

NetScaler® オブザーバビリティエクス ポーター

Contents

リリースノート	5
リリース 1.10.001	5
修正された問題	5
リリース 1.9.001	5
新機能	5
DEBUG 重大度レベルのサポート	5
監査ログとイベントの Kafka へのエクスポートのサポート	5
修正された問題	5
NetScaler® オブザーバビリティエクスポーター	6
サポートされているエンドポイント	6
概要	6
Zipkinによる分散トレーシングのサポート	6
トランザクションの収集とストリーミングのサポート	7
時系列データのサポート	7
NetScaler Observability Exporterの仕組み	7
NetScaler Observability Exporterを使用したZipkinによる分散トレース	7
トランザクションエンドポイントとしてのElasticsearchを備えたNetScaler Observability Exporter	8
エンドポイントとしてのKafkaを備えたNetScaler Observability Exporter	8
時系列データのエンドポイントとしてのPrometheusを備えたNetScaler Observability Exporter	8
エンドポイントとしてのSplunk Enterpriseを備えたNetScaler Observability Exporter	8
展開	8
機能	9
カスタムヘッダーロギング	9
Elasticsearchサポートの機能強化	9

NetScaler Observability Exporter をデプロイする	10
NetScaler Observability Exporter と Zipkin をエンドポイントとして使用	11
ネットスケーラー オブザーバビリティ エクスポートの展開	12
前提条件	12
トレースが有効なアプリケーションをデプロイする	12
NetScaler Observability Exporterのサポート付きでNetScaler CPXをデプロイする	13
YAMLファイルを使用してZipkin、Elasticsearch、およびKibanaをデプロイする	14
YAML ファイルを使用して NetScaler Observability Exporter をデプロイする	15
NetScaler Observability Exporter のデプロイを検証する	15
ネットスケーラー オブザーバビリティ エクスポート と プロメテウス および グラファナ	17
ネットスケーラー オブザーバビリティ エクスポート の展開	17
前提条件	17
サンプルアプリケーションを展開する	18
NetScaler Observability Exporter のサポートが有効な NetScaler CPX を展開する	19
YAMLファイルを使用してPrometheusとGrafanaをデプロイする	19
YAMLファイルを使用してNetScaler Observability Exporterを展開する	20
メトリックをエクスポートするようにNetScalerを構成する（オプション）	21
テレメトリデータをプルするようにPrometheusを構成する（オプション）	22
Grafanaを構成する	22
Grafanaの視覚化を作成する	23
Grafana用の事前構築済みダッシュボードをインポートする	23
NetScaler Observability Exporter と Elasticsearch をエンドポイントとして	24
NetScaler Observability Exporter を展開する	24
前提条件	24
NetScaler Observability Exporterサポートを使用してNetScaler CPXを展開する	25
YAMLファイルを使用してElasticsearchとKibanaを展開する	26
YAMLファイルを使用してNetScaler Observability Exporterをデプロイする	27
NetScaler Observability Exporterのデプロイを検証する	27

NetScalerを複数のNetScaler Observability Exporterインスタンスと手動で統合する	28
エンドポイントとしてKafkaを使用するNetScaler Observability Exporter	33
ネットスケーラー オブザーバビリティ エクスポート の展開	33
前提条件	33
NetScaler Observability Exporterサポート付きでNetScaler CPXを展開する	35
YAML を使用して NetScaler Observability Exporter をデプロイする	35
NetScaler Observability Exporterの展開を確認する	41
複数のNetScaler Observability ExporterインスタンスをNetScalerに手動で統合する	42
NetScaler Observability ExporterからKafkaへのJSON形式でのトランザクションエクスポート のサポート	44
NetScaler Observability Exporter と Splunk Enterprise をエンドポイントとして	47
NetScaler Observability Exporter を展開する	47
前提条件	47
NetScaler Observability Exporterのサポートを使用してNetScaler CPXをデプロイする	49
YAML ファイルを使用して NetScaler Observability Exporter を展開する	50
NetScaler Observability Exporter の展開を検証する	52
Splunk 用の事前構築済みダッシュボードをインポートする	52
NetScaler Observability Exporter のトラブルシューティング	54
設定パラメータの説明	56
追加情報	60
コンテナロギングのサポート	62
ログの説明	62
エラーログ	63
情報ログ	64
警告ログ	67
デバッグログ	67
fix	84

リリースノート

リリースノートには、特定のビルドで導入された新機能と機能強化、およびビルドで修正された問題が記載されています。

リリース 1.10.001

修正された問題

- Kafka の `ServerUrl` オプションが複数のブートストラップブローカーを受け入れない問題を修正しました。この修正により、`ServerUrl` オプションはコンマ区切りのブートストラップ Kafka ブローカーのリストを受け入れるようになりました。例:

`X.X.X.X:9092,Y.Y.Y.Y:9092`。

リリース 1.9.001

新機能

DEBUG 重大度レベルのサポート

新しい重大度レベルである DEBUG がサポートされるようになりました。DEBUG 重大度レベルは、致命的、エラー、情報、デバッグメッセージを含む包括的なコンテナログを提供します。

多数のログがサポートされるようになりました。ログの完全なリストについては、[ログの説明](#)を参照してください。

監査ログとイベントの Kafka へのエクスポートのサポート

NetScaler® Observability Exporter は、NetScaler イベントと監査ログを JSON 形式で Kafka にエクスポートできるようになりました。

修正された問題

- SSL 経由の Splunk エクスポートにおけるデータ損失の問題を修正しました。この修正の一環として、新しいフィールド `ConnectionPoolSize` が導入されました。
`ConnectionPoolSize` と `MaxConnections` を使用して、データのエクスポートレートを制御できます。これらのフィールドの詳細については、[構成パラメーターの説明](#)を参照してください。

NetScaler® オブザーバビリティエクスポート

NetScaler Observability Exporterは、NetScalerからメトリクスとトランザクションを収集し、サポートされているエンドポイント向けに適切な形式（JSON、AVROなど）に変換するコンテナです。NetScaler Observability Exporterによって収集されたデータを、目的のエンドポイントにエクスポートできます。エンドポイントにエクスポートされたデータを分析することで、NetScalerによってプロキシされたアプリケーションのマイクロサービスレベルで貴重な洞察を得ることができます。

サポートされているエンドポイント

NetScaler Observability Exporterは現在、以下のエンドポイントをサポートしています。

- [ジブキン](#)
- [カフカ](#)
- [エラスティックサーチ](#)
- [プロメテウス](#)
- [スプリング エンタープライズ](#)

概要

Zipkinによる分散トレーシングのサポート

マイクロサービスアーキテクチャでは、単一のエンドユーザーリクエストが複数のマイクロサービスにまたがることもあり、トランザクションの追跡やエラーの原因特定は困難です。そのような場合、従来のパフォーマンス監視方法では、障害が発生した場所やパフォーマンス低下の原因を正確に特定することはできません。リクエストを処理する各マイクロサービスに固有のデータポイントをキャプチャし、それらを分析して意味のある洞察を得る方法が必要です。

分散トレーシングは、トランザクションをエンドツーエンドで追跡し、複数のマイクロサービス間でどのように処理されているかを理解する方法を提供することで、この課題に対処します。

[OpenTracing](#)は、分散トレーシングを設計および実装するための仕様と標準APIセットです。分散

トレーサーを使用すると、マイクロサービス間のデータフローを視覚化し、マイクロサービスアーキテクチャのボトルネックを特定するのに役立ちます。

NetScaler Observability ExporterはNetScaler向けに分散トレーシングを実装しており、現在、分散トレーサーとして[Zipkin](#)をサポートしています。

現在、NetScalerを使用してアプリケーションレベルでパフォーマンスを監視できます。NetScaler Observability ExporterをNetScalerと併用することで、NetScaler CPX、MPX、またはVPXによってプロキシされた各アプリケーションのマイクロサービスに関するトレーシングデータを取得できます。

トランザクションの収集とストリーミングのサポート

NetScaler Observability Exporterは、トランザクションを収集し、エンドポイントにストリーミングすることをサポートしています。現在、NetScaler Observability Exporterは、トランザクションエンドポイントとしてElasticsearchとKafkaをサポートしています。

時系列データのサポート

NetScaler Observability Exporterは、NetScalerインスタンスから時系列データ（メトリクス）を収集し、Prometheusにエクスポートすることをサポートしています。Prometheusは、メトリクスなどの時系列データを保存するための監視ソリューションです。その後、PrometheusをGrafanaのデータソースとして追加し、NetScalerのメトリクスをグラフィカルに表示して分析できます。

NetScaler Observability Exporterの仕組み

NetScaler Observability Exporterを使用したZipkinによる分散トレース

Logstreamは、NetScalerインスタンスからトランザクションを効率的に転送するためのトランスポートモードの1つとして使用される、Citrixが所有するプロトコルです。NetScaler Observability Exporterは、複数のNetScalerからLogstreamレコードとしてトレースデータを収集し、それらを集約します。NetScaler Observability Exporterは、データをトレーサーが理解できる形式に変換し、トレーサー（この場合はZipkin）にアップロードします。Zipkinの場合、データはZipkin固有のキー値を持つJSONに変換されます。

Zipkinユーザーインターフェイスを使用してトレースを表示できます。ただし、Zipkinと[Elasticsearch](#)および[Kibana](#)を使用することで、トレース分析を強化することもできます。Elasticsearchはトレースデータの長期保存を提供し、Kibanaはデータに対するより深い洞察を得ることを可能にします。

トランザクションエンドポイントとしてのElasticsearchを備えたNetScaler Observability Exporter

Elasticsearchがトランザクションエンドポイントとして指定されている場合、NetScaler Observability ExporterはデータをJSON形式に変換します。Elasticsearchサーバー上で、NetScaler Observability Exporterは各ADCのElasticsearchインデックスを1時間ごとに作成します。これらのインデックスは、データ、時間、ADCのUUID、およびHTTPデータのタイプ（http_eventまたはhttp_error）に基づいています。その後、NetScaler Observability Exporterは、各ADCのElasticsearchインデックスの下にJSON形式でデータをアップロードします。すべての通常のトランザクションはhttp_eventインデックスに配置され、異常はhttp_errorインデックスに配置されます。

エンドポイントとしてのKafkaを備えたNetScaler Observability Exporter

NetScaler Observability Exporterは、トランザクションを[Avro](#)またはJSONとしてKafkaにエクスポートします。監査ログとイベントはJSONとしてエクスポートされます。

時系列データのエンドポイントとしてのPrometheusを備えたNetScaler Observability Exporter

Prometheusが時系列データの形式として指定されている場合、NetScaler Observability ExporterはNetScalerからさまざまなメトリクスを収集し、適切なPrometheus形式に変換してPrometheusサーバーにエクスポートします。これらのメトリクスには、仮想サーバーのカウンター、分析プロファイルがバインドされているサービス、HTTP、TCPなどのグローバルカウンターが含まれます。

エンドポイントとしてのSplunk Enterpriseを備えたNetScaler Observability Exporter

Splunk Enterpriseがトランザクションエンドポイントとして指定されている場合、NetScaler Observability Exporterはインデックス、監査ログ、およびイベントを収集し、Splunk Enterpriseにエクスポートします。Splunk Enterpriseはインデックスをキャプチャし、リポジトリ内のリアルタイムデータを関連付け、そこからレポート、グラフ、ダッシュボード、および視覚化を生成できます。Splunk Enterpriseはこれらのデータのグラフィカルな表現を提供します。

展開

Kubernetes YAMLを使用してNetScaler Observability Exporterを展開できます。Kubernetes YAMLを使用してNetScaler Observability Exporterを展開するには、[展開](#)を参照してください。Helm

チャートを使用してNetScaler Observability Exporterを展開するには、[Helmチャートを使用した展開](#)を参照してください。

機能

カスタムヘッダーロギング

カスタムヘッダーロギングにより、トランザクションのすべてのHTTPヘッダーのロギングが可能になり、現在Kafkaエンドポイントでサポートされています。詳細については、[カスタムヘッダーロギング](#)を参照してください。

Elasticsearchサポートの機能強化

NetScaler Observability Exporterリリース1.2.001以降、NetScaler Observability ExporterがデータをElasticsearchサーバーに送信する際、一部のフィールドが文字列形式で利用可能になりました。また、Elasticsearchのインデックス構成オプションも追加されました。文字列形式のフィールドとElasticsearchインデックスの構成方法の詳細については、[Elasticsearchサポートの機能強化](#)を参照してください。

NetScaler Observability Exporter をデプロイする

このトピックでは、Kubernetes YAML ファイルを使用して NetScaler Observability Exporter をデプロイする方法に関する情報を提供します。

注:

Kubernetes YAML ファイルまたは Helm チャートを使用して NetScaler Observability Exporter をデプロイできます。NetScaler Operator を使用して NetScaler Observability Exporter をデプロイすることもできます。デプロイ手順については、[NetScaler Operator を使用した NetScaler Observability Exporter のデプロイ](#)を参照してください。

NetScaler のデプロイに応じて、NetScaler Observability Exporter を使用して、NetScaler CPX、MPX、または VPX からメトリックとトランザクションをエクスポートできます。

次の図は、サポートされているすべてのエンドポイントを含む NetScaler Observability Exporter のデプロイを示しています。



NetScaler Observability Exporter は、Kafka、Elasticsearch、Prometheus、Zipkin の次のエンドポイントをサポートしています。必要なエンドポイントに応じて、そのエンドポイントで NetScaler Observability Exporter をデプロイできます。

必要なエンドポイントに基づいて、次のいずれかのデプロイ手順を使用できます。

- [Zipkin を使用した NetScaler Observability Exporter のデプロイ](#)
- [Prometheus を使用した NetScaler Observability Exporter のデプロイ](#)
- [Elasticsearch を使用した NetScaler Observability Exporter のデプロイ](#)
- [Kafka を使用した NetScaler Observability Exporter のデプロイ](#)
- [Splunk を使用した NetScaler Observability Exporter のデプロイ](#)

NetScaler Observability Exporter と Zipkin をエンドポイントとして使用

NetScaler Observability Exporter は、エンドポイントとして [Zipkin](#) を使用して OpenTracing (OpenTracing は現在 OpenTelemetry の一部です) をサポートしています。NetScaler Observability Exporter は、NetScaler から収集されたトレースデータを OpenTracing に適したサポートされている形式に変換し、Zipkin にエクスポートします。Zipkin は、マイクロサービスアーキテクチャにおける遅延の問題をトラブルシューティングするために必要なタイミングデータを収集するのに役立つ分散トレースシステムです。Elasticsearch はトレースデータの長期保存に使用され、トレースは Zipkin UI または Kibana を使用して視覚化できます。

次の図は、Zipkin アーキテクチャの仕組みを示しています。

1. トレースが有効になっている場合、最初に、次のマイクロサービスポッドにパケットを転送する前に、追加のオープン・トレース・ヘッダー (`x-trace-id` 、 `x-span-id` 、および `x-parent-span-id`) を HTTP パケットに追加します。
2. この通信またはトランザクションに関する情報は、NetScaler Observability Exporter にプッシュされます。この情報には、ヘッダーの詳細、タイムスタンプ (このリクエストが開始された時刻とプロセス全体の期間)、およびアノテーション (アノテーションには、そのリクエストに関連付けられた HTTP、SSL、および TCP が含まれます) が含まれます。
3. その後、NetScaler Observability Exporter はすべての NetScaler から複数のトレースメッセージを受信し、それらを Zipkin が理解できる JSON 形式に集約し、API を介して Zipkin にプッシュします。
4. 同様に、マイクロサービスでトレースが有効になっている場合、そのトレースは API を介して Zipkin に送信されます。
5. Zipkin API はトレースデータを Elasticsearch データベースに保存し、最終的に完全なトレースを与えられた HTTP リクエストに結合し、Kibana などの視覚化ツールで視覚化します。各マイクロサービスでリクエストが費やした時間を確認できます。



NetScaler オブザーバビリティ エクスポーターの展開

NetScaler の展開に基づいて、NetScaler Observability Exporter を Kubernetes クラスターの外部または内部に展開できます。NetScaler Observability Exporter を Kubernetes クラスター内のポッドとして展開するか、クラスター外の NetScaler MPX または VPX フォームファクターで構成を有効にすることができます。NetScaler が提供する Kubernetes YAML ファイルを使用して NetScaler Observability Exporter を展開できます。

次の図は、NetScaler Ingress Controller をサイドカーとして持つインGRESSゲートウェイとしての NetScaler を示しています。NetScaler Observability Exporter は、NetScaler から収集されたトレースデータを Zipkin API に送信します。その後、トレースデータは Elasticsearch サーバーにアップロードされます。Elasticsearch から、データは視覚化のために Zipkin UI または Kibana UI に送信されます。

NetScaler オブザーバビリティ エクスポーターの展開(/en-us/netScaler-observability-exporter/media/coe-zipkin-deployment.png)

前提条件

- `kube-dns` または `CoreDNS` アドオンが有効になっている Kubernetes クラスターがあることを確認してください。

NetScaler Observability ExporterをZipkinとともにデプロイするには、以下のタスクを実行する必要があります。

1. トレースサポートを有効にして、必要なアプリケーションをデプロイします。
2. NetScaler Observability Exporterサポートを有効にしてNetScaler CPXをデプロイします。
3. YAMLファイルを使用して、[Zipkin](#)、[Elasticsearch](#)、および[Kibana](#)をデプロイします。
4. YAMLファイルを使用してNetScaler Observability Exporterをデプロイします。

トレースが有効なアプリケーションをデプロイする

以下は、トレースが有効なアプリケーションデプロイの例です。

注:

事前デプロイされたWebアプリケーションがある場合は、手順1と2をスキップしてください。

1. 独自の証明書とキーを使用して、シークレット`ingress.crt`とキー`ingress.key`を作成します。

この例では、デフォルトの名前空間に`ing`というシークレットが作成されます。

```
kubectl create secret tls ing --cert=ingress.crt --key=ingress.key
```

2. `watches-app-tracing.yaml`からYAMLファイルにアクセスして、アプリケーションをデプロイします。

```
kubectl create -f watches-app-tracing.yaml
```

3. イングレスのスマートアノテーションを使用して、アプリケーションのYAMLファイルのイングレスアノテーションで指定することにより、インポートする必要がある特定のパラメーターを定義します。

```
ingress.citrix.com/analyticsprofile: '{"webinsight":  
{"httpurl":"ENABLED", "httpuseragent":"ENABLED",  
"httpHost":"ENABLED", "httpMethod":"ENABLED", "httpContentType":"ENABLED"  
}}'
```

注: パラメーターは `watches-app-tracing.yaml` ファイルで事前定義されています。

アノテーションの詳細については、[Ingress annotations documentation](#)を参照してください。

NetScaler Observability Exporterのサポート付きでNetScaler CPXをデプロイする

NetScaler Observability Exporterのサポートを有効にしたNetScaler CPXをデプロイできます。

NetScaler CPXをデプロイする際に、デプロイメントYAMLファイル `cpx-ingress-tracing.yaml` を変更して、NetScaler Observability Exporterのサポートに必要な構成情報を含めることができます。

NetScaler Observability Exporterのサポート付きでNetScaler CPXインスタンスをデプロイするには、次の手順を実行します。

1. `cpx-ingress-tracing.yaml` および `cic-configmap.yaml` ファイルをダウンロードします。
2. 必要なキーと値のペアでConfigMapを作成し、ConfigMapをデプロイします。特定のエンドポイントについては、[ディレクトリ](#)にある `cic-configmap.yaml` ファイルを使用できます。

- 必要に応じて、NetScaler CPX関連のパラメータを変更します。たとえば、`cpx-ingress-tracing.yaml` ファイルの `args` の下に次のように行を追加します。

```
args:

  - --configmap

    default/cic-configmap
```

- `cic-configmap.yaml` ファイルを編集して、`NS_ANALYTICS_CONFIG` エンドポイント構成でNetScaler Observability Exporterの次の変数を指定します。

```
server: 'coe-zipkin.default.svc.cluster.local' # COE service FQDN
```

- 次のコマンドを使用して、NetScaler Observability Exporterのサポート付きでNetScaler CPXをデプロイします。

```
kubectl create -f cpx-ingress-tracing.yaml

kubectl create -f cic-configmap.yaml
```

注:

`default` 以外の名前空間を使用している場合は、`coe-zipkin.default.svc.cluster.local` を `coe-zipkin.<desired-namespace>.svc.cluster.local` に変更します。ADCがKubernetesクラスターの外部にある場合、NetScaler Observability ExporterのIPアドレスとノードポートアドレスを指定する必要があります。

YAMLファイルを使用してZipkin、Elasticsearch、およびKibanaをデプロイする

YAMLを使用してZipkin、Elasticsearch、およびKibanaをデプロイするには、次の手順を実行します。

- 次のYAMLファイルをダウンロードします。

- [zipkin.yaml](#)

- [Elasticsearch.yaml](#)
- [kibana.yaml](#)

2. *default* 以外のカスタム名前空間を使用したい場合は、名前空間の定義を編集します。
3. Zipkin、Elasticsearch、および Kibana をデプロイするには、次のコマンドを実行します。

```
kubectl create -f zipkin.yaml  
  
kubectl create -f elasticsearch.yaml  
  
kubectl create -f kibana.yaml
```

注:

Zipkin、Elasticsearch、および Kibana は、同じ Kubernetes クラスターのデフォルトの名前空間にデプロイされます。

YAML ファイルを使用して NetScaler Observability Exporter をデプロイする

YAML ファイルを使用して NetScaler Observability Exporter をデプロイできます。 [coe-zipkin.yaml](#) ファイルをダウンロードします。

Kubernetes YAML を使用して NetScaler Observability Exporter をデプロイするには、Elasticsearch エンドポイントで次のコマンドを実行します。

```
kubectl create -f coe-zipkin.yaml
```

注:

デフォルト以外のカスタム名前空間を使用している場合は、NetScaler Observability Exporter の YAML ファイルを変更します。

NetScaler Observability Exporter のデプロイを検証する

NetScaler Observability Exporter のデプロイを検証するには、次の操作を実行します。

1. 次のコマンドを使用してアプリケーションにリクエストを送信し、デプロイを検証します。

```
kubectl run -i --tty busybox --image=busybox --restart=Never --rm --  
wget --no-check-certificate "https://cpx-ingress-  
zipkin.default.svc.cluster.local/serial/view/watches"
```

2. Kubernetes ノードの IP アドレスとノードポートを使用して、Zipkin ユーザーインターフェイスを開きます。

```
http://*k8-node-ip-address*:*node-port*/
```

以下の画像で、*Watches* アプリケーションのトレースを表示できます。*Watches* アプリケーションには、各ウォッチタイプに対応する複数のマイクロサービスがあり、互いに通信してアプリケーションデータを提供します。トレースデータは、アプリケーション **FASTTRACK** が他のマイクロサービスと比較して、サービスの提供により多くの時間を要したことを示しています。このようにして、パフォーマンスの低いワークロードを特定し、トラブルシューティングを行うことができます。



Kibanaダッシュボードでも生データを表示できます。 `http://<node-ip>:<node-port>` を使用してKibanaを開き、*zipkin* インデックスパターンを定義することから始めます。

`timestamp_millis` フィールドをタイムスタンプフィールドとして使用します。インデックスパターンを作成したら、**Discover** タブをクリックすると、Zipkinによって収集されたトレース情報を表示できます。



NetScaler Observability Exporterに関連するトラブルシューティングについては、[NetScaler CPXのトラブルシューティング](#) を参照してください。

ネットスケラー オブザーバビリティ エクスポート と プロメテウス および グラファナ

NetScaler Observability ExporterからデータをプルするためのエンドポイントとしてPrometheusを設定できます。また、Grafanaを設定して、同じデータをグラフィカルに視覚化することもできます。

NetScaler Observability Exporterには、Prometheusからのプルリクエストに基づいてメトリクスを提供するプッシュゲートウェイサーバーがあり、ポート5563でリッスンしています。NetScaler Observability Exporterは、時系列データをPrometheusにエクスポートします。

PrometheusとGrafanaのアーキテクチャ(/en-us/netscaler-observability-exporter/media/coe-prometheus-grafana-architecture.png)

ネットスケラー オブザーバビリティ エクスポート の展開

YAMLファイルを使用してNetScaler Observability Exporterを展開できます。NetScalerの展開に応じて、NetScaler Observability ExporterをKubernetesクラスターの外部または内部に展開します。NetScaler Observability Exporterは、Kubernetesクラスター内のポッドとして、またはクラスター外のNetScaler MPXまたはVPXアプライアンスに展開できます。

前提条件

- kube-dnsまたはCoreDNSアドオンが有効になっているKubernetesクラスターがあることを確認してください。

Prometheusエンドポイントを使用してNetScaler Observability Exporterを展開するには、次のタスクが含まれます。

- サンプルアプリケーションの展開
- NetScaler Observability Exporterのサポートが有効になっているNetScaler CPXの展開
- YAMLファイルを使用したPrometheusとGrafanaの展開
- YAMLファイルを使用したNetScaler Observability Exporterの展開

- NetScalerを構成してメトリクスをエクスポートする（オプション）
- Prometheusを構成してテレメトリデータをプルする（オプション）
- Grafana を構成する
- Grafana の視覚化を作成する

サンプルアプリケーションを展開する

以下は、サンプルWebサーバーアプリケーションを展開するための手順の例です。

注:

事前展開されたWebアプリケーションがある場合は、手順1から手順3までのステップをスキップしてください。

1. 独自の証明書とキーを使用して、シークレット `ingress.crt` とキー `ingress.key` を作成します。

この例では、デフォルトの名前空間で `ing` と呼ばれるシークレットが作成されます。

```
kubectl create secret tls ing --cert=ingress.crt --key=ingress.key
```

2. アプリケーションを展開するには、[webserver-es.yaml](#) からYAMLファイルにアクセスします。

```
kubectl create -f webserver-es.yaml
```

3. イングレスのスマートアノテーションを使用して、アプリケーションのYAMLファイルのイングレスアノテーションで指定することにより、インポートする必要がある特定のパラメーターを定義します。

```
ingress.citrix.com/analyticsprofile: '{"webinsight":  
{"httpurl":"ENABLED", "httpuseragent":"ENABLED",  
"httpHost":"ENABLED", "httpMethod":"ENABLED", "httpContentType":"ENABLED"  
}}'
```

注:

パラメーターは `webserver-es.yaml` ファイルで事前定義されています。

アノテーションの詳細については、[Ingress annotations documentation](#) を参照してください。

NetScaler Observability Exporter のサポートが有効な NetScaler CPX を展開する

NetScaler CPX を、NetScaler Ingress Controller とともに NetScaler Observability Exporter のサポートが有効なサイドカーとして展開できます。NetScaler CPX YAML ファイル `cpx-ingress-es.yaml` を変更して、NetScaler Observability Exporter のサポートに必要な構成情報を含めることができます。

以下に、アプリケーションのデプロイ手順の例を示します。

1. `cpx-ingress-prometheus.yaml` と `cic-configmap.yaml` ファイルをダウンロードします。
2. 必要なキーと値のペアを持つ ConfigMap を作成し、ConfigMap をデプロイします。特定のエンドポイントについては、[directory](#) で利用可能な `cic-configmap.yaml` ファイルを使用できます。
3. 必要に応じて、NetScaler CPX 関連のパラメータを変更します。
4. `cic-configmap.yaml` ファイルを編集し、`NS_ANALYTICS_CONFIG` エンドポイント構成で NetScaler Observability Exporter の以下の変数を指定します。

```
server: 'coe-prometheus.default.svc.cluster.local' # COE service FQDN
```

注:

`default` 以外の名前空間を使用した場合は、`coe-prometheus.default.svc.cluster.local` を `coe-prometheus.<desired-namespace>.svc.cluster.local` に変更します。

5. 以下のコマンドを使用して、NetScaler Observability Exporter をサポートする NetScaler CPX をデプロイします。

```
kubectl create -f cpx-ingress-prometheus.yaml

kubectl create -f cic-configmap.yaml
```

YAML ファイルを使用して Prometheus と Grafana をデプロイする

YAML ファイルを使用して Prometheus と Grafana をデプロイするには、以下の手順を実行します。

1. [prometheus-grafana.yaml](#) から `Prometheus-Grafana` YAMLファイルをダウンロードします。
2. `default` 以外の名前空間を使用したい場合は、名前空間の定義を編集します。
3. PrometheusとGrafanaをデプロイするには、以下のコマンドを実行します。

```
kubectl create -f prometheus-grafana.yaml
```

注:

PrometheusとGrafanaは、同じKubernetesクラスターのデフォルトの名前空間にデプロイされます。

YAMLファイルを使用してNetScaler Observability Exporterを展開する

YAMLファイルを使用してNetScaler Observability Exporterを展開できます。YAMLファイルは[coe-prometheus.yaml](#)からダウンロードしてください。

- NetScaler Observability Exporterバージョン1.3.001およびそれ以前のバージョンでは、`coe-prometheus.yaml` YAMLファイルで提供されているConfigMap構成を使用できます。
- NetScaler Observability Exporterバージョン1.4.001では、展開前に `coe-prometheus.yaml` ファイル内のConfigMapを次のように変更する必要があります。

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: coe-config-prometheus

data:

  lstreamd_default.conf: |

    {

    "Endpoints": {
```

```
"ZIPKIN": {  
  
    "ServerUrl": "http://0.0.0.0:0",  
  
    "RecordType": {},  
  
    "PrometheusMode": "yes"  
  
}  
  
}  
  
}
```

Kubernetes YAMLを使用してNetScaler Observability Exporterを展開するには、次のコマンドを実行します。

```
kubectl create -f coe-prometheus.yaml
```

注:

カスタム名前空間を使用している場合は、NetScaler Observability ExporterのYAMLファイルを変更してください。

メトリックをエクスポートするようにNetScalerを構成する（オプション）

注:

NetScaler Ingress Controllerを使用してNetScalerを構成しない場合は、NetScalerで次の手動構成を実行できます。

NetScalerを手動で構成して、メトリックをNetScaler Observability Exporterにエクスポートできます。NetScaler Observability ExporterのIP/FQDNアドレスをHTTPサービスとして指定し、デフォルトの `ns_analytics_time_series_profile` 分析プロファイルと組み合わせます。メトリックのエクスポートを有効にし、出力モードをPrometheusに設定します。

以下は構成例です。

```
add server COE_instance 192.168.1.102

add service coe_metric_collector_svc_192.168.1.102 COE_instance HTTP 5563

set analytics profile ns_analytics_time_series_profile -collector
coe_metric_collector_svc_192.168.1.102 -Metrics ENABLED -OutputMode
Prometheus
```

テレメトリデータをプルするようにPrometheusを構成する（オプション）

Prometheusサービスは、[Quayコンテナレジストリ](#)およびDocker HubでDockerイメージとして利用できます。

Prometheusを起動し、ポート9090で公開するには、次のコマンドを実行します。

```
docker run -p 9090:9090 prom/prometheus
```

NetScaler Observability Exporterをスクレイプターゲットとして手動で追加するには、`prometheus.yaml` ファイルを編集します。YAMLファイルで、NetScaler Observability ExporterのIP/FQDNアドレスとポート5563をスクレイプターゲットとして指定します。

```
scrape_configs:

  - job_name: coe

    static_configs:

      - targets: ['192.168.1.102:5563']
```

Grafanaを構成する

現在のデプロイでは、Prometheusサーバーはすでにデータソースとして追加されています。既存のPrometheusサーバーをデプロイに使用する場合は、Grafanaにデータソースとして追加されていることを確認してください。詳細については、[Grafana support for Prometheus](#)を参照してください。

Grafanaの視覚化を作成する

Grafanaダッシュボードを作成し、データに適した主要なメトリックと視覚化タイプを選択できます。

次の手順は、ADC CPUメトリックをGrafanaパネルに追加する方法を示しています。

1. パネルタイトルをADC CPUとして指定します。
2. クエリタブで、クエリAのメトリックを`cpu_use`として指定します。
3. 設定タブで、**視覚化タイプ**を選択します。

Grafanaでデータとその表現を変更できます。詳細については、[Grafana Documentation](#)を参照してください。



Grafana用の事前構築済みダッシュボードをインポートする

事前構築済みダッシュボードをGrafanaにインポートすることもできます。利用可能な[ダッシュボード](#)を参照してください。

NetScaler Observability Exporterに関連するトラブルシューティングについては、[NetScaler CPX troubleshooting](#)を参照してください。

NetScaler Observability Exporter と Elasticsearch をエンドポイントとして

NetScaler Observability Exporter は、NetScaler からメトリクスとトランザクションを収集するコンテナです。データをサポートされている形式（JSONなど）に変換し、エンドポイントとして Elasticsearch にエクスポートします。Elasticsearch は Lucene ライブラリに基づいた検索エンジンです。HTTP Web インターフェイスとスキーマフリーの JSON ドキュメントを備えた、分散型でマルチテナント対応の全文検索エンジンを提供します。

NetScaler Observability Exporter を展開する

YAML ファイルを使用して NetScaler Observability Exporter を展開できます。NetScaler の展開に応じて、NetScaler Observability Exporter を Kubernetes クラスターの外部または内部に展開できます。NetScaler Observability Exporter は、Kubernetes クラスター内のポッドとして、またはクラスター外の NetScaler MPX または VPX アプライアンスに展開できます。

次の図は、NetScaler を Ingress Gateway として、NetScaler Ingress Controller と NetScaler Observability Exporter をサイドカーとして示しています。NetScaler Observability Exporter は、NetScaler アプリケーションのメトリクスとトランザクションデータを Elasticsearch に送信し、同じデータを Kibana にエクスポートします。Kibana はデータのグラフィカルな表現を提供します。

ネットスケーラー オブザーバビリティ エクスポートの展開(/en-us/netscaler-observability-exporter/media/coe-es-architecture.png)

前提条件

- Kubernetes クラスターで `kube-dns` または `CoreDNS` アドオンが有効になっていることを確認してください。

以下の手順では、YAML ファイルを使用して Kubernetes の `default` 名前空間に NetScaler Observability Exporter を展開します。`default` 以外のプライベート名前空間に展開する場合は、YAML ファイルを編集して名前空間を指定してください。

以下は、アプリケーション展開手順の例です。

注:

Web アプリケーションがすでに展開されている場合は、手順 1 と 2 をスキップしてください。

1. 独自の証明書とキーを使用して、シークレット `ingress.crt` とキー `ingress.key` を作成します。

この例では、デフォルトの名前空間に `ing` という名前のシークレットが作成されます。

```
kubectl create secret tls ing --cert=ingress.crt --key=ingress.key
```

2. アプリケーションを展開するには、[webserver-es.yaml](#) から YAML ファイルにアクセスします。

```
kubectl create -f webserver-es.yaml
```

3. アプリケーションのYAMLファイルのイングレスアノテーションで、イングレスのスマートアノテーションを使用して、インポートする必要がある特定のパラメータを定義します。

```
ingress.citrix.com/analyticsprofile: '{"webinsight":  
{"httpurl":"ENABLED", "httpuseragent":"ENABLED",  
"httpHost":"ENABLED", "httpMethod":"ENABLED", "httpContentType":"ENABLED"  
}}'
```

注:

パラメータは `webserver-es.yaml` ファイルで事前定義されています。

アノテーションの詳細については、[Ingress annotations documentation](#)を参照してください。

NetScaler Observability Exporterサポートを使用して NetScaler CPXを展開する

NetScaler Ingress ControllerとともにNetScaler Observability Exporterサポートを有効にして、NetScaler CPXをサイドカーとして展開できます。NetScaler Observability Exporterサポートに必要な構成情報を含めるために、NetScaler CPX YAMLファイル `cpx-ingress-es.yaml` を変更できます。

NetScaler Observability Exporterサポートを使用してNetScaler CPXインスタンスを展開するには、次の手順を実行します。

1. [cpx-ingress-es.yaml](#)と[cic-configmap.yaml](#)ファイルをダウンロードします。
2. 必要なキーと値のペアを持つConfigMapを作成し、ConfigMapを展開します。特定のエンドポイントで[directory](#)にある `cic-configmap.yaml` ファイルを使用できます。
3. 必要に応じて、NetScaler CPX関連のパラメータを変更します。
4. `cic-configmap.yaml` ファイルを編集し、`NS_ANALYTICS_CONFIG` エンドポイント構成でNetScaler Observability Exporterの次の変数を指定します。

```
server: 'coe-es.default.svc.cluster.local' # COE service FQDN
```

注:

`default`以外の名前空間を使用している場合は、`coe-es.default.svc.cluster.local` `to` を `coe-es.<desired-namespace>.svc.cluster.local` に変更します。ADCがKubernetesクラスターの外部にある場合は、NetScaler Observability ExporterのIPアドレスとノードポートアドレスを指定する必要があります。

5. 次のコマンドを使用して、NetScaler Observability ExporterサポートでNetScaler CPXを展開します。

```
kubectl create -f cpx-ingress-es.yaml

kubectl create -f cic-configmap.yaml
```

YAMLファイルを使用してElasticsearchとKibanaを展開する

1. Elasticsearch YAMLファイルを[elasticsearch.yaml](#)から、Kibana YAMLファイルを[kibana.yaml](#)からダウンロードします。
2. `default`以外の名前空間を使用する場合は、名前空間の定義を編集します。
3. ElasticsearchとKibanaをデプロイするには、次のコマンドを実行します。

```
kubectl create -f elasticsearch.yaml
```

```
kubectl create -f kibana.yaml
```

注: ElasticsearchとKibanaは、同じKubernetesクラスターのデフォルトの名前空間にデプロイされます。

YAMLファイルを使用してNetScaler Observability Exporterをデプロイする

YAMLファイルを使用してNetScaler Observability Exporterをデプロイできます。 [coe-es.yaml](#)からYAMLファイルをダウンロードします。

Kubernetes YAMLを使用してNetScaler Observability Exporterをデプロイするには、Elasticsearchエンドポイントで次のコマンドを実行します。

```
kubectl create -f coe-es.yaml
```

注:

カスタム名前空間を使用している場合は、NetScaler Observability ExporterのYAMLファイルを変更します。

NetScaler Observability Exporterのデプロイを検証する

NetScaler Observability Exporterのデプロイを検証するには、次の手順を実行します。

1. 次のコマンドを使用してデプロイを検証します。

```
kubectl get deployment,pods,svc -o wide
```



2. ブラウザを使用してURL: `https://kubernetes-node-IP:cpx-ingress-es nodeport/` でアプリケーションにアクセスします。

たとえば、ステップ1から `<http://10.102.40.41:30176/>` にアクセスします。ここで、`10.102.40.41` はKubernetesノードIPの1つです。



3. ブラウザを使用してKibanaにアクセスします。URL: `https://<kubernetes-node-IP>:<kibana nodeport>/`。

例えば、ステップ1から `<http://10.102.40.41:32529/>` にアクセスします。ここで、`10.102.40.41` はKubernetesノードのIPの1つです。

- a. 「自分で探索」をクリックします。



- b. 「Elasticsearchインデックスに接続」をクリックします。



- c. 「保存済みオブジェクト」をクリックします。

- d. [KibanaAppTrans.ndjson](#)からKibanaダッシュボードをダウンロードしてインポートします。



- e. 「アプリトランザクションダッシュボード」をクリックします。



ダッシュボードが表示されます。



NetScalerを複数のNetScaler Observability Exporterインスタンスと手動で統合する

NetScaler Observability Exporterは手動で設定することもできます。前のセクションで説明したように、YAMLファイルを使用してNetScaler Observability Exporterを自動的にデプロイすることをお勧めします。MPXおよびVPXフォームファクターのNetScalerについても手動設定を実行できます。

```
enable feature appflow®

enable ns mode ULFD

add dns nameserver <KUBE-CoreDNS>

add server COEsvr <FQDN/IP>
```

```
add servicegroup COEsvcgrp LOGSTREAM -autoScale™ DNS

bind servicegroup COEsvcgrp COEsvr <PORT>

add lb vserver COE LOGSTREAM 0.0.0.0 0

bind lb vserver COE COEsvcgrp

add analytics profile web_profile -collectors COE -type
webinsight -httpURL ENABLED -httpHost ENABLED -httpMethod ENABLED
-httpUserAgent ENABLED -httpContentType ENABLED

add analytics profile tcp_profile -collectors COE -type
tcpinsight

bind lb vserver <WEB-VSERVER> -analyticsProfile web_profile

bind lb vserver <WEB-VSERVER> -analyticsProfile tcp_profile

# To enable metrics push to prometheus

add service metrictest_SVC <IP> HTTP <PORT>

set analyticsprofile ns_analytics_time_series_profile
-collectors metrictest_SVC -metrics ENABLED -outputMode prometheus
```

FQDNを使用してNetScaler Observability Exporterを追加する

```
enable feature appflow

enable ns mode ULFD

add dns nameserver <KUBE-CoreDNS>

add server COEsvr <FQDN>

add servicegroup COEsvcgrp LOGSTREAM -autoScale DNS

bind servicegroup COEsvcgrp COEsvr <PORT>
```

```

add lb vserver COE LOGSTREAM 0.0.0.0 0

bind lb vserver COE COEsvcgrp

add analytics profile web_profile -collectors COE -type
webinsight -httpURL ENABLED -httpHost ENABLED -httpMethod ENABLED
-httpUserAgent ENABLED -httpContentType ENABLED

add analytics profile tcp_profile -collectors COE -type
tcpinsight

bind lb vserver <WEB-VSERVER> -analyticsProfile web_profile

bind lb vserver <WEB-VSERVER> -analyticsProfile tcp_profile

# To enable metrics push to prometheus

add service metrictest_SVC <IP> HTTP <PORT>

set analyticsprofile ns_analytics_time_series_profile
-collectors metrictest_SVC -metrics ENABLED -outputMode prometheus

```

NetScalerがアプリケーションデータログをNetScaler Observability Exporterに送信しているかどうかを確認するには：

```
nsconmsg -g lstream_tot_trans_written -d current
```

カウンター値は、NetScaler Observability Exporterに送信されたアプリケーショントランザクション（例：HTTPトランザクション）の数を示します。

カウンター値(/en-us/netScaler-observability-exporter/media/coe-es-counter-value.png)

NetScaler Observability Exporterに送信されるアプリケーショントラフィックレート（例：HTTP req/sec）が `lstream_tot_trans_written` と等しくない場合は、次のコマンドを使用して確認できます。

```
nsconmsg -g nslstream_err_ulf_data_not_sendable -d current
```

カウンター値は、ネットワーク輻輳、ネットワーク帯域幅の不足などにより、NetScalerがデータをNetScaler Observability Exporterに送信できないことを示します。データは利用可能なバッファに保存されます。

さまざまなトランザクションデータと個々のフィールド、およびそれらのデータ型に関する情報は、NetScalerの次の場所で入手できます。

```
shell/netscaler/appflow/ns_ipfix.yaml
```

データロケーション(/en-us/netscaler-observability-exporter/media/coe-es-shell-ipfix.png)

アプリケーショントランザクションレコードがNetScalerからNetScaler Observability Exporterにエクスポートされていることを確認するには、次のコマンドを使用します。

```
nsconmsg -g appflow_tmpl -d current
```

パフォーマンスデータ(/en-us/netscaler-observability-exporter/media/coe-es-appflow-current.png)

時系列データ用のメトリックデータエクスポートログのNetScalerへの保存場所：

```
/var/nslog/metrics_prom.log
```

Elasticsearch関連のカウンターを確認するには、次のコマンドを実行します。

```
kubectl exec -it <cpx-pod-name> [-c <cpx-container-name>] [-n  
<namespace-name>] -- bash
```

```
tail -f /var/ulflg/counters/lstrmd_counters_codes.log | grep -iE  
"(http_reqs_done|elk)"
```

NetScaler Observability Exporterの設定が正しく適用されていることを確認するには、次の場所にあるログを見つけてください。

```
vi /var/logproxy/lstreamd/conf/lstreamd.conf
```

NetScaler Observability Exporter が失敗した場合は、以下の場所にあるログとファイルを収集し、NetScaler サポートに連絡してください。

`/var/crash/` (Location of the coredump files, if any.)

`/var/ulfdlog/` (Location of the ``libulfd`` logs and counter details.)

`/var/log` (Location of the console logs, `lstreamd` logs and so on.)

NetScaler Observability Exporter に関連するトラブルシューティングについては、「[NetScaler CPX のトラブルシューティング](#)」を参照してください。

エンドポイントとしてKafkaを使用する NetScaler Observability Exporter

NetScaler Observability Exporterは、NetScalerからメトリックとトランザクションを収集するコンテナです。また、データをKafkaでサポートされている形式（AVROなど）に変換し、エンドポイントにエクスポートします。Kafkaは、高性能なデータパイプラインとストリーミング分析のためのオープンソースの分散イベントストリーミングプラットフォームです。

注記：

NetScalerはKafka APIのv0バージョンを使用しており、Kafkaバージョン3.Xまでサポートされています。Kafkaバージョン4.Xはサポートされていません。

ネットスケーラー オブザーバビリティ エクスポートの展開

NetScaler Observability ExporterはYAMLファイルを使用して展開できます。NetScalerの展開に基づいて、NetScaler Observability Exporterを使用してNetScalerからメトリックとトランザクションデータをエクスポートできます。NetScaler CPXは、Kubernetesクラスター内のポッドとして、またはクラスター外のNetScaler MPXまたはVPXフォームファクターとして展開できます。

次の図は、NetScalerをIngress Gatewayとして、NetScaler Observability Exporterをサイドカーとして使用する構成を示しています。これはNetScalerアプリケーションのトランザクションデータをKafkaに送信します。



前提条件

- Kubernetesクラスターで `kube-dns` または `CoreDNS` アドオンが有効になっていることを確認してください。
- Kafkaサーバーがインストールされ、構成されていることを確認してください。
- KafkaブローカーのIPアドレスまたはFQDNアドレスが必要です。
- Kafkaトピック `HTTP` を定義している必要があります。
- データを検証するためにKafka Consumerがあることを確認してください。

注記：

この例のシナリオでは、YAMLファイルを使用してNetScaler Observability ExporterをKubernetesのdefault名前空間に展開します。default以外のプライベートKubernetes名前空間に展開する場合は、YAMLファイルを編集して名前空間を指定してください。

以下は、アプリケーション展開手順のサンプルです。

注記：

ウェブアプリケーションがすでに展開されている場合は、ステップ1と2をスキップしてください。

1. 独自の証明書とキーを使用して、シークレット`ingress.crt`とキー`ingress.key`を作成します。

この例では、デフォルトの名前空間に`ing`というシークレットが作成されます。

```
kubectl create secret tls ing --cert=ingress.crt --key=ingress.key
```

2. サンプルアプリケーションを展開するには、`webserver-kafka.yaml`からYAMLファイルにアクセスします。

```
kubectl create -f webserver-kafka.yaml
```

3. アプリケーションのYAMLファイルの`ingress`アノテーションで、`ingress`のスマートアノテーションを使用して、インポートする必要がある特定のパラメーターを定義します。

```
ingress.citrix.com/analyticsprofile: '{"webinsight":  
{"httpurl":"ENABLED", "httpuseragent":"ENABLED",  
"httpHost":"ENABLED", "httpMethod":"ENABLED", "httpContentType":"ENABLED"  
}}'
```

注記：

パラメーターは `webserver-kafka.yaml` ファイルで事前定義されています。

アノテーションの詳細については、[Ingress annotations documentation](#)を参照してください。

NetScaler Observability Exporterサポート付きでNetScaler CPXを展開する

NetScaler Observability Exporterサポート付きでNetScaler CPXをサイドカーとして展開できます。NetScaler CPX YAML ファイル `cpx-ingress-kafka.yaml` を編集して、NetScaler Observability Exporterサポートに必要な構成情報を含めることができます。

NetScaler Observability Exporterサポート付きでNetScaler CPXインスタンスを展開するには、以下の手順を実行します。

1. `cpx-ingress-kafka.yaml`と`cic-configmap.yaml`ファイルをダウンロードします。
2. 必要なキーと値のペアを持つConfigMapを作成し、デプロイします。特定のエンドポイント向けに、[ディレクトリ](#)で利用可能な `cic-configmap.yaml` ファイルを使用できます。
3. 必要に応じて、NetScaler CPX 関連のパラメーターを変更します。
4. `cic-configmap.yaml` ファイルを編集し、`NS_ANALYTICS_CONFIG` エンドポイント構成で NetScaler Observability Exporter の次の変数を指定します。

```
server: 'coe-kafka.default.svc.cluster.local' # COE service FQDN
```

5. 次のコマンドを使用して、NetScaler Observability Exporter のサポート付きで NetScaler CPX をデプロイします。

```
kubectl create -f cpx-ingress-kafka.yaml  
  
kubectl create -f cic-configmap.yaml
```

注:

`default` 以外の名前空間を使用した場合は、`coe-kafka.default.svc.cluster.local` から `coe-kafka.<desired-namespace>.svc.cluster.local` に変更する必要があります。

YAML を使用して NetScaler Observability Exporter をデプロイする

YAML ファイルを使用して NetScaler Observability Exporter をデプロイできます。NetScaler Observability Exporter のデプロイに使用できる `coe-kafka.yaml` ファイルをダウンロードします。

Kubernetes YAML を使用して NetScaler Observability Exporter をデプロイするには、Kafka エンドポイントで次のコマンドを実行します。

```
kubect1 create -f coe-kafka.yaml
```

必要な変更のために YAML ファイルを編集するには、次の手順を実行します。

1. 次の YAML 定義を使用して ConfigMap を編集します。

注:

ブートストラップ Kafka ブローカーアドレス (コンマ区切り) と目的の Kafka トピックが指定されていることを確認してください。

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: coe-config-kafka

data:

  lstreamd_default.conf: |

    {

      "Endpoints": {

        "KAFKA": {

          "ServerUrl": "X.X.X.X:9092,Y.Y.Y.Y:9092", #Specify the
bootstrap Kafka broker addresses (separated by comma in case multiple
bootstrap brokers are to be configured)

          "KafkaTopic": "HTTP", #Specify the desired kafka topic

          "RecordType": {

            "HTTP": "all",
```

```
        "TCP": "all",

        "SWG": "all",

        "VPN": "all",

        "NGS": "all",

        "ICA®": "all",

        "APPFW": "none",

        "BOT": "none",

        "VIDEOOPT": "none",

        "BURST_CQA": "none",

        "SLA": "none",

        "MONGO": "none"

    },

    "ProcessAlways": "yes",

    "FileSizeMax": "40",

    "ProcessYieldTimeOut": "500",

    "FileStorageLimit": "1000",

    "SkipAvro": "no",

    "AvroCompress": "yes"

}

}
```

注: JSON 形式でトランザクションをエクスポートするには、[Kafka への JSON 形式でのトランザクションのエクスポート](#) を参照してください。

2. Kafka ノードのホスト名と IP または FQDN アドレスを指定します。3 ノード Kafka クラスターには、次の YAML 定義を使用します。

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: coe-kafka

  labels:

    app: coe-kafka

spec:

  replicas: 1

  selector:

    matchLabels:

      app: coe-kafka

  template:

    metadata:

      name: coe-kafka

      labels:

        app: coe-kafka

    spec:

      hostAliases:

        - ip: "X.X.X.X" # Here we specify kafka node1 Ipaddress
```

```
      hostnames:

        - "kafka-node1"

      - ip: "Y.Y.Y.Y" # Here we specify kafka node2 Ipaddress

      hostnames:

        - "kafka-node2"

      - ip: "Z.Z.Z.Z" # Here we specify kafka node3 Ipaddress

      hostnames:

        - "kafka-node3"

    containers:

      - name: coe-kafka

        image: "quay.io/citrix/citrix-observability-exporter:1.3.001"

        imagePullPolicy: Always

        ports:

          - containerPort: 5557

            name: lstream

        volumeMounts:

          - name: lstreamd-config-kafka

            mountPath: /var/logproxy/lstreamd/conf/
lstreamd_default.conf

            subPath: lstreamd_default.conf

          - name: core-data

            mountPath: /var/crash/
```

```
volumes:

  - name: lstreamd-config-kafka

    configMap:

      name: coe-config-kafka

  - name: core-data

    emptyDir: {}
```

3. 必要に応じて、以下のYAML定義を使用して、NetScaler Observability ExporterポートをNetScalerに公開するためのサービス構成を編集します。

Citrix-observability-exporterヘッドレスサービス:

```
apiVersion: v1

kind: Service

metadata:

  name: coe-kafka

  labels:

    app: coe-kafka

spec:

  clusterIP: None

  ports:

    - port: 5557

      protocol: TCP

  selector:
```

```
app: coe-kafka
```

Citrix-observability-exporter ノードポートサービス

```
apiVersion: v1

kind: Service

metadata:

  name: coe-kafka-nodeport

  labels:

    app: coe-kafka

spec:

  type: NodePort

  ports:

    - port: 5557

      protocol: TCP

  selector:

    app: coe-kafka
```

NetScaler Observability Exporterの展開を確認する

NetScaler Observability Exporterの展開を確認するには、以下を実行します。

1. 次のコマンドを使用して展開を確認します。

```
kubectl get deployment,pods,svc -o wide
```

サンプル出力(</en-us/netScaler-observability-exporter/media/coe-kafka-verification-sample-output.png>)

2. ブラウザでURL: `https://<kubernetes-node-IP>:<cpx-ingress-kafka nodeport>` を使用してアプリケーションにアクセスします。

For example, from step 1, access `<http://10.102.61.56:31202/>` in which, `10.102.61.56` is one of the Kubernetes node IPs.

HTTPメソッド (/en-us/netscaler-observability-exporter/media/coe-kafka-http-methods.png)

3. Kafka Consumerを使用してトランザクションデータを表示します。 [PythonKafkaConsumer](#) からKafka Consumerにアクセスします。

次の画像は、Kafka Consumerからのサンプルデータを示しています。

Kafka Consumerからのデータ (/en-us/netscaler-observability-exporter/media/coe-kafka-consumer-transactions.png)

複数のNetScaler Observability ExporterインスタンスをNetScalerに手動で統合する

NetScaler Observability ExporterはNetScalerで手動で構成できます。手動構成は、MPXおよびVPXフォームファクタのNetScalerに適しています。Citrixは、前述のセクションで説明されているように、YAMLファイルを使用してNetScaler Observability Exporterを自動的に展開することを推奨しています。

NetScaler Observability Exporter (coe-kafka.yaml) およびWebアプリケーション (webserver-kafka.yaml) の展開については、前述のセクションを参照してください。

```
enable feature appflow

enable ns mode ULFD

add service COE_svc1 <COE IP1> LOGSTREAM <COE PORT1>

add service COE_svc2 <COE IP2> LOGSTREAM <COE PORT2>

add service COE_svc3 <COE IP3> LOGSTREAM <COE PORT3>

add lb vserver COE LOGSTREAM 0.0.0.0 0

bind lb vserver COE COE_svc1

bind lb vserver COE COE_svc2
```

```
bind lb vsrvr COE COE_svc3

add analytics profile web_profile -collectors COE -type webinsight
-httpURL ENABLED -httpHost ENABLED -httpMethod ENABLED -httpUserAgent
ENABLED -httpContentType ENABLED

add analytics profile tcp_profile -collectors COE -type tcpinsight

bind lb/cs vsrvr <WEB-PROXY> -analyticsProfile web_profile

bind lb/cs vsrvr <WEB-PROXY> -analyticsProfile tcp_profile

# To enable metrics push to prometheus

add service metrhost_SVC <IP> HTTP <PORT>

set analyticsprofile ns_analytics_time_series_profile -collectors
metrhost_SVC -metrics ENABLED -outputMode prometheus
```

FQDNを使用してNetScaler Observability Exporterを追加する

```
enable feature appflow

enable ns mode ULFD

add dns nameserver <KUBE-CoreDNS>

add server COEsvr <FQDN>

add servicegroup COEsvcgrp LOGSTREAM -autoScale DNS

bind servicegroup COEsvcgrp COEsvr <PORT>

add lb vsrvr COE LOGSTREAM 0.0.0.0 0

bind lb vsrvr COE COEsvcgrp

add analytics profile web_profile -collectors COE -type webinsight
-httpURL ENABLED -httpHost ENABLED -httpMethod ENABLED -httpUserAgent
ENABLED -httpContentType ENABLED
```

```
add analytics profile tcp_profile -collectors COE -type tcpinsight

bind lb vserver <WEB-VSERVER> -analyticsProfile web_profile

bind lb vserver <WEB-VSERVER> -analyticsProfile tcp_profile

# To enable metrics push to prometheus

add service metrichost_SVC <IP> HTTP <PORT>

set analyticsprofile ns_analytics_time_series_profile -collectors
metrichost_SVC -metrics ENABLED -outputMode prometheus
```

NetScaler Observability Exporterに関連するトラブルシューティングについては、[NetScaler CPX troubleshooting](#)を参照してください。

NetScaler Observability ExporterからKafkaへのJSON形式でのトランザクションエクスポートのサポート

NetScaler Observability ExporterからKafkaへ、AVRO形式に加えてJSON形式でトランザクションをエクスポートできるようになりました。JSON形式でのトランザクションをサポートするために、KafkaデプロイメントのConfigMapに新しいパラメーター `DataFormat` が導入されました。このパラメーターはAVROとJSONの値を指定できます。JSONベースのトランザクションを許可するには、`DataFormat` の値をJSONとして、[coe-kafka.yaml](#)ファイルで設定します。デフォルト値はAVROです。

次の例は、データ形式がJSONとして構成されたYAMLファイルを示しています。

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: coe-config-kafka

data:

  lstreamd_default.conf: |
```

```
{  
  
  "Endpoints": {  
  
    "KAFKA": {  
  
      "DataFormat": "JSON",  
  
      "ServerUrl": "X.X.X.X:9092,Y.Y.Y.Y:9092", #Specify the  
bootstrap Kafka broker addresses (separated by comma in case multiple  
bootstrap brokers are to be configured)  
  
      "KafkaTopic": "HTTP", #Specify the desired kafka topic  
  
      "RecordType": {  
  
        "HTTP": "all",  
  
        "TCP": "all",  
  
        "SWG": "all",  
  
        "VPN": "all",  
  
        "NGS": "all",  
  
        "ICA": "all",  
  
        "APPPFW": "none",  
  
        "BOT": "none",  
  
        "VIDEOOPT": "none",  
  
        "BURST_CQA": "none",  
  
        "SLA": "none",  
  
        "MONGO": "none"  
  
      },  
  
      "TimeSeries": {
```

```
        "EVENTS": "yes",  
  
        "AUDITLOGS": "yes"  
    }  
  
}  
  
}  
  
}
```

NetScaler Observability Exporter と Splunk Enterprise をエンドポイントとして

NetScaler Observability Exporter は、NetScaler からメトリクスとトランザクションを収集し、さまざまなエンドポイントにデータを送信するコンテナです。NetScaler Observability Exporter は、Splunk Enterprise をエンドポイントとしてサポートしています。

Splunk Enterprise は、機械生成されたビッグデータを検索、監視、分析するためのデータプラットフォームです。Splunk Enterprise は、リポジトリ内のリアルタイムデータをキャプチャ、インデックス化、関連付けし、そこからレポート、グラフ、ダッシュボード、視覚化を生成できます。

分析のために、NetScaler からの監査ログ、イベント、トランザクションを受信するエンドポイントとして Splunk Enterprise を追加できます。Splunk Enterprise は、これらのデータのグラフィカルな表現を提供します。Splunk Enterprise に送信するトランザクション、イベント、監査ログの種類を有効または無効にできます。

NetScaler Observability Exporter を展開する

YAML ファイルを使用して NetScaler Observability Exporter を展開できます。NetScaler の展開に応じて、NetScaler Observability Exporter を Kubernetes クラスターの外部または内部に展開できます。NetScaler Observability Exporter を Kubernetes クラスター内のポッドとして、またはクラスター外の NetScaler MPX または VPX アプライアンスに展開できます。

次の図は、NetScaler を Ingress Gateway として、NetScaler Ingress Controller と NetScaler Observability Exporter をサイドカーとして示しています。NetScaler Observability Exporter は、NetScaler アプリケーションのメトリクスとトランザクションデータを Splunk Enterprise に送信します。Splunk Enterprise は、データのグラフィカルな表現を提供します。



前提条件

- kube-dns または CoreDNS アドオンが有効になっている Kubernetes クラスターがあることを確認してください。

注:

以下の手順では、YAML ファイルを使用して Kubernetes の *default* 名前空間に NetScaler Observability Exporter を展開します。*default* 以外のプライベート名前空間に展開する場合は、YAML ファイルを編集して名前空間を指定してください。

NetScaler Observability Exporter を展開するには、次の手順を実行します。

注:

Web アプリケーションがすでに展開されている場合は、手順 1 と 2 をスキップしてください。

1. 独自の証明書とキーを使用して、シークレット `ingress.crt` とキー `ingress.key` を作成します。

この例では、デフォルトの名前空間で `ing` という名前のシークレットが作成されます。

```
kubectl create secret tls ing --cert=ingress.crt --key=ingress.key
```

2. `webserver-splunk.yaml` から YAML ファイルにアクセスして、アプリケーションをデプロイします。

```
kubectl create -f webserver-splunk.yaml
```

3. アプリケーションの YAML ファイルのイングレスアノテーションで指定することにより、イングレスのスマートアノテーションを使用して、インポートする必要がある特定のパラメーターを定義します。

```
ingress.citrix.com/analyticsprofile: '{"webinsight":  
{"httpurl":"ENABLED", "httpuseragent":"ENABLED",  
"httpHost":"ENABLED", "httpMethod":"ENABLED", "httpContentType":"ENABLED"  
}}'
```

注:

パラメーターは `webserver-splunk.yaml` ファイルで事前定義されています。

アノテーションの詳細については、[Ingress annotations documentation](#)を参照してください。

NetScaler Observability Exporterのサポートを使用して NetScaler CPXをデプロイする

NetScaler Ingress ControllerとともにNetScaler Observability Exporterのサポートを有効にして、NetScaler CPXをサイドカーとしてデプロイできます。NetScaler Observability Exporterのサポートに必要な構成情報を含めるように、NetScaler CPX YAMLファイル `cpx-ingress-splunk.yaml` を変更できます。

以下は、アプリケーションのデプロイ手順の例です。

1. `cpx-ingress-splunk.yaml`と`cic-configmap.yaml`ファイルをダウンロードします。
2. 必要なキーと値のペアを持つConfigMapを作成し、ConfigMapをデプロイします。特定のエンドポイントに対して[directory](#)で利用可能な `cic-configmap.yaml` ファイルを使用できます。
3. 必要に応じて、NetScaler CPX関連のパラメーターを変更します。
4. `cic-configmap.yaml` ファイルを編集し、`NS_ANALYTICS_CONFIG` エンドポイント構成で NetScaler Observability Exporterの以下の変数を指定します。

```
server: 'coe-splunk.default.svc.cluster.local' # COE service FQDN
```

注：

`default` 以外の名前空間を使用している場合は、`coe-splunk.default.svc.cluster.local` to を `coe-splunk.<desired-namespace>.svc.cluster.local` に変更します。NetScaler が Kubernetes クラスターの外部にある場合は、NetScaler Observability Exporter の IP アドレスとノードポートアドレスを指定する必要があります。

5. 以下のコマンドを使用して、NetScaler Observability Exporter をサポートする NetScaler CPXを展開します。

```
kubectl create -f cpx-ingress-splunk.yaml

kubectl create -f cic-configmap.yaml
```

YAML ファイルを使用して NetScaler Observability Exporter を展開する

YAML ファイルを使用して NetScaler Observability Exporter を展開できます。[coe-splunk.yaml](#) から YAML ファイルをダウンロードします。`coe-splunk.yaml` ファイルを編集して、適切な名前空間の Splunk サーバーアドレスを指定してください。

以下は、`coe-splunk.yaml` ファイルの `lstreamd_default.conf` セクションで `ServerUrl` を指定する方法の例です。ここで、`ServerUrl` は Splunk サーバーのアドレスを意味します。

```
lstreamd_default.conf: |  
  
{  
  
  "Endpoints": {  
  
    "SPLUNK": {  
  
      "ServerUrl": "http://10.102.34.155:8088",  
  
      "AuthToken": "",  
  
      "Index": "",  
  
      "RecordType": {  
  
        "HTTP": "all",  
  
        "TCP": "all",  
  
        "SWG": "all",  
  
        "VPN": "all",  
  
        "NGS": "all",  
  
        "ICA@": "all",  
  
        "APPPFW": "none",  
  
        "BOT": "all",  
  
        "VIDEOOPT": "none",
```

```
    "BURST_CQA": "none",

    "SLA": "none",

    "MONGO": "none"

  },

  "TimeSeries": {

    "EVENTS": "yes",

    "AUDITLOGS": "yes"

  },

  "ProcessAlways": "no",

  "ProcessYieldTimeOut": "500",

  "MaxConnections": "512",

  "JsonFileDump": "no"

}

}

}
```

注:

YAML ファイルを使用して NetScaler Observability Exporter を展開する際、Splunk サーバーアドレスに加えて、Splunk Enterprise でデータを送信する **Index** 名を指定できます。デフォルトでは、この **IndexPrefix** オプションは空であり、データは Splunk Enterprise のデフォルトインデックスである **main** にアップロードされます。

Kubernetes YAML を使用して NetScaler Observability Exporter を展開するには、Splunk Enterprise エンドポイントで次のコマンドを実行します。

```
kubectl create -f coe-splunk.yaml
```

注:

カスタム名前空間を使用している場合は、NetScaler Observability Exporter の YAML ファイルを変更してください。

NetScaler Observability Exporter の展開を検証する

NetScaler Observability Exporter、Web アプリケーション、NetScaler CPX、および NetScaler Ingress Controller を展開した後、展開を検証できます。

展開を検証するには、次の手順を実行します。

1. 以下のコマンドを使用して展開を検証します。

```
kubectl get deployment,pods,svc -o wide
```

2. URL を使用してブラウザでアプリケーションにアクセスします。例:

```
https://kubernetes-node-IP:cpx-ingress-splunk nodeport/
```

3. URLを使用してブラウザでSplunkサーバーにアクセスします。例：

```
https://splunk-node-IP:splunk nodeport/
```



Splunk 用の事前構築済みダッシュボードをインポートする

NetScaler が提供する事前構築済みの Splunk ダッシュボードをインポートできます。ダッシュボードをインポートするための JSON ファイルは、[GitHub リポジトリ](#) で入手できます。これらのダッシュボードでは、インスタンス IP アドレス、アプリケーション名、クライアントおよびサーバー IP アドレスなどのパラメータに基づいてトランザクションをフィルタリングするオプションが提供されます。

以下はHTTPダッシュボードのサンプルです。このダッシュボードには、HTTPヘッダーベースのチャート、トランザクション遅延、応答タイプ分布などのデータが表示されます。



以下は、Splunk用のTCPダッシュボードの例です。このダッシュボードには、アプリケーションごとの帯域幅分布、TCPジッター、クライアントおよびサーバーのRTTなどのデータが表示されます。

TCPダッシュボード(/en-us/netscaler-observability-exporter/media/splunk-tcp-dashboard.jpeg)

NetScaler Observability Exporter のトラブルシューティング

このドキュメントでは、NetScaler Observability Exporter の使用中に発生する可能性のある問題のトラブルシューティング方法について説明します。

- NetScaler がアプリケーションデータログを NetScaler Observability Exporter に送信していることを確認するにはどうすればよいですか？

NetScaler がアプリケーションデータログを NetScaler Observability Exporter に送信していることを確認するには、次のコマンドを実行します。

```
nsconmsg -g lstream_tot_trans_written -d current
```

カウンター値は、NetScaler Observability Exporter に送信されたアプリケーショントランザクション（HTTPトランザクションなど）の数を示します。

NetScaler Observability Exporter に送信されるアプリケーショントラフィックレート（HTTP req/sec など）が `lstream_tot_trans_written` と等しくない場合は、次のコマンドを使用して確認できます。

```
nsconmsg -g nslstream_err_ulf_data_not_sendable -d current
```

カウンター値は、ネットワーク輻輳、ネットワーク帯域幅の不足などにより、NetScaler が NetScaler Observability Exporter にデータを送信できず、データが利用可能なバッファに保存されていることを示します。

さまざまなトランザクションデータと個々のフィールド、およびそれらのデータ型に関する情報は、NetScaler の次の場所で入手できます。

```
shell
```

```
/netscaler/appflow/ns_ipfix.yaml
```

NetScaler から NetScaler Observability Exporter にエクスポートされた現在のレコードタイプを確認するには、次のコマンドを使用します。

```
nsconmsg -g appflow_tmpl -d current
```

時系列データ用のメトリックデータエクスポートログの NetScaler への場所：

```
/var/nslog/metrics_prom.log
```

Kafka 関連のカウンターを確認するには、次のコマンドを実行します。

```
kubect1 exec -it <cpx-pod-name> [-c <cpx-container-name>] [-n  
<namespace-name>] -- bash
```

```
tail -f /var/ulflog/counters/lstrmd_counters_codes.log | grep -iE  
"(http_reqs_done|kafka)"
```

NetScaler Observability Exporter の構成が正しく適用されていることを確認するには、次の場所にあるログを見つけてください。

```
vi /var/logproxy/lstreamd/conf/lstreamd.conf
```

NetScaler Observability Exporter が失敗した場合は、次の場所にあるログを収集し、NetScaler サポートに連絡してください。

```
/var/crash/ (Location of the coredump files, if any.)  
  
/var/ulflog/ (Location of `libulfd` logs and counter details.)  
  
/var/log (Location of console logs, lstreamd logs and so on.)
```

NetScaler CPX 関連のトラブルシューティングについては、「[NetScaler CPX のトラブルシューティング](#)」を参照してください。

設定パラメータの説明

このトピックには、`lstreamd_default.conf` ファイルパラメータの説明が含まれています。

`lstreamd_default.conf` パラメータは、エンドポイント固有の設定に使用されます。

- `ServerUrl` : サーバーのアドレスを指定します。

プロトコルは、Apache Kafka の場合は Kafka、Splunk および ElasticSearch の場合は HTTP または HTTPS、Zipkin の場合は HTTP にすることができます。

以下は、異なるエンドポイントのサーバー URL を指定する方法の例です。

カフカ:

```
kafka-server:9992  
  
1.2.3.4:10000
```

スプラック:

```
http://splunk-server:80  
  
https://splunk-server:443  
  
http://1.2.3.4:1000  
  
https://5.6.7.8:2000
```

ElasticSearch:

```
http://elastic-server:80  
  
https://elastic-server:443  
  
http://9.8.7.1:80  
  
https://1.2.3.5:3000
```

ジブキン:

```
http://zipkin-server:80
```

```
http://1.2.3.4:80
```

- **KafkaTopic** :

トランザクションレコードを送信するための Kafka クラスター上のトピックを指定します。デフォルト値は HTTP です。

- **DataFormat** :

Kafka に送信されるデータの形式を指定します。値は JSON または AVRO のいずれかです。デフォルトのデータ形式は AVRO です。

- **MaxConnections** :

エンドポイントへの最大並列 TCP 接続数を指定します。デフォルト値は 64 です。

- **FileSizeMax** :

Kafka エンドポイントの場合、AVRO ファイルはエンドポイントにプッシュされる前にディスクに作成され保存されます。このパラメータは、そのような各ファイルのサイズをキビバイト (KiB) で指定します。1つのファイルには複数のトランザクションレコードを含めることができます。デフォルト値は 48 KiB です。

- **RecordType** :

エクスポートするレコードの種類を指定します。

- HTTP
- TCP
- SWG
- VPN
- ICA®
- APPFW
- BOT
- VIDEOPT
- BURST_CQA
- SLA

- MONGO
- MQTT

NetScaler Observability Exporterは、さまざまなインサイトのトランザクションレコードのフィルタリングを可能にします。デフォルトでは、どのレコードもエクスポートされません。必要なレコードをエクスポートするには、これらのフィールドを適切に設定する必要があります。例:

```
"TCP": "all",  
  
"SWG": "none",  
  
"APPFW": "all"
```

- **EVENTS** :

NetScaler Observability Exporterは、時系列データ（イベントおよび監査ログ）をSplunkおよびKafkaにエクスポートできます。イベントのエクスポートを許可するには、このフィールドを **yes** に設定します。デフォルト値は **no** です。

- **AUDITLOGS** :

監査ログをSplunkおよびKafkaにエクスポートできます。監査ログのエクスポートを許可するには、このフィールドを **yes** に設定します。デフォルト値は **no** です。

- **ConnectionPoolSize** :

Splunkの接続プールサイズを変更します。 **ConnectionPoolSize** と **MaxConnections** は、データが（エンドポイントに）エクスポートされるレートを制御するために使用される場合があります。

- **ElkMaxSendBuffersPerSec** :

ElasticSearchにデータがエクスポートされる最大レート。ELK JSONバッファのサイズは32 KIBです。このフィールドは、毎秒エクスポートされるELKバッファの最大数を設定します。デフォルト値は64です。

- **transRateLimitEnabled** :

時として、受信トラフィックが増加し、Citrix® Observability Exporterがそれに対応できない場合があります。そのような場合、HTTP/HTTPS経由でのSplunk、ElasticSearch、ZipkinなどのJSONエンドポイントへのエクスポートがボトルネックになる可能性があります。メモリはCitrix Observability Exporterが終了するまで制御不能に増加し続けます。このようなシナリオ

を避けるため、Kafkaを含むJSONベースのエンドポイントに対してレート制限を設定できます。Kafkaは効率的なプロトコルであるため、Kafkaへの影響は低いです。JSONベースのエンドポイントのレート制限を有効にするには、このフィールドを **yes** に設定します。デフォルト値は **no** です。

- **transQueueLimit** :

Citrix Observability ExporterがJSONバッファを破棄し始めるまでに蓄積できるJSONバッファの数を指定します。Zipkinの場合、1つのJSONバッファは約64 KiBで、制限が1000の場合、約64 MBのJSONデータを意味します。SplunkおよびElasticSearchの場合、1つのJSONバッファは約32 KiBで、制限が1000の場合、約32 MBのデータを意味します。デフォルト値は1024です。

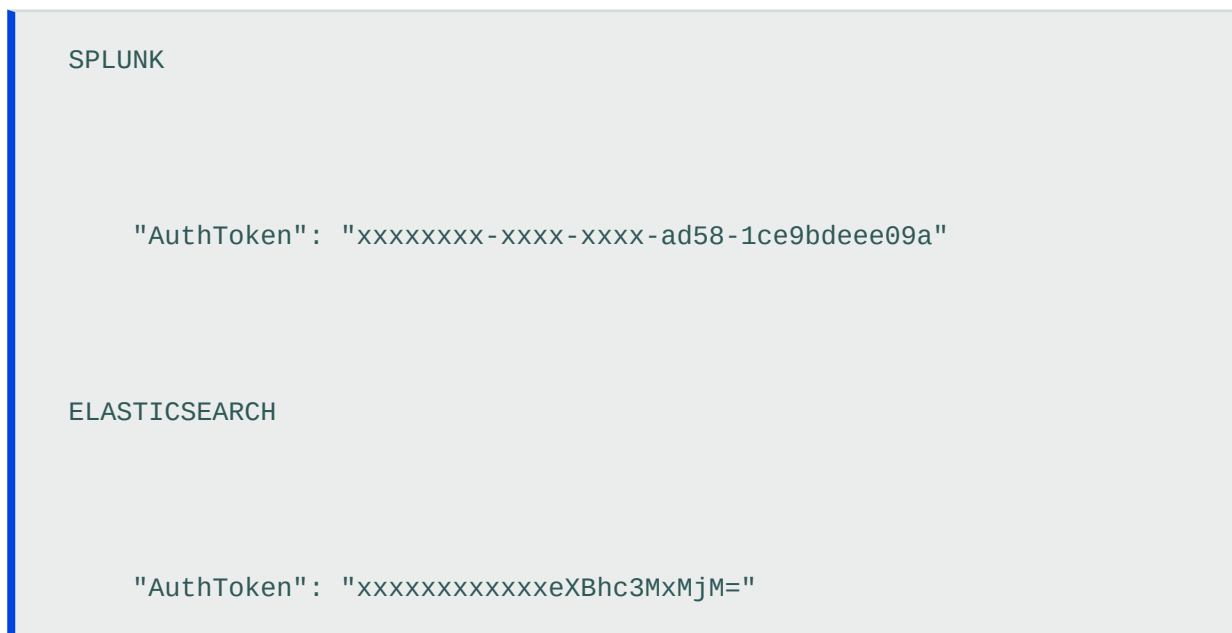
- **transRateLimitWindow** :

再計算ウィンドウを秒単位で指定します。値はゼロより大きい必要があります。ウィンドウサイズが小さいほど、レート制限はより効果的ですが、小さい値を指定するとわずかなCPUオーバーヘッドが発生する可能性があります。デフォルト値は5秒です。

- **AuthToken** :

認証トークンを使用して、Splunkのトークンベース認証を実行できます。ElasticSearchのパスワードベース認証にも使用できます。

例:



- **Index** :

処理されたデータが保存されるSplunkインデックス。デフォルト値は「」です。これはデフォルトのインデックスを意味します。

- **IndexPrefix** :

ElasticSearchで使用するインデックスプレフィックスを指定します。ElasticSearchはAPIを通じて必要に応じてインデックスを作成できます。Kibanaはインデックスパターンを作成でき、それを容易にするためにこのフィールドが使用されます。すべてのインデックス名は、このプレフィックスの後にインデックス作成時刻が続きます。時刻または時刻形式は

IndexInterval に基づいています。

- **IndexInterval** :

インデックスパターンに従ってElasticSearchインデックスがローテーションされる間隔。間隔は次のいずれかの値として構成できます。

- 1時間ごと
- 12時間ごと
- 毎日
- 毎週
- 2週間ごと
- 月次
- 6ヶ月
- 年次

デフォルト値は「」です。これは、インデックスが回転しないことを意味します。

追加情報

lstreamd_default.conf ファイルパラメータを設定する際のガイドラインは以下のとおりです。

- Zipkinは、Splunk、ElasticSearch、またはKafkaと並行してサポートされています。
- Prometheusはプルベースであるため、設定する必要はありません。NetScaler Observability Exporterは、NetScalerメトリクスと自身のメトリクスをPrometheusにエクスポートします。

コンテナのポート5563は、'/metrics'パスで安全でないHTTPを使用してスクレイピングできます。

例:

```
http://coe-fqdn:5563/metrics
```

Prometheusは常に **ON** であり、メトリクスはトランザクション、監査ログ、イベントと並行してエクスポートできます。

- 現在、監査ログやイベントのような時系列データはSplunkとKafkaにのみエクスポートできますが、トランザクションやメトリクスと並行してエクスポートできます。
- 1つのNetScaler Observability Exporterに対して、**lstreamd_default.conf** ファイル内で同じタイプの複数のエンドポイントを設定してはいけません。たとえば、2つのSplunkインスタンス、2つのKafkaインスタンス、2つのElasticSearchインスタンス、または1つのSplunkと1つのElasticSearchなどを設定することはできません。Zipkinの場合、SplunkおよびElasticSearchと並行して設定できますが、複数のZipkinインスタンスを設定することはできません。たとえば、2つのZipkinインスタンスを並行して持つことはできません。
- オプションとしてマークされているフィールドは、一部に事前定義されたデフォルト値がある場合があるため、無視できます。
- **lstreamd_default.conf** に使用されるJSONパーサーは大文字と小文字を区別します。また、余分なコンマや不足しているコンマ、またはJSON形式を無効にする可能性のあるものがないことを確認してください。
- **lstreamd_default.conf** ファイルのパラメータの一部は、このドキュメントには記載されていません。記載されていないパラメータは内部的なものであり、変更を意図していません。それらには、事前に定義されたデフォルト値があります。

コンテナロギングのサポート

NetScaler Observability Exporterで、さまざまな重大度レベルに応じてロギングを有効にできるようになりました。これらのログは、エンドポイント固有の構成に関する情報を取得するのに役立ちます。

以下のロギング重大度レベルがサポートされており、デフォルト値は **INFO** です。

- **NONE** : メッセージはログに記録されません。
- **FATAL** : 致命的なメッセージのみがログに記録されます。
- **ERROR** : 致命的なメッセージとエラーメッセージのみがログに記録されます。
- **INFO** : 致命的、エラー、および情報メッセージのみがログに記録されます。
- **DEBUG** : 致命的、エラー、情報、およびデバッグメッセージのみがログに記録されます。

Kubernetes YAMLベースの展開の場合、デフォルト値はNONEです。ただし、NetScaler Observability ExporterのHelmおよびOpenShiftオペレーター展開の場合、ロギングはデフォルトで有効になっており、INFOに設定されています。

各エンドポイントにNetScaler Observability Exporterを展開する際に、環境変数 **NSOE_LOG_LEVEL** を使用してロギングを構成できます。

次の例は、NetScaler Observability Exporter展開YAMLでログレベルを構成する方法を示しています。

```
env:  
  
  - name: NSOE_LOG_LEVEL  
  
    value: "INFO"
```

ログの説明

取得できるログの種類は次のとおりです。

エラーログ

ログ	重大度	説明
エンドポイントが見つかりません - 無効な構成形式	エラー	lstreamd.conf に「Endpoints」フィールドがありません。
Splunk 認証トークンが見つかりません	エラー	Splunk HEC の AuthToken が見つかりません。
不明なレコードタイプ - 無効な構成	エラー	認識されないレコードです。
レコードタイプオプション - 無効な構成	エラー	認識されない引数です。all/none/anomalous のみが受け入れられます。
不明な時系列 - 無効な構成	エラー	AUDITLOGS/EVENTSのみが受け入れられます。
レート制限を無効に設定 - 無効なレート制限オプション - 無効な構成	エラー	レート制限の構成が無効です。
異常な構成が見つかりません	エラー	異常な構成が見つかりません。
構成ファイル <lstreamd.conf> が存在しません	エラー	OE は lstreamd.conf の読み取りを試みましたが、見つかりませんでした。
エンドポイント名またはサーバーアドレスが無効です（空です）	エラー	SPLUNK、KAFKA、ES、ZIPKIN などのエンドポイントの種類が空であるか、そのアドレス (ServerUrl) が空です。
無効な値 - <endpoint-type> の Timeseries/ <timeseries-type> エクスポートに <val> が「いいえ」に設定されています	エラー	指定された時系列タイプに無効な値です（"yes"または"no"のみ許可されます） - デフォルトは"no"です。

ログ	重大度	説明
クライアントのパケットエンジンコンテキストの割り当てに失敗しました	エラー	ULFDコンシューマースレッドがパケットエンジンコンテキスト（NetScaler/パケットエンジンごとのコンテキスト）の割り当てに失敗しました。
ログストリームバッファを保持するためのメモリの割り当てに失敗しました - メモリ不足かもしれません	エラー	ULFDコンシューマースレッドがOOM（メモリ不足）になり、受信したログストリームバッファを保持するために必要な約8 KiBのメモリを割り当てられませんでした。
クライアント <code><client-id></code> のプロセッサの初期化に失敗しました	エラー	クライアントのプロセッサの初期化に失敗しました。

情報ログ

ログ	重大度	説明
Istreamd プロセス開始中	情報	NSOE が起動しました。
Istreamd.conf が見つかりません - 新規作成し、エージェント/adm オンプレミスモードに切り替えています	情報	Istreamd.conf が見つかりませんでした。
ELK ヘルスステータス ダウン	情報	ElasticSearch は停止しています。
ELK ヘルスステータス アップ	情報	ElasticSearch は稼働しています。
Splunk ヘルスステータス ダウン	情報	Splunk は停止しています。
Splunk ヘルスステータス UP	情報	Splunk が稼働中です。

ログ	重大度	説明
NetScaler IP: <code><nsip></code> Core: <code><core-id></code> Partition: <code><partition-id></code> を削除しています。	情報	NetScaler から切断されました。
新しい NetScaler IP: <code><nsip></code> Core: <code><core-id></code> Partition: <code><partition-id></code> を追加しています。	情報	新しい NetScaler に接続しました。
新しい NetScaler の割り当て	情報	新しい NetScaler に接続しました。
新しい NetScaler の初期化	情報	新しい NetScaler に接続しました。
KAFKA用にJSONデータ形式が設定されました	情報	JSONはKafkaにエクスポートされます。
Kafka用にAVROデータ形式が設定されました	情報	AVROはKafkaにエクスポートされます。
Zipkinのトレースを有効にする	情報	Zipkinが構成されました。
エンドポイント <code><type></code> への最大接続数を <code><maxSockets></code> に設定しました	情報	<code>MaxConnections</code> は、解析されたか、または表示されたエンドポイントの表示値にデフォルト設定されました。
Kafka Exporterを有効にする	情報	KafkaはIstreamd.confで構成され、有効になりました。
設定されたKafkaブローカーリスト: <code><brokers></code>	情報	設定されたKafkaブローカーリスト (serverUrlから) が適用されました。
設定されたKafkaトピック: <code><topic></code>	情報	KafkaTopicは解析されたか、または表示された値にデフォルト設定されました。

ログ	重大度	説明
<endpoint-type> が <all / anomalous / none> <rec-type> レコード をエクスポートするように設 定されました	情報	表示された <rec-type: ex HTTP_A/TCP_A/...> タイプ のレコードのみが、表示され たエンドポイントに対してエ クスポートされるように設定 されました。
時系列/ <timeseries-type> を <endpoint-type> に対し て<はいいいえ>に設定しま した	情報	監査ログ/イベント/メトリク スは、指定されたエンドポイ ントに対して有効/無効にされ ました。
プロセッサのイールドタイム アウトを <val> に設定しま した	情報	プロセッサスレッドのイール ドタイムは設定されたか、ま たはデフォルト設定されまし た。
Prometheusモードが有効に なりました - Prometheusは RESTポートでメトリクスを スクレイピングできるよう になりました	情報	PrometheusはRESTポートで 「GET /metrics」を実行する 権限を与えられています。
新しいクライアント接続を受 信しました	情報	NetScaler (パケットエンジ ン) からの接続を受信しまし た。
クライアントからの切断を受 信しました	情報	NetScaler (パケットエンジ ン) からの切断を受信しまし た。
クライアントからのリセット を受信しました	情報	NetScaler (パケットエンジ ン) からの切断 (リセット) を 受信しました。
JSONトランザクションの レート制限が有効になりまし た	情報	Splunk、ElasticSearch、 Zipkin、Kafka (JSON) など、 JSONベースのエンドポイン トへのトランザクションの レート制限を有効にしまし た。

ログ	重大度	説明
JSONトランザクションのレート制限を1秒あたり <num> のログストリームバッファに設定	情報	ログストリームバッファの1秒あたりのレート制限。
JSONトランザクションのエクスポートキュー制限を <num> のJSONバッファに設定	情報	JSONエクスポートキューの制限 - このしきい値を超えると、エクスポートはJSONバッファのドロップを開始します。
JSONトランザクションのレート制限ウィンドウを <num> 秒に設定	情報	JSONベースのエンドポイントへのトランザクションのレート制限ウィンドウ。値が低いほどスパイクを捕捉できます。

警告ログ

ログ	重大度	説明
Prometheusモードが無効 - PrometheusはRESTポートでメトリクスをスクレイピングする権限がありません	警告	Prometheusは、restポートで「GET /metrics」を実行する権限がありません。

デバッグログ

ログ	重大度	説明
プロセッサ #1 をスレッドインデックス: 0、スレッドID: <id> で生成しました。	デバッグ	最初の処理スレッドを作成し、開始しました。
ULFDレシーバーをスレッドインデックス: 1、スレッドID: <id> で生成しました。	デバッグ	ULFDコンシューマースレッドを作成し、開始しました。

ログ	重大度	説明
エクスポートスレッドインデックス: 2、スレッドID: <code><id></code> で生成しました。	デバッグ	エクスポートスレッドを作成し、開始しました。
プロセッサ #2 をスレッドインデックス: 3、スレッドID: <code><id></code> で生成しました。	デバッグ	2番目の処理スレッドを作成し、開始しました。
ULFDレシーバーは信号番号- <code><signo></code> を受信しました	デバッグ	ULFDコンシューマースレッドが信号を受信しました (通常はNSULFDから)。通常、SIGUSR2はカウンタファイルのローテーションを誘発するために使用されます。
ULFDレシーバーがタイムアウトイベントを受信しました	デバッグ	ULFDコンシューマースレッドがタイムアウトイベントを受信しました (NSULFDから毎秒受信)。これは、CLMメタレコードを毎分クリーンアップするために使用されます。
ULFDレシーバーが処理できないイベントを受信しました - 無視します	デバッグ	ULFDコンシューマースレッドが不明なイベントを受信したため、無視しました。
ULFDレシーバーがログストリームバッファを受信しました	デバッグ	ULFDコンシューマースレッドがデータバッファ (ログストリーム) を受信しました。
受信したログストリームバッファは破損しています - 解析できません	デバッグ	ULFDコンシューマースレッドが破損したデータバッファ (ログストリーム) を受信しました。
受信したログストリームバッファを解析しました: client-id=<>, core-id=<>, partition-id=<>, namespace=<>	デバッグ	ULFDコンシューマースレッドが受信したデータバッファを解析しました。

ログ	重大度	説明
レートリミッターはログストリームバッファを破棄することを決定しました	デバッグ	JSONトランザクションのレート制限によりバッファが破棄されました。
レートリミッターはログストリームバッファを受け入れることを決定しました	デバッグ	JSONトランザクションのレートリミッターによってバッファが受け入れられました。
このクライアントのプロセッサがキューを十分に速く空にしていなかったため、ログストリームバッファが破棄されました	デバッグ	このNetScaler（パケットエンジン）に関連付けられた処理スレッドがバッファを十分に速く消費していなかったため、バッファが破棄されました。
設定が非異常バッファの受け入れを許可していないため、ログストリームバッファが破棄されました	デバッグ	<code>Lstreamd</code> が異常なトランザクションのみを処理するように構成されているため、バッファードロップが発生しました。
Logstreamバッファの処理が完了したため、解放しています	デバッグ	ULFDコンシューマースレッドは受信したLogstreamバッファの処理を完了しました。そのため、共有メモリを解放します。
ULFDレシーバーがLogstreamバッファを破棄しました。バッファが破損していたか、メモリ不足が発生したためです。	デバッグ	ULFDコンシューマースレッドはLogstreamバッファを破棄しました。バッファが破損していたか、OOM（メモリ不足）が発生したためです。
クライアント <code><client-id></code> 用に <code><AVRO/JSON></code> プロセッサを初期化しました	デバッグ	クライアント用に表示されたプロセッサを初期化しました (NetScaler / パケットエンジンが初めて接続したときに実行されます)。

ログ	重大度	説明
<num-pushed> 個の Logstreamバッファをプロセッサへプッシュしました	デバッグ	表示された数のLogstreamバッファを、関連する処理スレッドのキューへプッシュしました。
thread-id=<thread-id> , client-id=<client-id> を使用してJSONプロセッサのタイマーが起動しました	デバッグ	JSON処理スレッドの処理タイマーがタイムアウトしました。
スレッドID= <thread-id> のJSONプロセッサ、 client-id=<client-id> がCPUを解放しました	デバッグ	JSON処理スレッドがCPUを解放しました。
JSON処理が開始されました	デバッグ	キューに蓄積されたログストリームバッファのJSONへの変換を開始しました。
ログストリームバッファの処理が開始されました	デバッグ	ログストリームバッファの処理を開始しました。
JSONエンドポイントが正常ではないため、トランザクションを破棄しています (ヘルスチェックをスキップするには「ProcessAlways」を「yes」に設定してください)。	デバッグ	トランザクションは、JSONエンドポイント (Splunk/ElasticSearch) が正常でなかったか、またはKafka (JSON) が有効になっていなかったため、破棄されました。
トランザクションが破棄されました: 未設定のレコードタイプ - <rec-type> のJSONオブジェクトキューを作成できません	デバッグ	トランザクションは、設定されていない（または「なし」に設定されている）レコードタイプに属していたため、破棄されました。
レコードタイプ <rec-type> のJSONオブジェクトキューを作成しています	デバッグ	設定されたレコードタイプのJSONオブジェクトキューを作成しました (JSONバッファをエクスポーターにプッ

ログ	重大度	説明
		シュするまで保持するために使用されます)。
トランザクションが破棄されました: 不明なプロトコルID <code><protocol-id></code> のJSONオブジェクトキューを作成できません	デバッグ	トランザクションは、不明なレコードタイプ/プロトコルIDに属していたため、破棄されました。
JSONオブジェクトキューを作成しました (protocol-id= <code><protocol-id></code>)	デバッグ	表示されたレコードタイプ/プロトコルIDのJSONオブジェクトキューを作成しました。
JSONオブジェクトキューのインデックスを <code><index-string></code> に設定しました	デバッグ	JSONオブジェクトキューのインデックスを設定しました (インデックスを使用するElasticSearch/Splunkエンドポイントに有効です)。
トランザクションが破棄されました: JSONオブジェクトキューを作成できません - メモリ不足の可能性があります	デバッグ	メモリ不足のためJSONオブジェクトキューを作成できなかったため、トランザクションは破棄されました。
トランザクション処理が開始されました	デバッグ	トランザクションのJSONへの変換を開始しました。
異常: フィールドはインデックス化されていますが、テーブルに見つかりませんでした - コード= <code><id></code>	デバッグ	フィールドはインデックス化されていますが、ローカルメタテーブルに見つかりませんでした。
トランザクションのレコードタイプ <code><rec-type></code> を生成しました	デバッグ	トランザクションの印刷されたタイプのJSONレコードを生成しました
トランザクション処理が完了しました	デバッグ	トランザクションのJSONへの変換を完了しました。

ログ	重大度	説明
このクライアントから AppFlowコードネームを持つメタレコードがまだ到着していないため、ログストリームバッファの処理に失敗しました	デバッグ	NetScaler (パケットエンジン) からコードマップが届いていないため、トランザクションを破棄しました。
ログストリームバッファの処理が完了しました	デバッグ	ログストリームバッファの処理が完了しました。
JSON処理が完了しました	デバッグ	キューに積み重ねられたログストリームバッファのJSONへの変換が完了しました
変換されたJSONトランザクションを処理済みキューにプッシュしました	デバッグ	変換されたJSONトランザクションを処理済みキュー（すべてのレコードタイプの処理済みJSONがエクスポーターに引き渡される前に待機する共通キュー）にプッシュしました。
処理済みキューに積み重ねられたすべてのJSONをエクスポーターにプッシュしました	デバッグ	処理済みキューで待機しているすべてのJSONトランザクションをエクスポーターに引き渡しました。
積み重ねられたJSONバッファのKafkaクライアントへのエクスポートを開始しました	デバッグ	エクスポートキューに積み上げられたJSONバッファのKafkaクライアントへのエクスポートを開始しました。
積み上げられたJSONバッファのKafkaクライアントへのエクスポートが完了しました	デバッグ	エクスポートキューに積み上げられたJSONバッファのKafkaクライアントへのエクスポートが完了しました。
Kafkaクライアントへのエクスポートに失敗したため、JSONバッファをエクスポートキューに戻しています	デバッグ	JSONバッファをKafkaクライアントにエクスポートできませんでした。そのため、エクスポートキューに戻します。

ログ	重大度	説明
エクスポートキューが設定された制限に達したため、エクスポーターは <numDropped> 個のJSONバッファを破棄しました	デバッグ	レート制限のため、エクスポートキューから破棄します。
Kafkaトピックマネージャーがまだ作成されていないため、JSONバッファをKafkaクライアントにエクスポートできませんでした	デバッグ	Kafkaトピックマネージャーがまだ作成されていないため、JSONバッファをKafkaクライアントにエクスポートできませんでした。
Kafkaトピックマネージャーを作成しました	デバッグ	Kafka Topic Manager (Kafkaクライアントの状態を維持するために使用) を構築しました。
Kafka Topic Managerがトピックを作成しています - <topic>	デバッグ	Kafka Topic Managerが設定されたトピックの作成を試みています。
Kafka Topic Managerはトピックの作成に失敗しました - メモリ不足の可能性があります	デバッグ	Kafka Topic ManagerはOOMが発生したため、設定されたトピックの作成に失敗しました。
Kafka Topic Managerがトピックを作成しました	デバッグ	Kafka Topic Managerが設定されたトピックを作成しました
設定されたトピックを検証できません - Kafkaクライアントへのメタ接続の取得に失敗しました。メモリ不足の可能性があります	デバッグ	Kafka Topic Managerはトピックの検証に失敗しました - OOMが発生した可能性があります。
トピック検証のためにKafkaクライアントにメタ要求を送信しました	デバッグ	Kafka Topic Manager は、トピック検証のために Kafka クライアントにメタ要求を正常に発行しました。

ログ	重大度	説明
Kafka クライアントへのメタ接続の作成に失敗しました。 メモリ不足が原因かもしれません。	デバッグ	Kafka クライアントへのメタ接続の作成に失敗しました。 OOM が発生した可能性があります。
Kafka クライアントへのメタ接続を作成しました - connection-id= <id>	デバッグ	設定されたトピックのために Kafka クライアントへのメタ接続を作成しました。
Kafka クライアントへの既存のメタ接続を再利用しました - connection-id= <id>	デバッグ	Kafka クライアントへの既存のメタ接続を再利用しました。
Kafka クライアントへの古いメタ接続をクリーンアップしました - connection-id= <id>	デバッグ	設定されたトピックのために、Meta Connection Pool から Kafka クライアントへの誤ったメタ接続を削除しました。
Kafka クライアントへの古い Produce 接続をクリーンアップしています - connection-id= <id>	デバッグ	Kafka クライアントへの誤ったプロデュース接続をプロデュース接続プールから削除しました。
Kafka クライアントへのプロデュース接続をクリーンアップしました - connection-id= <id>	デバッグ	Kafka クライアントへの有効なプロデュース接続をプロデュース接続プールから削除しました。
Kafka クライアントへの既存のプロデュース接続を再利用しました - connection-id= <id>	デバッグ	Kafka クライアントへの既存のプロデュース接続を再利用しました。
partition= <id> , topic= <topic> のプロデュース接続をクリーンアップしています。	デバッグ	Kafka クライアントへのプロデュース接続のクリーンアップを開始しました。

ログ	重大度	説明
アクティブなプロデューズ接続を強制的にクリーンアップしました。	デバッグ	パーティションクリーンアップ中に、Kafkaクライアントへのアクティブなプロデューズ接続を強制的にクリーンアップしました。
非アクティブなプロデューズ接続をクリーンアップしました。	デバッグ	パーティションクリーンアップ中に、非アクティブなプロデューズ接続をKafkaクライアントにクリーンアップしました。
Kafkaプロデューズ接続をStale Connection Poolにプッシュできませんでした - Produce Connection Poolに見つかりません！ - connection-id= <id>	デバッグ	Kafka Produce Connection PoolでKafkaプロデューズ接続を見つけることができませんでした。
Kafkaプロデューズ接続をStale Connection Poolにプッシュしました - connection-id= <id>	デバッグ	Kafkaプロデューズ接続をKafka Erroneous Connection Queueにプッシュしました。
Kafkaプロデューズ接続をFree Connection Poolにプッシュできませんでした - Produce Connection Poolに見つかりません！ - connection-id= <id>	デバッグ	Kafka Produce Connection PoolでKafkaプロデューズ接続を見つけることができませんでした。
Kafkaプロデューズ接続をFree Connection Poolにプッシュしました - connection-id= <id>	デバッグ	Kafkaプロデューズ接続をKafka Erroneous Connection Queueにプッシュしました。
パーティションのプロデューズ接続をクリーンアップしました。	デバッグ	パーティションの生成接続をクリーンアップしました。

ログ	重大度	説明
トピック <topic> の無効なパーティションのクリーンアップを開始します	デバッグ	無効なパーティションのクリーンアップを開始しました。
無効なパーティションをクリーンアップしました - id= <id>	デバッグ	無効なパーティションを無効パーティションキューから削除しました。
無効なパーティションのクリーンアップが完了しました	デバッグ	トピックのパーティションのクリーンアップが完了しました。
失敗したJSONファイルをKafkaに再試行できませんでした - トピックの状態が無効です	デバッグ	トピックの状態が無効なため、失敗したJSONファイルをKafkaに再試行できませんでした。
異常: Kafka JSONファイルの再試行に失敗しました	デバッグ	何らかの異常により、失敗したJSONファイルをKafkaに再試行できませんでした。
Kafka JSONファイルを再試行しました	デバッグ	再試行のためにKafka JSONファイルをKafkaクライアントにプッシュしました。
トピックのクリーンアップを開始します	デバッグ	トピックのクリーンアップを開始しました（7秒ごとに実行）。
トピック <topic> をクリーンアップしました。	デバッグ	印刷されたトピックをクリーンアップしました。
トピックのクリーンアップを完了しました	デバッグ	トピックのクリーンアップを完了しました。
KafkaトピックマネージャーによってKafkaトピックがまだ作成されていないため、JSONバッファをKafkaクライアントにエクスポートできませんでした	デバッグ	Kafka Topic Managerがまだトピックを作成していないため、JSON BufferをKafka Clientにエクスポートできませんでした。

ログ	重大度	説明
トピックが有効な状態であるため、Kafka JSONファイルはパーティション- <code><pid></code> にプッシュされました。	デバッグ	Kafka JSONファイルはKafka Clientにエクスポートされました。
トピックが無効な状態であるため、Kafka JSONファイルをKafka Clientにプッシュできませんでした。	デバッグ	トピックが無効な状態であるため、JSONファイルをKafka Clientにエクスポートできませんでした。
トピックが有効な状態であるため、Kafka JSON Bufferはパーティション- <code><pid></code> にプッシュされました。	デバッグ	Kafka JSON BufferはKafka Clientにエクスポートされました。
トピックが無効な状態であるため、Kafka JSON BufferをKafka Clientにプッシュできませんでした。	デバッグ	トピックが無効な状態であるため、JSON BufferをKafka Clientにエクスポートできませんでした。
Kafka ClientへのProduce Requestの送信に失敗したため、Kafka JSON Bufferが停止しました。メモリ不足の可能性があります。	デバッグ	メモリ不足のため、Kafka JSONバッファが完全に失われました。
Produce接続の不足によりKafka JSONバッファが停止しました - おそらくメモリ不足です	デバッグ	メモリ不足のため、Kafka JSONバッファが完全に失われました。
Kafka JSONバッファをKafka クライアントにプッシュしました	デバッグ	Kafka JSONバッファをKafka クライアントにエクスポートしました。
KafkaクライアントへのProduceリクエストの送信に失敗したため、Kafka JSONファイルが停止しました - おそらくメモリ不足です	デバッグ	メモリ不足のため、Kafka JSONファイルが完全に失われました。

ログ	重大度	説明
Produce接続の不足により Kafka JSONファイルが停止 しました - おそらくメモリ不 足です	デバッグ	メモリ不足のため、Kafka JSONファイルが完全に失わ れました。
Kafka JSONファイルをKafka クライアントにプッシュしま した	デバッグ	Kafka JSONファイルをKafka クライアントにエクスポート しました。
Kafkaクライアントは、プッ シュされたJSONバッファが 空だったため拒否しました	デバッグ	Kafkaクライアントは、プッ シュされたJSONバッファが 空だったため拒否しました。
Kafkaクライアントは、プッ シュされたJSONバッファの 生成リクエストを作成できま せんでした。おそらくメモリ 不足です	デバッグ	Kafkaクライアントは、生成 リクエストの作成中にOOM (メモリ不足) になりまし た。
Kafkaクライアントは、プッ シュされたJSONバッファの Kafkaファイルを作成できま せんでした。おそらくメモリ 不足です	デバッグ	Kafkaクライアントは、Kafka ファイルの作成中にOOM (メモリ不足) になりまし た。
KafkaクライアントはJSON バッファをKafkaにディス パッチしました	デバッグ	KafkaクライアントはJSON バッファをKafkaにディス パッチしました。
Kafkaクライアントは、プッ シュされたJSONファイルの 生成リクエストを作成できま せんでした。おそらくメモリ 不足です	デバッグ	プロデュースリクエストの作 成中にKafkaクライアントで OOMが発生しました。
KafkaクライアントがJSON ファイルをKafkaにディス パッチしました	デバッグ	KafkaクライアントがJSON ファイルをKafkaにディス パッチしました。

ログ	重大度	説明
Kafka JSONファイルの消失 - コード= <code><errno></code>	デバッグ	表示されたエラーコードのため、Kafkaファイルが永久に失われました。
異常: このKafka接続が属するトピックが存在しません！	デバッグ	異常: このKafkaプロデュース接続は存在しないトピックに属していました。
異常: このKafka接続が属するパーティションが存在しません！	デバッグ	異常: このKafkaプロデュース接続は存在しないトピックに属していました。
Kafka JSONファイルの消失 - コード= <code><errno></code>	デバッグ	表示されたエラーコードが原因で、Kafkaファイルが永久に失われました。
異常: ファイルは非アクティブなProduce接続で送信されました	デバッグ	異常: ファイルは非アクティブなproduce接続で送信されました。
Kafka Produceリクエスト (JSON) が失敗しました - コード= <code><errno></code>	デバッグ	Kafka (JSON) ファイルのエクスポートに失敗しました。
異常: Kafkaトピックが停止しました - JSONファイルの停止	デバッグ	異常: Kafkaトピックが停止しました - JSONファイルの停止。
Kafka Produceリクエストが成功しました - JSONファイルがアップロードされました	デバッグ	JSONファイルをKafkaにアップロードしました。
再試行中にKafka JSONファイルが停止しました - 設定された再試行レートは0です！	デバッグ	KafkaRetryRate が 0 のため、新しいJSON ファイルを破棄しました。
再試行中に古い Kafka JSONファイルが失われました - 再試行キューの制限に達しました	デバッグ	再試行キューが満杯（最大 32K ファイルが保存されています）のため、古いJSON ファイルを破棄しました。

ログ	重大度	説明
新しい Kafka JSON ファイルを再試行キューにプッシュしました	デバッグ	新しい Kafka JSON ファイルを再試行キューにプッシュしました。
Kafka クライアントがトピックメタデータリクエストの作成に失敗しました - メモリ不足の可能性があります	デバッグ	Kafka クライアントが OOM になりました。
Kafka クライアントがトピックメタデータリクエストを Kafka にディスパッチしました	デバッグ	Kafka クライアントがトピックメタデータリクエストを Kafka にディスパッチしました。
Kafka クライアントへのプロデュース接続の作成に失敗しました - メモリ不足の可能性があります	デバッグ	Kafka クライアントが OOM を実行しました。
Kafka クライアントへのプロデュース接続を作成しました - connection-id= <id>	デバッグ	Kafka クライアントはプロデュース接続を正常に作成しました。
異常: この Kafka メタ接続が属するトピックが存在しません!	デバッグ	異常: この Kafka メタ接続は存在しないトピックに属していました。
Kafka から無効なトピックメタデータ応答を受信しました - code= <errno>, empty=<true/false>, status=<sent/not sent>	デバッグ	トピックの状態が無効化されました。
Kafka から受信したトピックメタデータ応答にはトピックが含まれていませんでした	デバッグ	トピックの状態が無効化されました。
Kafka クライアントはトピック- <topic> のパーティション- <id> の作成に失敗	デバッグ	Kafka クライアントが OOM になりました

ログ	重大度	説明
しました。おそらくメモリ不足です。		
Kafkaクライアントが、リーダー= <code><leader></code> 、パーティションID= <code><id></code> 、トピック= <code><topic></code> でパーティションを作成しました。トピックの状態が検証されました。	デバッグ	Kafkaクライアントがパーティションを作成しました。
Kafkaクライアントがトピック <code><topic></code> の有効なパーティションを見つけられず、トピックの状態を無効にしました。	デバッグ	Kafkaクライアントがトピックに適したパーティションを見つけられず、その状態を無効にしました。
まれな状態: Kafkaクライアントへのアクティブなプロデュース接続を持つパーティションを再作成しています。古いパーティションを無効にし、新しいパーティションを作成しています。パーティションID= <code><id></code> 、トピック= <code><topic></code>	デバッグ	Kafka上のそのパーティションにプロデュース接続がアクティブにファイルをプッシュしている間に、パーティションのリーダーが変更されました。そのため、そのパーティションを無効にし、代わりに新しいパーティションを作成しました。
Kafkaクライアントが、リーダー= <code><leader></code> 、パーティションID= <code><id></code> 、トピック= <code><topic></code> でパーティションを再作成しました。トピックの状態が検証されました。	デバッグ	Kafkaクライアントがパーティションを再作成しました。
Kafkaクライアントがパーティション <code><id></code> をリーダーなしに設定しています。トピック= <code><topic></code>	デバッグ	出力されたトピックパーティションのリーダーが見つかりませんでした。
Kafkaクライアントがパーティション <code><id></code> をリー	デバッグ	出力されたトピックパーティションのリーダーが見つかりました。

ログ	重大度	説明
ダーとして設定しています。 トピック= <code><topic></code>		
トピック <code><topic></code> のどのパーティションにもリーダーが存在しないため、その状態を無効にします。	デバッグ	設定されたトピックのリーダーが見つからなかったため、その状態を無効にします。
KafkaクライアントがKafkaから有効なトピックメタデータ応答を受信しました。トピック <code><topic></code> は検証済みです。	デバッグ	トピックの状態が検証されました。
監査ログに対してPOST以外のリクエストを受信しました。POSTのみがサポートされているため、破棄します。	デバッグ	監査ログに対してPOST以外のリクエストを受信しました。このURLではPOSTのみがサポートされているため、破棄します。
監査ログのPOSTリクエストにx-appflow-idヘッダーが存在しないため、破棄します。	デバッグ	監査ログのPOSTリクエストにx-appflow-idヘッダーが存在しません -- 破棄します。
リクエスターは監査ログをPOSTする権限がありません -- 破棄します	デバッグ	リクエスターは監査ログをPOSTする権限がありません -- 破棄します。
監査ログリクエストを受信しました	デバッグ	監査ログリクエストを受信しました。
イベントに対する非POSTリクエストを受信しました -- POSTのみがサポートされています、破棄します	デバッグ	イベントに対する非POSTリクエストを受信しました -- このURLではPOSTのみがサポートされています、破棄します。
イベントのPOSTリクエストにx-appflow-idヘッダーが存在しません -- 破棄します	デバッグ	イベントのPOSTリクエストにx-appflow-idヘッダーが存在しません -- 破棄します。

ログ	重大度	説明
リクエスターはイベントをPOSTする権限がありません -- 破棄します	デバッグ	リクエスターはイベントをPOSTする権限がありません -- 破棄します。
イベントリクエストを受信しました	デバッグ	イベントリクエストを受信しました。
空の監査ログ/イベントバッファの終了	デバッグ	空のバッファはエクスポートされません。
メモリ不足によるイベント/監査ログバッファの終了	デバッグ	OOMが発生しました。
イベント/監査ログを処理しました	デバッグ	イベント/監査ログを処理しました。
過大なイベント/監査ログの終了	デバッグ	1つの監査ログ/イベントが大きすぎました (32 KiBを超過)。
JSON解析エラーによるイベント/監査ログバッファの破損	デバッグ	JSON解析エラーによるイベント/監査ログバッファの破損。
処理中の <num> イベント/監査ログの破損	デバッグ	処理中に一部の監査ログ/イベントが破損しました。
<num> イベント/監査ログJSONバッファをエクスポートキューにプッシュしました	デバッグ	一部のイベント/監査ログをエクスポートキューにプッシュしました。

fix

This is the content for the sample heading topic. It is hidden from the main table of contents but remains accessible via links.