws_access_key_id = xxxxxxxxxxxx
aws_secret_access_key = xxxxxxxxxxxxxxxxxxxxxxxxxx
root@stlrx2540m1-108:~#
```

13. バケットにアクセスできることを確認します。

```
root@stlrx2540m4-01:~# aws s3 –endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 ls kafkasgdbucket1-2
2021-10-29 21:04:18       1388 1
2021-10-29 21:04:20       1388 2
2021-10-29 21:04:22       1388 3
root@stlrx2540m4-01:~#
```

14. s3 およびバケット構成用に s3-sink プロパティ ファイルを構成します。

```
root@stlrx2540m1-108:~# cat /data/confluent/confluent-
6.2.0/share/confluent-hub-components/confluentinc-kafka-connect-
s3/etc/quickstart-s3.properties | grep -v ^#
name=s3-sink
connector.class=io.confluent.connect.s3.S3SinkConnector
tasks.max=1
topics=s3_testtopic
s3.region=us-west-2
s3.bucket.name=kafkasgdbucket1-2
store.url=http://kafkasgd.rtpppe.netapp.com:10444/
s3.part.size=5242880
flush.size=3
storage.class=io.confluent.connect.s3.storage.S3Storage
format.class=io.confluent.connect.s3.format.avro.AvroFormat
partitioner.class=io.confluent.connect.storage.partitioner.DefaultPartit
ioner
schema.compatibility=NONE
root@stlrx2540m1-108:~#
```

15. いくつかのレコードを s3 バケットにインポートします。

```
kafka-avro-console-producer --broker-list localhost:9092 --topic
s3_topic \
--property
value.schema='{"type":"record","name":"myrecord","fields":[{"name":"f1",
"type":"string"}]}'
{"f1": "value1"}
{"f1": "value2"}
{"f1": "value3"}
{"f1": "value4"}
{"f1": "value5"}
{"f1": "value6"}
{"f1": "value7"}
{"f1": "value8"}
{"f1": "value9"}
```

16. s3-sink コネクタをロードします。

```
root@stlrx2540m1-108:~# confluent local services connect connector load
s3-sink  --config /data/confluent/confluent-6.2.0/share/confluent-hub-
components/confluentinc-kafka-connect-s3/etc/quickstart-s3.properties
The local commands are intended for a single-node development
environment only,
NOT for production usage.
https://docs.confluent.io/current/cli/index.html
{
  "name": "s3-sink",
  "config": {
    "connector.class": "io.confluent.connect.s3.S3SinkConnector",
    "flush.size": "3",
    "format.class": "io.confluent.connect.s3.format.avro.AvroFormat",
    "partitioner.class":
"io.confluent.connect.storage.partitioner.DefaultPartitioner",
    "s3.bucket.name": "kafkasgdbucket1-2",
    "s3.part.size": "5242880",
    "s3.region": "us-west-2",
    "schema.compatibility": "NONE",
    "storage.class": "io.confluent.connect.s3.storage.S3Storage",
    "store.url": "http://kafkasgd.rtpppe.netapp.com:10444/",
    "tasks.max": "1",
    "topics": "s3_testtopic",
    "name": "s3-sink"
  },
  "tasks": [],
  "type": "sink"
}
root@stlrx2540m1-108:~#
```

17. s3-sink のステータスを確認します。

```
root@stlrx2540m1-108:~# confluent local services connect connector
status s3-sink
The local commands are intended for a single-node development
environment only,
NOT for production usage.
https://docs.confluent.io/current/cli/index.html
{
  "name": "s3-sink",
  "connector": {
    "state": "RUNNING",
    "worker_id": "10.63.150.185:8083"
  },
  "tasks": [
    {
      "id": 0,
      "state": "RUNNING",
      "worker_id": "10.63.150.185:8083"
    }
  ],
  "type": "sink"
}
root@stlrx2540m1-108:~#
```

18. ログをチェックして、s3-sink がトピックを受け入れる準備ができていることを確認します。

```
root@stlrx2540m1-108:~# confluent local services connect log
```

19. Kafka のトピックを確認します。

```
kafka-topics --list --bootstrap-server localhost:9092
…
connect-configs
connect-offsets
connect-statuses
default_ksql_processing_log
s3_testtopic
s3_topic
s3_topic_new
root@stlrx2540m1-108:~#
```

20. s3 バケット内のオブジェクトを確認します。

```
root@stlrx2540m1-108:~# aws s3 --endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 ls --recursive kafkasgdbucket1-
2/topics/
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000003.avro
2021-10-29 21:24:00        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000006.avro
2021-10-29 21:24:08        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000009.avro
2021-10-29 21:24:08        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000012.avro
2021-10-29 21:24:09        213
topics/s3_testtopic/partition=0/s3_testtopic+0+0000000015.avro
root@stlrx2540m1-108:~#
```

21. 内容を確認するには、次のコマンドを実行して、各ファイルを S3 からローカルファイルシステムにコピーします。

```
root@stlrx2540m1-108:~# aws s3 --endpoint-url
http://kafkasgd.rtpppe.netapp.com:10444 cp s3://kafkasgdbucket1-
2/topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro
tes.avro
download: s3://kafkasgdbucket1-
2/topics/s3_testtopic/partition=0/s3_testtopic+0+0000000000.avro to
./tes.avro
root@stlrx2540m1-108:~#
```

22. レコードを印刷するには、avro-tools-1.11.0.1.jar（ "Apacheアーカイブ" ）。

```
root@stlrx2540m1-108:~# java -jar /usr/src/avro-tools-1.11.0.1.jar
tojson tes.avro
21/10/30 00:20:24 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
{"f1":"value1"}
{"f1":"value2"}
{"f1":"value3"}
root@stlrx2540m1-108:~#
```

Instaclustr Kafka Connect コネクタ

Instaclustr は Kafka Connect コネクタとその詳細をサポートします - "詳細"。Instaclustrは追加のコネクタを提供します "詳細"

合流型自己バランスクラスター

これまでに Kafka クラスターを管理したことがある場合は、クラスター全体でワークロードのバランスをとるためにパーティションを別のブローカーに手動で再割り当てする際の課題をよくご存知でしょう。大規模な Kafka を導入している組織では、特にミッションクリティカルなアプリケーションがクラスター上に構築されている場合、大量のデータの再シャッフルは困難で面倒でリスクを伴う可能性があります。ただし、Kafka の使用例が最小であっても、プロセスには時間がかかり、人為的エラーが発生しやすくなります。

私たちのラボでは、クラスター トポロジの変更や負荷の不均一性に基づいて再バランス調整を自動化する Confluent の自己バランス調整クラスター機能をテストしました。 Confluent の再バランス テストは、ノード障害が発生したとき、またはスケーリング ノードでブローカー間でデータの再バランス調整が必要になったときに、新しいブローカーを追加する時間を測定するのに役立ちます。従来の Kafka 構成では、クラスターの拡大に伴って再バランス調整するデータの量も増加しますが、階層型ストレージでは再バランス調整は少量のデータに制限されます。当社の検証によると、階層型ストレージの再バランス調整は、従来の Kafka アーキテクチャでは数秒または数分かかり、クラスターの拡大に伴って直線的に増加します。

自己バランス型クラスターでは、パーティションの再バランスが完全に自動化され、Kafka のスループットが最適化され、ブローカーのスケーリングが加速され、大規模なクラスターを実行する際の運用上の負担が軽減されます。定常状態では、自己バランス型クラスターがブローカー間のデータの偏りを監視し、パーティションを継続的に再割り当てしてクラスターのパフォーマンスを最適化します。プラットフォームを拡大または縮小する場合、自己バランス型クラスターは新しいブローカーの存在または古いブローカーの削除を自動的に認識し、後続のパーティションの再割り当てをトリガーします。これにより、ブローカーを簡単に追加および廃止できるようになり、Kafka クラスターの弾力性が根本的に向上します。これらの利点は、手動による介入、複雑な計算、またはパーティションの再割り当てに通常伴う人的エラーのリスクを必要とせずに得られます。その結果、データの再バランス調整ははるかに短い時間で完了し、クラスターを常に監視する必要がなくなり、より価値の高いイベント ストリーミング プロジェクトに集中できるようになります。

Instaclustr はセルフリバランス機能もサポートしており、複数の顧客に実装されています。

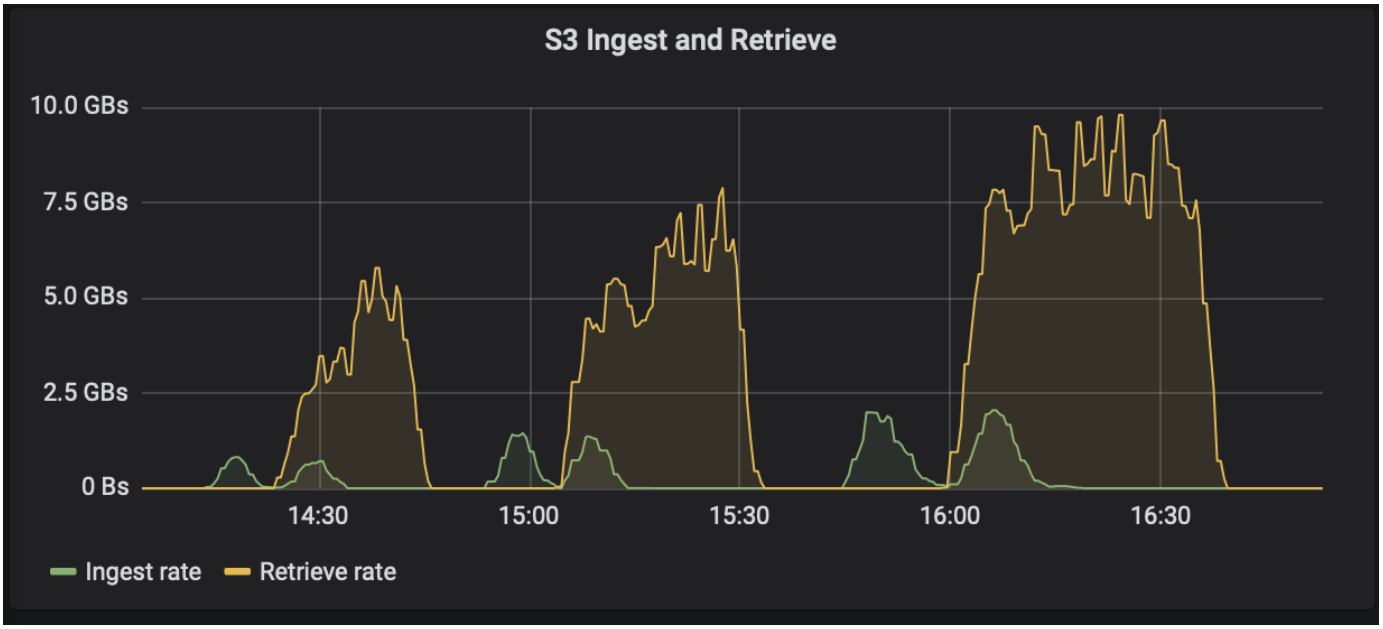
ベストプラクティスガイドライン

このセクションでは、この認定から得られた教訓を紹介します。

- 当社の検証によると、Confluent がデータを保存するには S3 オブジェクト ストレージが最適です。
- Confluent 階層型ストレージ構成では、ブローカー データ ディレクトリに保持されるデータのサイズは、データがオブジェクト ストレージに移動されるときのセグメント サイズと保持期間に基づいているため、高スループット SAN (具体的には FC) を使用してブローカーのホット データまたはローカル ディスクを保持できます。
- オブジェクト ストアでは、segment.bytes が大きいほどパフォーマンスが向上します。512 MB でテストしました。
- Kafkaでは、トピックに生成される各レコードのキーまたは値の長さ（バイト単位）は、

`length.key.value`パラメータ。 StorageGRIDの場合、S3 オブジェクトの取り込みおよび取得のパフォーマンスがより高い値に向上しました。たとえば、512 バイトでは 5.8 GBps の取得、1024 バイトでは 7.5 GBps の s3 取得、2048 バイトでは 10 GBps に近い取得が実現しました。

次の図は、S3オブジェクトの取り込みと取得を次の表に基づいて示しています。 `length.key.value` 。



- *Kafka のチューニング。*階層化ストレージのパフォーマンスを向上させるには、TierFetcherNumThreads と TierArchiverNumThreads を増やすことができます。一般的なガイドラインとして、TierFetcherNumThreads を物理 CPU コアの数に合わせて増やし、TierArchiverNumThreads を CPU コアの数の半分まで増やします。たとえば、サーバーのプロパティで、8 つの物理コアを持つマシンがある場合は、confluent.tier.fetcher.num.threads = 8 および confluent.tier.archiver.num.threads = 4 に設定します。
- *トピック削除の時間間隔。*トピックが削除されても、オブジェクト ストレージ内のログ セグメント ファイルの削除はすぐには開始されません。むしろ、それらのファイルが削除されるまでの期間が、デフォルト値の 3 時間に設定されています。この間隔の値を変更するには、構成 confluent.tier.topic.delete.check.interval.ms を変更します。トピックまたはクラスターを削除する場合は、それぞれのバケット内のオブジェクトを手動で削除することもできます。
- *階層化ストレージの内部トピックに関する ACL。*オンプレミス展開で推奨されるベスト プラクティスは、階層化ストレージに使用される内部トピックで ACL 承認者を有効にすることです。 ACL ルールを設定して、このデータへのアクセスをブローカー ユーザーのみに制限します。これにより、内部トピックが保護され、階層化ストレージ データとメタデータへの不正アクセスが防止されます。

```
kafka-acls --bootstrap-server localhost:9092 --command-config adminclient-
configs.conf \
--add --allow-principal User:<kafka> --operation All --topic "_confluent-
tier-state"
```

ユーザーを置き換える `<kafka>`デプロイメント内の実際のブローカー プリンシパルを使用します。

例えば、コマンド `confluent-tier-state`階層化ストレージの内部トピックに ACL を設定します。現在、階層化

ストレージに関連する内部トピックは 1 つだけです。この例では、内部トピックのすべての操作に対してプリンシパル Kafka 権限を提供する ACL を作成します。

サイジング

Kafka のサイズ設定は、シンプル、詳細、リバース、パーティションの 4 つの構成モードで実行できます。

単純

シンプル モードは、Apache Kafka を初めて使用するユーザーや初期状態のユース ケースに適しています。このモードでは、スループット MBps、読み取りファンアウト、保持期間、リソース使用率 (デフォルトは 60%) などの要件を指定します。また、オンプレミス (ベアメタル、VMware、Kubernetes、OpenStack) やクラウドなどの環境も入力します。この情報に基づいて、Kafka クラスターのサイズを決定すると、ブローカー、Zookeeper、Apache Kafka Connect Workers、スキーマ レジストリ、REST プロキシ、ksqlDB、および Confluent コントロール センターに必要なサーバーの数がわかります。

階層型ストレージの場合、Kafka クラスターのサイズ設定にはきめ細かな構成モードを検討してください。粒度モードは、経験豊富な Apache Kafka ユーザーや明確に定義されたユースケースに適しています。このセクションでは、プロデューサー、ストリーム プロセッサ、およびコンシューマーのサイズ設定について説明します。

プロデューサー

Apache Kafka のプロデューサー (ネイティブ クライアント、REST プロキシ、Kafka コネクタなど) を記述するには、次の情報を提供します。

- *名前。*スパーク。
- *プロデューサータイプ。*アプリケーションまたはサービス、プロキシ (REST、MQTT、その他)、既存のデータベース (RDBMS、NOSQL、その他)。 「分からない」を選択することもできます。
- 平均スループット 1 秒あたりのイベント数 (たとえば 1,000,000)。
- ピークスループット 1 秒あたりのイベント数 (たとえば 4,000,000)。
- *平均メッセージサイズ*バイト単位、非圧縮（最大 1 MB、たとえば 1000）。
- *メッセージ形式*オプションには、Avro、JSON、プロトコル バッファー、バイナリ、テキスト、「わかりません」などがあります。
- *複製係数*オプションは 1、2、3 (Confluent 推奨)、4、5、または 6 です。
- *保持時間*ある日（例えば）。 Apache Kafka にデータをどれくらいの期間保存しますか? 任意の単位で -1 を入力すると、無限の時間になります。計算機では、無限保持の場合の保持期間を 10 年と想定しています。
- 「階層化ストレージを有効にしてブローカー数を減らし、無制限のストレージを可能にしますか?」のチェックボックスを選択します。
- 階層化ストレージが有効な場合、保持フィールドはブローカーにローカルに保存されるホット データ セットを制御します。アーカイブ保持フィールドは、アーカイブ オブジェクト ストレージにデータが保存される期間を制御します。
- アーカイブストレージの保持。 1年（例）。データをアーカイブストレージにどれくらいの期間保存しますか? 任意の単位で -1 を入力すると、継続時間は無制限になります。計算機では、無制限の保持期間とし

て 10 年間を想定しています。

- 成長乗数 1 (例)このパラメータの値が現在のスループットに基づいている場合は、1 に設定します。追加の成長に基づいてサイズを決定するには、このパラメータを成長乗数に設定します。
- プロデューサーインスタンスの数。 10（例）。いくつのプロデューサーインスタンスが実行されますか? この入力は、CPU 負荷をサイズ計算に組み込むために必要です。空白の値は、CPU 負荷が計算に組み込まれていないことを示します。

このサンプル入力に基づくと、サイズ設定はプロデューサーに次のような影響を及ぼします。

- 非圧縮バイトでの平均スループット: 1GBps。非圧縮バイトでのピーク スループット: 4GBps。圧縮バイトでの平均スループット: 400MBps。圧縮バイトでのピーク スループット: 1.6GBps。これはデフォルトの 60% の圧縮率に基づいています (この値は変更できます)。
  - 必要なブローカー上のホットセット ストレージの合計: レプリケーション、圧縮を含む 31,104 TB。必要なブローカー外アーカイブ ストレージの合計: 378,432 TB (圧縮済み)。使用"https://fusion.netapp.com"StorageGRID のサイズ設定用。

ストリーム プロセッサは、Apache Kafka からデータを消費し、Apache Kafka にデータを返すアプリケーションまたはサービスを記述する必要があります。ほとんどの場合、これらは ksqlDB または Kafka Streams に組み込まれています。

- *名前。*スパークストリーマー。
- *処理時間*このプロセッサは 1 つのメッセージを処理するのにどのくらいの時間がかかりますか?
  - 1 ミリ秒 (単純なステートレス変換) [例]、10 ミリ秒 (ステートフルなメモリ内操作)。
  - 100 ミリ秒 (ステートフル ネットワークまたはディスク操作)、1000 ミリ秒 (サードパーティの REST 呼び出し)。
  - 私はこのパラメータをベンチマークし、どれくらいの時間がかかるかを正確に把握しています。
- 出力保持 1日（例）ストリーム プロセッサは、Apache Kafka に出力を返します。この出力データを Apache Kafka にどれくらいの期間保存しますか? 任意の単位で -1 を入力すると、継続時間は無制限になります。
- 「階層化ストレージを有効にしてブローカー数を減らし、無制限のストレージを可能にしますか?」のチェックボックスを選択します。
- アーカイブストレージの保持。 1年（例）。データをアーカイブストレージにどれくらいの期間保存しますか? 任意の単位で -1 を入力すると、継続時間は無制限になります。計算機では、無制限の保持期間として 10 年間を想定しています。
- 出力パススルー率。 100（例）。ストリーム プロセッサは、Apache Kafka に出力を返します。受信スループットの何パーセントが Apache Kafka に出力されますか? たとえば、受信スループットが 20MBps で、この値が 10 の場合、出力スループットは 2MBps になります。
- これはどのアプリケーションから読み取るのでしょうか? プロデューサー タイプに基づくサイズ設定で使用される名前「Spark」を選択します。上記の入力に基づいて、ストリーム プロセッサ インスタンスとトピック パーティションの推定に対するサイズ設定の次のような影響が予想されます。
- このストリーム プロセッサ アプリケーションには、次の数のインスタンスが必要です。着信トピックにも、おそらくこれだけのパーティションが必要になります。このパラメータを確認するには、Confluent にお問い合わせください。
  - 成長乗数なしの平均スループットは 1,000
  - 成長乗数なしのピークスループットの場合は 4,000

- 成長乗数付き平均スループットは1,000
- 成長乗数によるピークスループット4,000

消費者

Apache Kafka からデータを消費し、Apache Kafka にデータを返さないアプリケーションまたはサービス (ネイティブ クライアントや Kafka コネクタなど) について説明します。

- 名前。 Spark の消費者。
- *処理時間*このコンシューマーが 1 つのメッセージを処理するのにどれくらいの時間がかかりますか?
  - 1 ミリ秒 (たとえば、ログ記録のような単純でステートレスなタスク)
  - 10 ミリ秒 (データストアへの高速書き込み)
  - 100 ミリ秒 (データストアへの書き込みが遅い)
  - 1000ミリ秒（サードパーティのREST呼び出し）
  - 期間が既知のその他のベンチマーク プロセス。
- *消費者タイプ。*既存のデータストア (RDBMS、NoSQL、その他) へのアプリケーション、プロキシ、またはシンク。
- これはどのアプリケーションから読み取るのでしょうか? このパラメータを、以前に決定したプロデューサーおよびストリームのサイズに接続します。

上記の入力に基づいて、コンシューマー インスタンスのサイズとトピック パーティションの見積りを決定する必要があります。コンシューマー アプリケーションには次の数のインスタンスが必要です。

- 平均スループットは2,000、成長乗数なし
- ピーク時のスループットは8,000、成長乗数なし
- 成長乗数を含む平均スループットは2,000
- 成長乗数を含むピークスループットは8,000

着信トピックにもこの数のパーティションが必要になる可能性があります。確認するには Confluent にお問い合わせください。

プロデューサー、ストリーム プロセッサ、コンシューマーの要件に加えて、次の追加要件も提供する必要があります。

- *再建の時間です。*たとえば、4 時間。 Apache Kafka ブローカー ホストに障害が発生し、そのデータが失われ、障害が発生したホストの代わりに新しいホストがプロビジョニングされた場合、この新しいホストはどのくらいの速さで再構築する必要がありますか? 値が不明な場合は、このパラメータを空白のままにしておきます。
- *リソース使用率目標（パーセント）*たとえば、60。平均スループット時にホストをどの程度利用したいですか? Confluent では、Confluent 自己バランス クラスターを使用している場合を除き、60% の使用率を推奨しています。その場合、使用率はさらに高くなります。

あなたの環境を説明してください

- クラスターはどのような環境で実行されますか**?** Amazon Web Services、Microsoft Azure、Google Cloud Platform、オンプレミスのベアメタル、オンプレミスの VMware、オンプレミスの OpenStack、オンプレ

ミスの Kubernates のどれですか?

- *ホストの詳細*コア数: 48 (例)、ネットワーク カードの種類 (10GbE、40GbE、16GbE、1GbE、またはその他の種類)。
- *ストレージボリューム*ホスト: 12 (例)ホストごとにいくつのハードドライブまたは SSD がサポートされますか? Confluent では、ホストごとに 12 台のハード ドライブを推奨しています。
- ストレージ容量/ボリューム（**GB**単位） 1000（例）。 1 つのボリュームにはギガバイト単位でどれくらいのストレージを保存できますか? Confluent では 1 TB のディスクを推奨します。
- *ストレージ構成*ストレージ ボリュームはどのように構成されますか? Confluent では、Confluent のすべての機能を活用するために RAID10 を推奨しています。 JBOD、SAN、RAID 1、RAID 0、RAID 5 などのタイプもサポートされています。
- 単一ボリュームのスループット **(MBps)**。 125（例）。単一のストレージ ボリュームは、メガバイト/秒単位でどのくらいの速度で読み取りまたは書き込みできますか? Confluent では、通常 125MBps のスループットを持つ標準ハードドライブを推奨しています。
- メモリ容量**(GB)** 64（例）。

環境変数を決定したら、「クラスターのサイズを設定」を選択します。上記の例のパラメータに基づいて、Confluent Kafka のサイズを次のように決定しました。

- *Apache Kafka。*ブローカー数: 22。クラスターはストレージに制限されています。ホスト数を減らし、無制限のストレージを可能にするために、階層化ストレージを有効にすることを検討してください。
- *Apache ZooKeeper。*数: 5、Apache Kafka Connect Workers: 数: 2、スキーマ レジストリ: 数: 2、REST プロキシ: 数: 2、ksqlDB: 数: 2、Confluent Control Center: 数: 1。

ユースケースを考慮しないプラットフォーム チームにはリバース モードを使用します。パーティション モードを使用して、1 つのトピックに必要なパーティションの数を計算します。見る https://eventsizer.ioリバースおよびパーティション モードに基づいてサイズを決定します。

まとめ

このドキュメントでは、検証テスト、階層化ストレージのパフォーマンス結果、チューニング、Confluent S3 コネクタ、自己バランス機能など、Confluent 階層化ストレージをNetAppストレージと共に使用するためのベスト プラクティス ガイドラインを示します。 ILM ポリシー、検証のための複数のパフォーマンス テストによる Confluent のパフォーマンス、業界標準の S3 API を考慮すると、 NetApp StorageGRIDオブジェクト ストレージは Confluent 階層型ストレージに最適な選択肢です。

詳細情報の入手方法

このドキュメントに記載されている情報の詳細については、次のドキュメントや Web サイトを参照してください。

- Apache Kafkaとは

  "https://www.confluent.io/what-is-apache-kafka/"

- NetApp製品ドキュメント

"https://www.netapp.com/support-and-training/documentation/"

- S3シンクパラメータの詳細

  "https://docs.confluent.io/kafka-connect-s3-sink/current/configuration_options.html#s3-configuration-options"

- Apache Kafka

  "https://en.wikipedia.org/wiki/Apache_Kafka"

- Confluent Platform の無限ストレージ

  "https://www.confluent.io/blog/infinite-kafka-storage-in-confluent-platform/"

- Confluent 階層型ストレージ - ベストプラクティスとサイジング

  "https://docs.confluent.io/platform/current/kafka/tiered-storage.html#best-practices-and-recommendations"

- Confluent Platform 用 Amazon S3 シンクコネクタ

  "https://docs.confluent.io/kafka-connect-s3-sink/current/overview.html"

- Kafka のサイズ設定

  "https://eventsizer.io"

- StorageGRIDのサイズ設定

  "https://fusion.netapp.com/"

- Kafkaのユースケース

  "https://kafka.apache.org/uses"

- Confluent Platform 6.0 における Kafka クラスターの自己バランス

  "https://www.confluent.io/blog/self-balancing-kafka-clusters-in-confluent-platform-6-0/"

  "https://www.confluent.io/blog/confluent-platform-6-0-delivers-the-most-powerful-event-streaming-platform-to-date/"

- Instaclustr の顧客サンプルとそのユースケースの詳細

  https://www.instaclustr.com/blog/netapp-and-pegasystems-open-source-support-package/、https://www.instaclustr.com/wp-content/uploads/Insta_Case_Study_Pegasystems_1_21sep25.pdf

  https://www.instaclustr.com/resources/customer-case-study-pubnub/

  https://www.instaclustr.com/resources/customer-case-study-tesouro/

著作権に関する情報

商標に関する情報