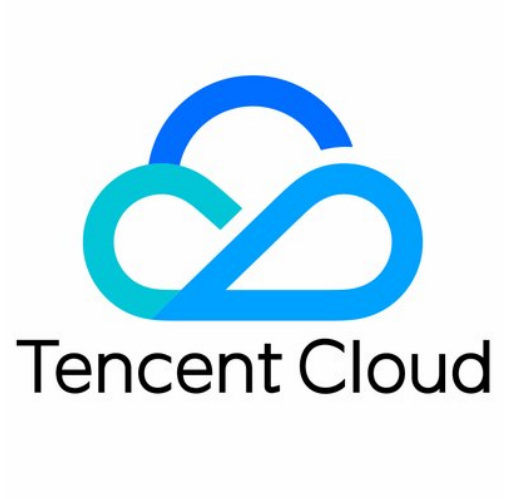
Tencent Kubernetes Engine

TKE Serverlessクラスターガイド

製品ドキュメント



カタログ：

TKE Serverlessクラスターガイド
TKE Serverlessクラスターの購入
リージョンとアベイラビリティーゾーン

最終更新日：：2023-04-27 16:40:34

リージョン

概要

リージョン（Region）とは、データセンターの物理的な場所です。Tencent Cloudでは、リージョンとリージョンの間は完全に分離されており、異なるリージョン間で最大限の安定性とフォールトトレランスが確保されます。アクセスの遅延を低減し、ダウンロード速度を向上させるためには、顧客に最も近いリージョンを選定することをお勧めします。
下表によってTKE Serverlessクラスターが現在サポートしているリージョン、アベイラビリティーゾーンおよびリソースタイプを確認できます。その他のリージョンは順次開放されます。

特徴

異なるリージョン間のネットワークは完全に隔離され、異なるリージョン間のクラウドサービスは、**デフォルトではプライベートネットワークを介して通信できません。**。
異なるリージョン間のクラウドサービスは、パブリックIP を介してInternetにアクセスする方式で通信を行うことができます。また異なるプライベートネットワークに属するクラウドサービスは、 CCNを介して通信を行うことができ、この通信方式の方がより高速で、安定しています。
CLBは、現在、デフォルトで同一リージョンのトラフィック転送をサポートし、ローカルドメインのCVMをバインドします。 クロスリージョンバインディング機能を有効にすることによって、CLBをサポートし、リージョン間でCVMをバインドすることができます。

アベイラビリティーゾーン

概要

アベイラビリティーゾーンとは、Tencent Cloudが同一リージョン内で電源とネットワークが互いに独立している物理的なデータセンターです。ビジネスの持続性を確保し、アベイラビリティーゾーン間の障害を互いに隔離させ（大規模な災害または大規模な電源設備の障害を除く）、障害が広がらないようにすることを目的としています。独立したアベイラビリティーゾーンでインスタンスを起動することにより、ユーザーは単一障害点からアプリケーションを保護できます。

APIインターフェースアベイラビリティーゾーン一覧照会 を介して、完全なアベイラビリティーゾーン一覧を確認できます。

特徴

同一リージョンで異なるアベイラビリティーゾーンに位置していても、同一のプライベートネットワーク下のクラウド製品はプライベートネットワークを介して相互接続されており、プライベートIPを使用して直接アクセスすることができます。

説明

プライベートネットワークの相互接続は同じアカウント下のリソースの相互接続を指し、異なるアカウントのリソースのプライベートネットワークは完全に隔離されます。

TKE Serverlessクラスターがサポートするリージョン

中国

リージョン	アベイラビリティーゾーン
華南地域（広州） ap-guangzhou	広州 3 ap-guangzhou-3
	広州 4 ap-guangzhou-4
	広州 6 ap-guangzhou-6
	広州 7 ap-guangzhou-7
華東地域（上海） ap-shanghai	上海 2 ap-shanghai-2
	上海 3 ap-shanghai-3
	上海 4 ap-shanghai-4
	上海 5 ap-shanghai-5
華東地域（南京）	南京 1

ap-nanjing	ap-nanjing-1
	南京 2 ap-nanjing-2
華北地域（北京） ap-beijing	北京 3 ap-beijing-3
	北京 4 ap-beijing-4
	北京 5 ap-beijing-5
	北京 6 ap-beijing-6
	北京 7 ap-beijing-7
西南地域（成都） ap-chengdu	成都 1 ap-chengdu-1
	成都 2 ap-chengdu-2
中国香港・マカオ・台湾地区（中国香港） ap-hongkong	中国香港2区（中国香港ノードは中国香港・マカオ・台湾地区 ap-hongkong-2

その他の国と地域

リージョン	アベイラビリティーゾーン
アジア太平洋地区東南部（シンガポール） ap-singapore	シンガポール1区（シンガポールノードはアジア太平洋東南地区 ap-singapore-1
	シンガポール2区（シンガポールノードはアジア太平洋東南地区 ap-singapore-2
アジア太平洋地区東南部（ジャカルタ） ap-jakarta	ジャカルタ1区（ジャカルタノードはアジア太平洋東南地区をカ ap-jakarta-1
アジア太平洋地区東北部（ソウル） ap-seoul	ソウル1区（ソウルノードはアジア太平洋東北地区をカバーす ap-seoul-1
	ソウル2区（ソウルノードはアジア太平洋東北地区をカバーす ap-seoul-2

アジア太平洋地区東北部（東京） ap-tokyo	東京2区（東京ノードのアベイラビリティーゾーンはアジア太 ap-tokyo-2
アジア太平洋地区南部（ムンバイ） ap-mumbai	ムンバイ1区（ムンバイノードはアジア太平洋南部地区をカバ ap-mumbai-1
	ムンバイ2区（ムンバイノードはアジア太平洋南部地区をカバ ap-mumbai-2
アジア太平洋地区東南部（バンコク） ap-bangkok	バンコク1区（バンコクノードはアジア太平洋東南地区をカバ ap-bangkok-1
北アメリカ地域（トロント） na-toronto	トロント1区（トロントノードは北米地区をカバーするために na-toronto-1
米国東部（バージニア） na-ashburn	バージニア1区（バージニアノードは米国東部地区をカバーす na-ashburn-1
	バージニア2区（バージニアノードは米国東部地区をカバーす na-ashburn-2
欧州地区（フランクフルト） eu-frankfurt	フランクフルト1区（フランクフルトノードは欧州地区をカバ eu-frankfurt-1

リソース仕様

最終更新日：：2023-04-28 15:30:11

概要

TKE Serverlessクラスターを使用すると、クラスターノードを気にする必要はありませんが、合理的にリソースを割り当てて正確に計算するために、ワークロードをデプロイする時にPod申請のリソース仕様を指定する必要があります。Tencent Cloudは指定した仕様に基づいてワークロード計算リソースを割り当て、この仕様に基づいて料金の計算も行います。
Kubernetes APIまたはKubectlを使用してTKE Serverlessクラスターワークロードを作成する場合、Annotationによってリソース仕様を指定することができます。指定しない場合、TKE Serverlessクラスターはワークロードが設定したコンテナRequest、Limitなどのパラメータに基づいて計算を行います。詳細については、リソース仕様の指定をご参照ください。
注意
リソース仕様はPod内コンテナが使用できる最大リソース量です。
現在割り当てをサポートしているCPU、GPU仕様は下表をご参照ください。割り当てるときはサポートする仕様を超えないでください。
Pod内すべてのContainerに設定されるRequestの和は最大Pod仕様を超えることはできません。
Pod内の任意のContainerに設置されるLimitはPod仕様を超えることはできません。

CPU仕様

TKE Serverlessクラスターはすべてのリソースタイプをサポートするリージョンで以下のCPU Pod仕様を提供します。TKE Serverlessクラスターは一連のCPUオプションを提供し、異なるCPUサイズは異なるメモリ選択間隔に対応します。ワークロードを作成する時は実際のニーズに応じて最適な仕様を選択し、リソース割り当てを行ってください。

Intel

CPU/コア	メモリ間隔/GiB	メモリ間隔粒度/GiB
0.25	0.5、1、2	-
0.5	1、2、3、4	-
1	1 - 8	1
2	2、4 - 16	1

4	8 - 32	1
8	16 - 32	1
12	24 - 48	1
16	32 - 64	1

星星海AMD

Tencent Cloudが自社開発した星星海サーバーをベースとして、信頼性、安全性、安定性の高いパフォーマンスを提供します。詳細については、CVM標準型SA2の紹介をご参照ください。

CPU/コア	メモリ間隔/GiB	メモリ間隔粒度/GiB
1	1 - 4	1
2	2 - 8	1
4	4 - 16	1
8	8 - 32	1
16	32 - 64	1

GPU仕様

TKE Serverlessクラスターは以下の型番のGPU Pod仕様を提供し、異なるGPUカードの型番およびサイズは異なるCPU、メモリオプションに対応します。ワークロードを作成する時は実際のニーズに応じて最適な仕様を選択し、リソース割り当てを行ってください。

注意

yamlによって作成し、GPUのワークロードを管理および使用する必要があります。 Annotationの説明をご参照ください。

GPUモデル	GPU/カード	CPU/コア	メモリ/GiB
NVIDIA Tesla V100 - 1個	1	8	40
NVIDIA Tesla V100 - 2個	2	18	80
NVIDIA Tesla V100 - 4個	4	36	160
NVIDIA Tesla V100 - 8個	8	72	320
1/4 NVIDIA T4 - 1/4個	1	4	20

1/2 NVIDIA T4 - 1/2個	1	10	40
NVIDIA T4 - 1個	1	8	32
NVIDIA T4 - 1個	1	20	80
NVIDIA T4 - 1個	1	32	128
NVIDIA T4 - 2個	2	40	160
NVIDIA T4 - 4個	4	80	320
NVIDIA A10 - PNV4 - 1個	1	28	116
NVIDIA A10 - PNV4 - 2個	2	56	232
NVIDIA A10 - PNV4 - 4個	4	112	466
NVIDIA A10 - PNV4 - 8個	8	224	932
NVIDIA A10 - GNV4 - 1個	1	12	44
NVIDIA A10 - GNV4v - 1/4個	1	6	24
NVIDIA A10 - GNV4v - 1/2個	1	14	58
NVIDIA A10 - GNV4v - 1個	1	28	116

TKE Serverlessクラスター管理
クラスターライフサイクル

最終更新日：：2023-04-26 19:23:11

クラスターライフサイクルの説明

ステータス	説明
作成中	TKE Serverlessクラスターを作成中です。クラウドリソースを申請しています。
実行中	クラスターは正常に実行されています。
アイドル中	クラスターはアイドル状態です。
アクティベート中	クラスターをアイドル状態から実行中状態に調整しています。
削除中	クラスターを削除中です。同時にクラウドリソースを削除および解放します。
異常	ネットワークに接続できないなど、クラスターに異常が存在します。

クラスターのアイドル状態の説明

リソースのアイドルコストを削減し、ユーザーがクラスターリソースをより合理的に計画し管理できるようにするため、TKE Serverlessはアイドルクラスター回収機能をリリースしました。既存のPodがなく、かつ7日間連続してPodの作成/削除が行われていないクラスターをアイドルクラスターとみなし、アイドルクラスターとマークされたTKE Serverlessクラスターはステータスが「アイドル中」と表示されます。
アイドルクラスターを定義する条件：
ユーザー側で作成された既存のPodがないこと。
7日間連続してPodの作成/削除が行われていないこと。
前回クラスターがアイドル状態から正常な状態になってから、3日以上経過していること。
クラスターの新規作成から7日を超えていること。
ステータスが「アイドル中」のクラスターは、元のクラスターのすべての情報を維持していますが、クラスターは使用不可の状態であり、クラスターモニタリングの表示、クラスターの詳細の表示はサポートされず、APIServer関連の操作も制限されます。操作バーの**その他 > アクティベート**をクリックしてクラスターをウェイクアップすることも、**削除**をクリックしてクラスターを削除することも可能です。
アイドルクラスターのアクティベート後のクラスターは、元のクラスターと機能も設定もまったく同じものです。アクティベートされたクラスターを3日間連続してPodがない状態にし、かつPodの作成/削除操作を行わないと、

再びアイドルクラスターとしてマークされることに注意が必要です。

イメージキャッシュ

最終更新日：：2023-05-19 16:30:41

イメージキャッシュ概要

イメージキャッシュを使用してインスタンスを作成する時にイメージの取得が高速化され、インスタンスを起動する所要時間を減少させます。この機能はTKE ServerlessクラスターPodおよびスーパーノードに適用されます。ここではイメージキャッシュの動作原理、課金説明、作成および使用方法などについてご説明します。

動作原理

イメージキャッシュアクセラレーションがインスタンスを有効にする時は事前にコンテナインスタンスを起動してイメージの取得を行います。このイメージはサイズがカスタマイズ可能なデータディスク内に保存され、同時にこのデータディスクをクラウドディスクのスナップショットとしてキャッシュします。つまりイメージデータはスナップショット内に保存されます。コンテナインスタンスまたはPodを作成する時にイメージキャッシュの自動マッチングまたは手動マッチングを選択し、イメージのスナップショットに基づいて対応する数量のハードディスク（データディスク）を作成し、インスタンスにマウントされ、イメージ層のダウンロードを回避し、それによってコンテナインスタンスおよびPodの作成速度をアップさせます。
イメージの再利用 機能に比べ、イメージキャッシュの利点は次のとおりです。
時間制限がなく、イメージキャッシュ（スナップショット）のライフサイクルに関連しています。
事前にPodをコールドスタートするだけで、スナップショットを作成するとこのPodが破棄されます。
アベイラビリティーゾーンの制限がなく、スナップショットに基づいてクラウドディスクを作成する時に自動的にアベイラビリティーゾーンをマッチングします。
ワークロードの制限がなく、同じリージョン内でマッチングできます。

課金説明

イメージキャッシュを作成する時は、以下のリソースに関連します。対応する課金状況は次のとおりです。

課金項目	課金説明	課金
イメージキャッシュ	イメージキャッシュを作成する時は、2コア4GiBのコンテナインスタンスを実行してイメージを取得する必要があります。イメージキャッシュの作成が完了すると、そのコンテナインスタンスは自動的にリリースされて課金が停止します。	コン
CBSデータディスク	イメージキャッシュを作成する時は、高性能なデータディスクをバインドしてイメージを保存する必要があります。このデータディスクのサイズはカスタマイズをサポー	CBS

	トし、デフォルトは20Gです。イメージキャッシュの作成が完了すると、このデータディスクは自動的にリリースされて課金が停止します。	
スナップショット	上記のデータディスクに基づいてスナップショットを作成し、このスナップショットのライフサイクルはイメージキャッシュのライフサイクルに一致します。スナップショットは使用時間および容量に基づいて課金されます。	スナ

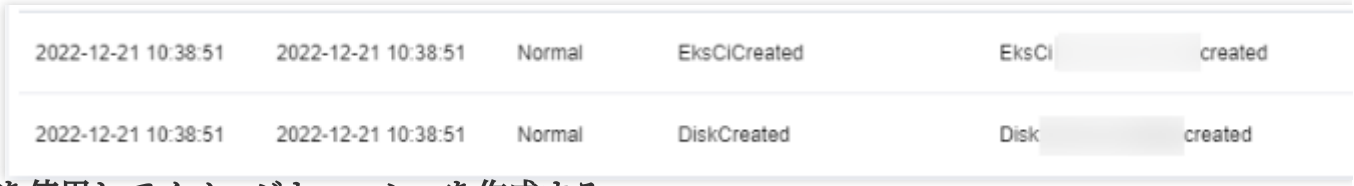
イメージキャッシュを使用する場合、マッチングしたイメージキャッシュのスナップショットに基づいて同じ容量の高性能データディスクを作成し、 Podにバインドするため、Pod自体を作成する料金に加え、追加で**データディスクの料金**が課金されます。
データディスク料金=容量 * *単価* * インスタンス実行時間

操作手順

コンソールを使用してイメージキャッシュを作成する

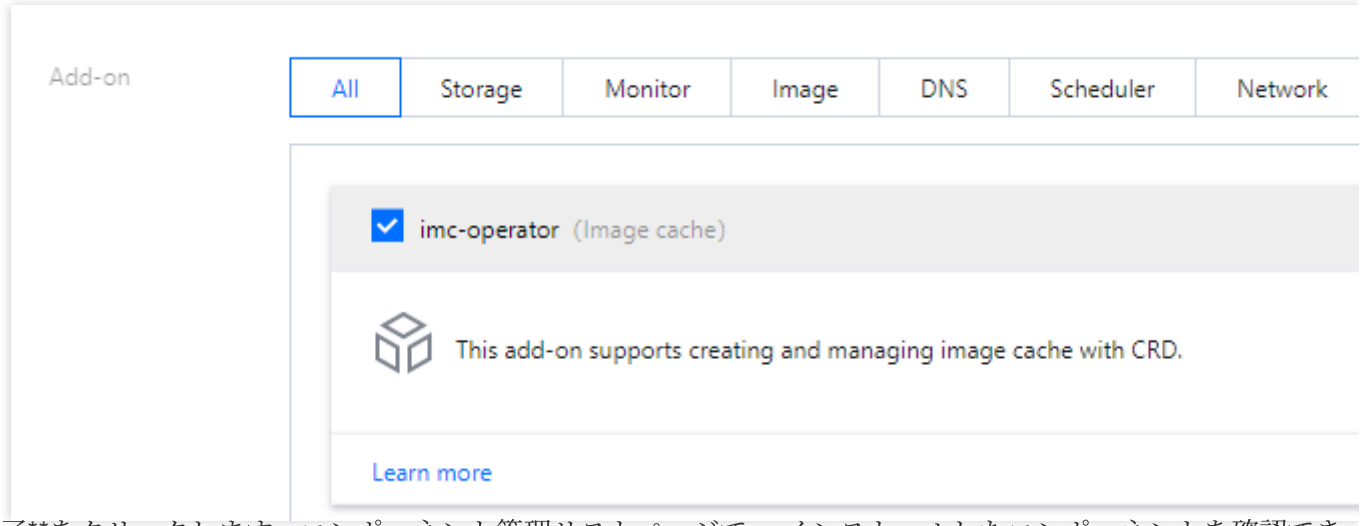
1. TKEコンソールにログインします。
2. 左側ナビゲーションバーの**アプリケーションセンター > イメージキャッシュ**を選択し、イメージキャッシュページで**インスタンスの新規作成**をクリックします。
3. **イメージキャッシュの作成**ページで、関連パラメータを設定します。
インスタンス名称：カスタマイズします。
所在リージョン：必要に応じて選択します。
コンテナネットワーク：コンテナネットワークアドレスの範囲内のIPアドレスをコンテナインスタンスに割り当てます。
セキュリティグループ：セキュリティグループはファイアウォールの機能があり、コンテナインスタンスのネットワーク通信を制限することができます。デフォルトはdefaultです。
OSタイプ：WindowsおよびLinuxの選択をサポートします。
イメージ：必要に応じてキャッシュするイメージおよびそのバージョンを選択します。
イメージ証明書：Dockerhubおよびその他のサードパーティのイメージウェアハウスのプライベートイメージを選択する場合、必ずイメージ証明書を入力する必要があります。すなわちウェアハウスのアクセスアドレス、ユーザー名およびパスワードです。
高度な設定：
キャッシュサイズ：このサイズはスナップショットおよびその後のインスタンスの作成時にバインドされるデータディスクのサイズを決定します。
注意：
イメージウェアハウスに表示されるイメージサイズは圧縮後のデータです。イメージキャッシュの作成にはイメージを抽出して展開するプロセスがあり、イメージが大きすぎるまたはイメージ圧縮比が大きすぎるとデフォルトの20GBのデータディスクでは不十分であるため、より大きなデータディスク空間を設定することをお勧めします。

期限切れポリシー：イメージキャッシュの保持時間を選択します。デフォルトは永久保持です。
4. **インスタンスの作成**をクリックします。作成が完了したらイメージキャッシュのリストで、イベント名をクリックして作成プロセスを確認できます。下図に示すとおりです。

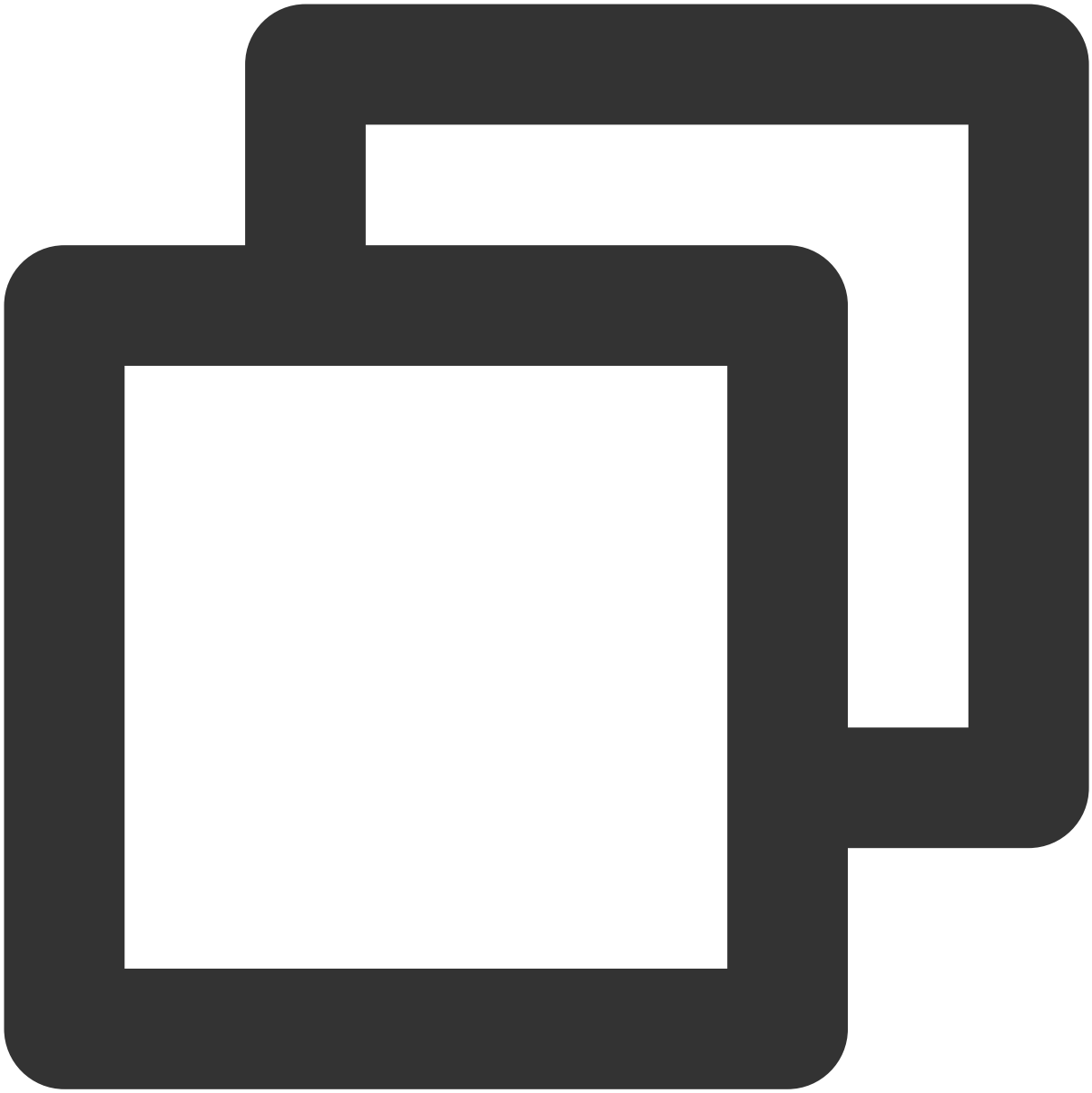


CRDを使用してイメージキャッシュを作成する

CRDを使用してイメージキャッシュサービスを作成する場合、クラスター内でイメージキャッシュコンポーネントをインストールし、インストール後にCRD+Controllerのモードでサポートし、Tencent Cloudイメージキャッシュサービスを使用します。Tencent Cloud APIインターフェースを呼び出す必要はありません。操作ステップは次のとおりです。
1. TKEコンソールにログインします。
2. クラスターリストでServerlessクラスターを選択し、クラスター詳細ページに進みます。
3. 左側ナビゲーションバーで**コンポーネント管理**を選択し、**新規作成をクリックします。**
4. **コンポーネント管理の新規作成**ページでimc-operator（イメージキャッシュ）コンポーネントを選択します。下図に示すとおりです。

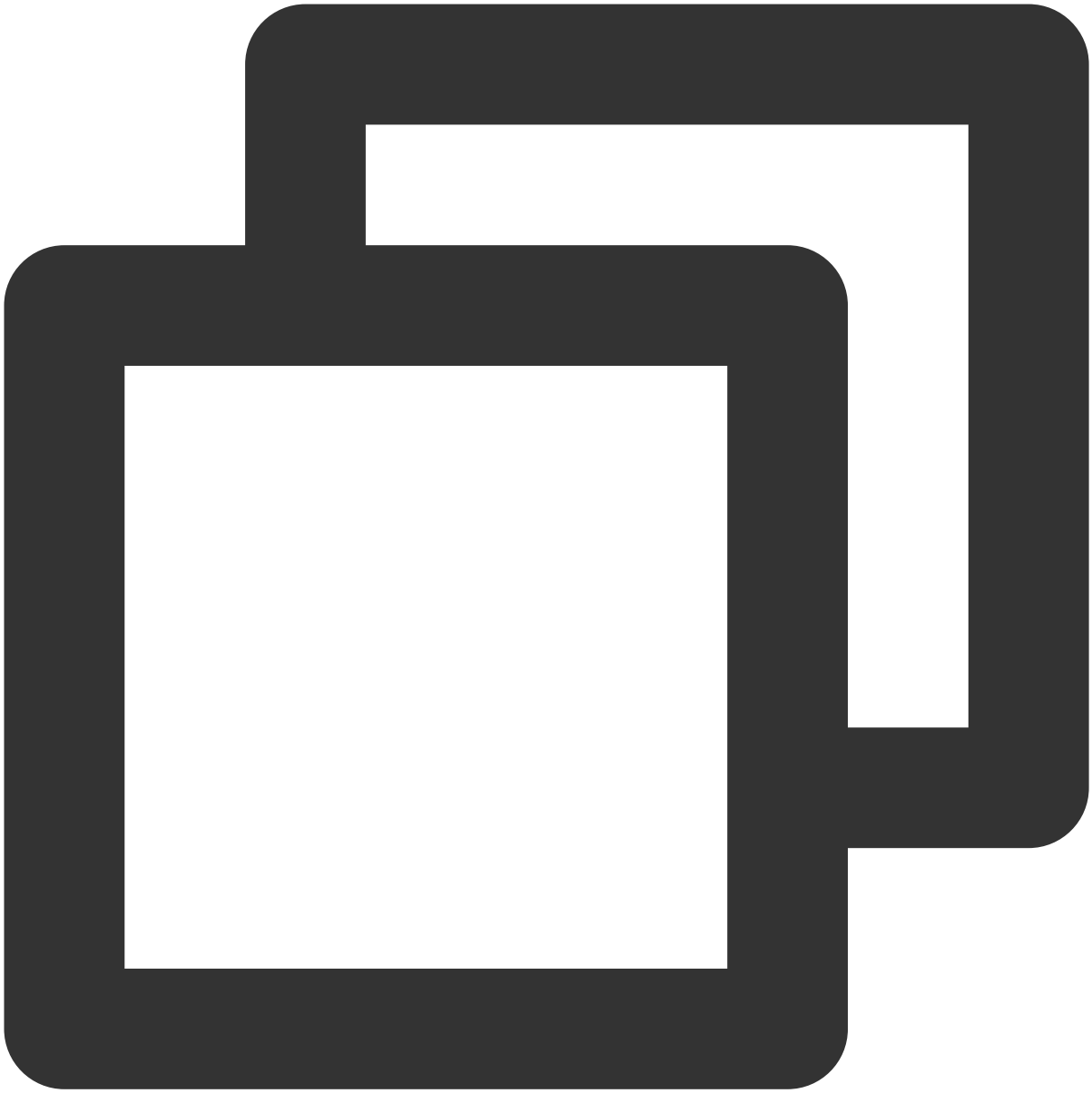


5. *完了**をクリックします。コンポーネント管理リストページで、 インストールしたコンポーネントを確認できます。
6. YAMLを編集します。 ImageCacheの作成：
ImageCacheを作成します
ImageCacheの確認
事例：



```
apiVersion: eks.cloud.tencent.com/v1
kind: ImageCache
metadata:
  name: imagecache-sample
spec:
  images:
    - nginx
#  imageCacheSize: 30
  # TODO(user): Add fields here
```

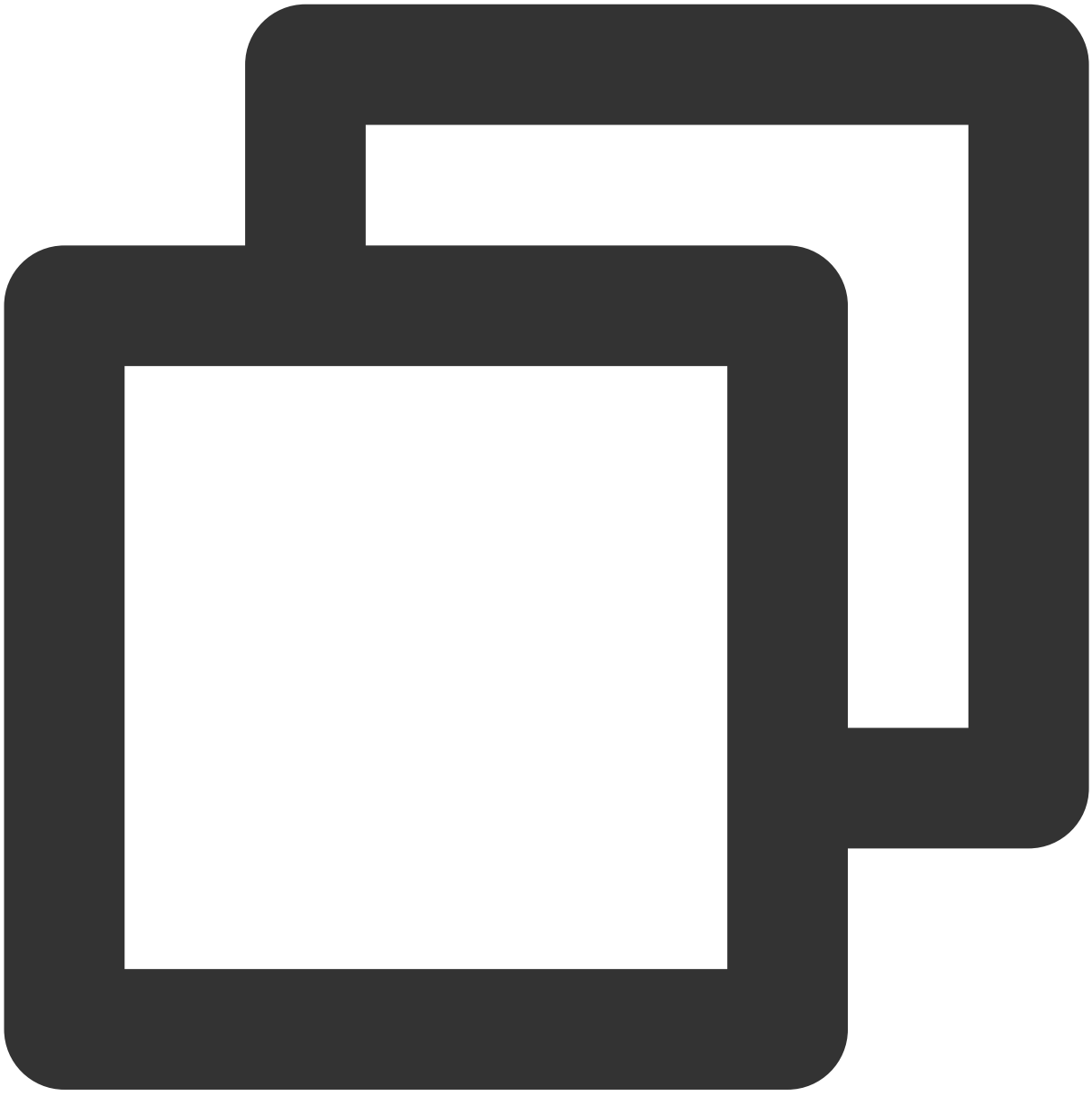
より多くのパラメータを持つ例：



```
apiVersion: eks.cloud.tencent.com/v1
kind: ImageCache
metadata:
  annotations:
    "eks.tke.cloud.tencent.com/eip-attributes": '{"InternetMaxBandwidthOut":2}' # e
  name: imagecache-sample-more-para
spec:
  images:
    - nginx
    - mysql
  imageCacheSize: 30
```

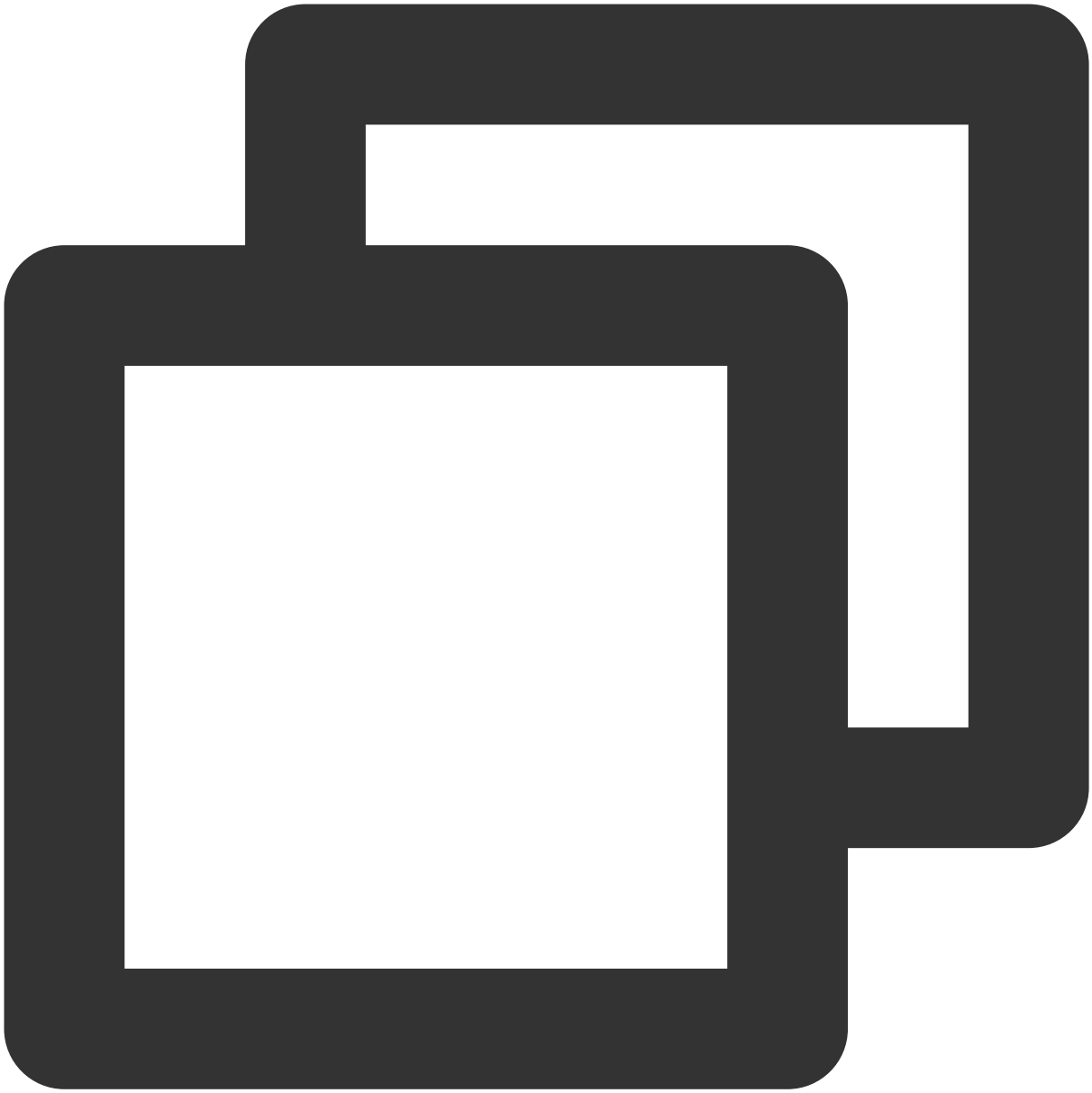
```
    retentionDays: 7
    imagePullSecrets:
      - imc-operator-system/qcloudregistrykey
```

事例：



```
kubectl get imc
```

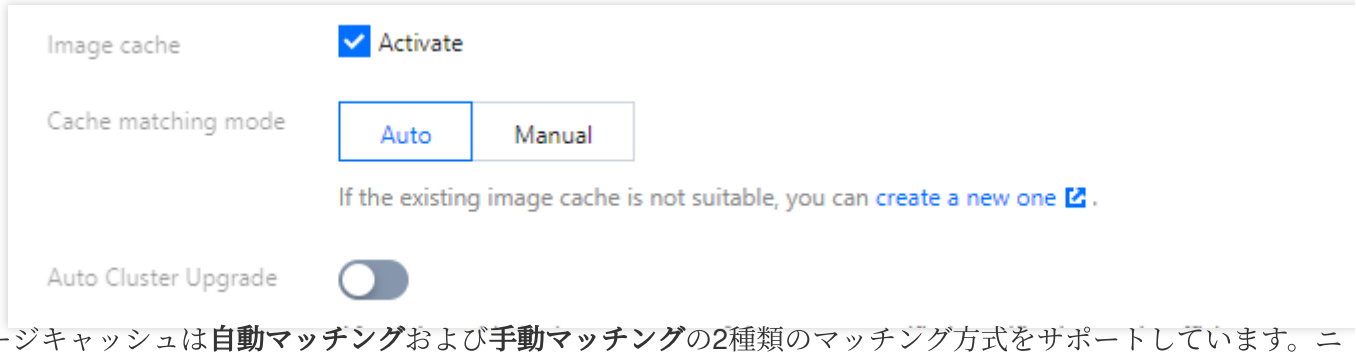
異常がある場合はeventsを確認できます。



```
kubectl describe imc xxx
```

作成したイメージキャッシュを使用する

Serverlessクラスター内でPodを作成する時は、**ワークロードの新規作成**ページで、**高度な設定の表示**をクリックし、イメージキャッシュ機能にチェックを入れて有効にします。下図に示すとおりです。



イメージキャッシュは**自動マッチング**および**手動マッチング**の2種類のマッチング方式をサポートしています。ニーズに応じて適切なマッチング方式を選択できます。

自動マッチング

手動マッチング

キャッシュのマッチングモードで**自動マッチング**を選択した場合、以下のマッチングポリシーに基づき、最適なイメージキャッシュを自動的にマッチングします。

イメージ名およびバージョンが完全に同じであれば、マッチングできます。

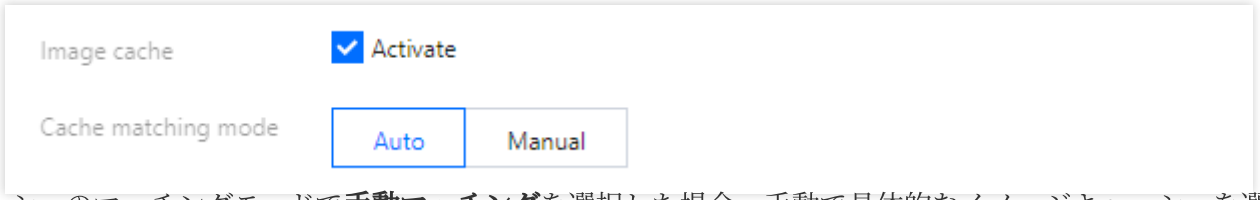
イメージのキャッシュサイズ。小容量のものが優先的にマッチングされます。

作成時間。作成時間が遅いものが優先的にマッチングされます。

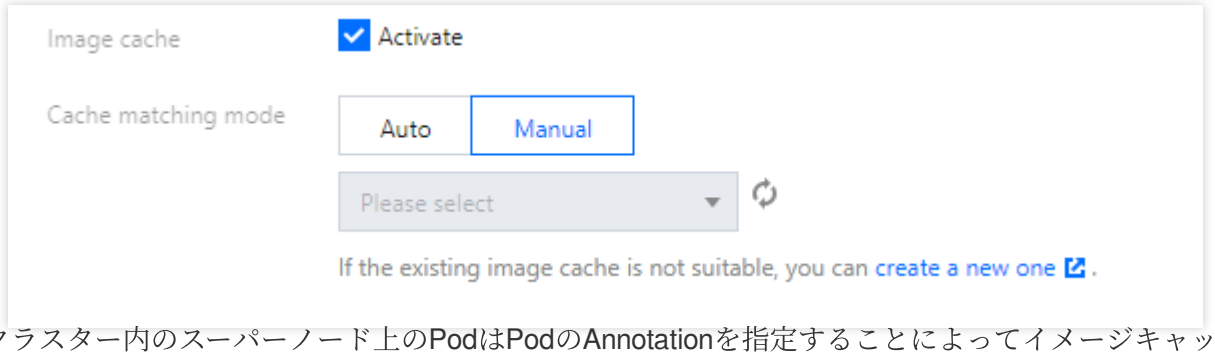
注意

双方がいずれもnginx: latestの場合でもマッチングできますが、作成時間が異なるため、バージョンが一致しない状況が存在する可能性があります。そのため、イメージキャッシュおよびインスタンスを作成する時に、バージョンを明確に記載することをお勧めします。

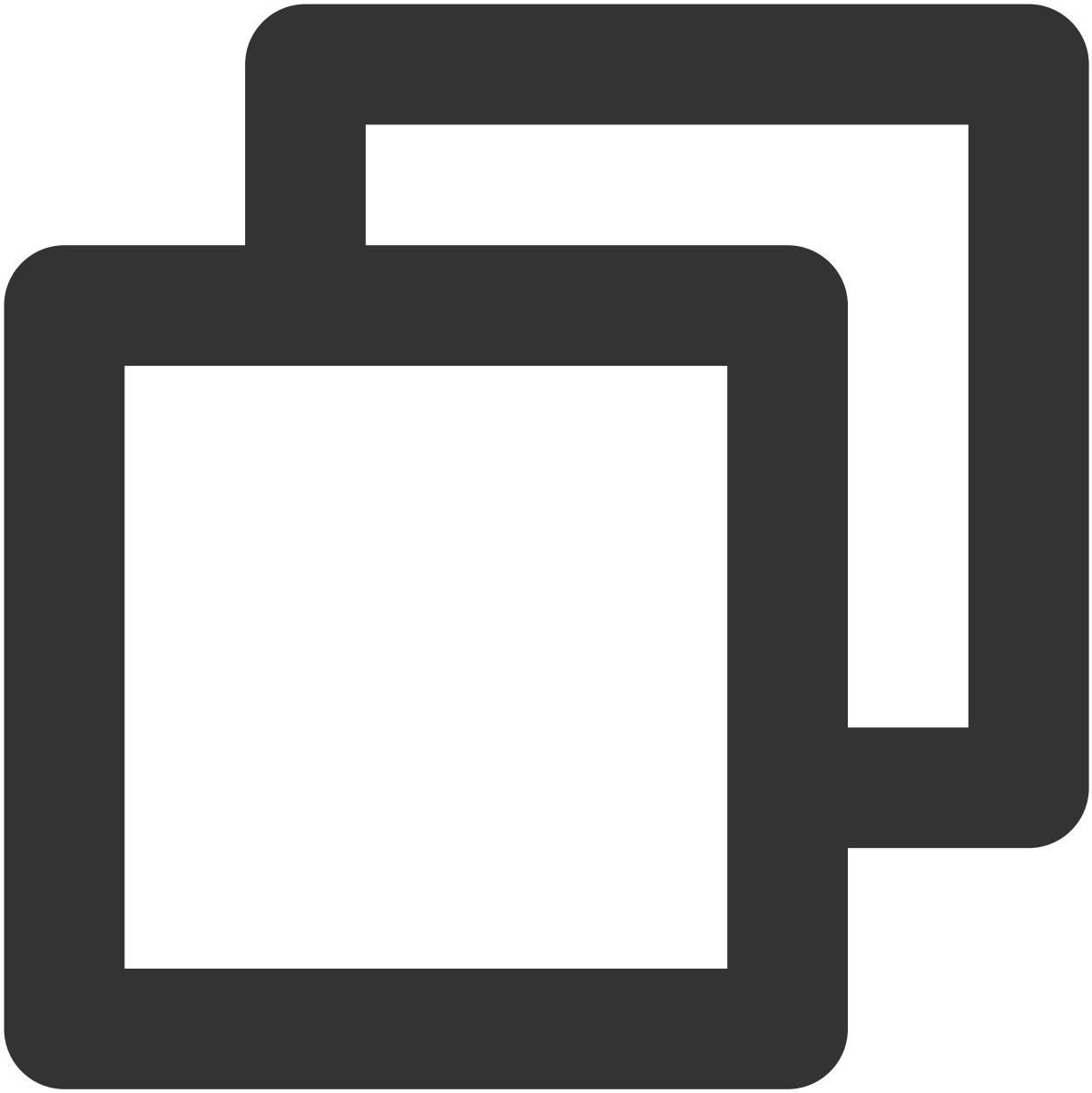
対応するイメージキャッシュにマッチングしない場合、自動的にイメージを正常に取得します。



キャッシュのマッチングモードで**手動マッチング**を選択した場合、手動で具体的なイメージキャッシュを選択する必要があります。手動でイメージキャッシュを指定した後、直接そのイメージキャッシュのスナップショットに基づいてデータディスクを作成してインスタンスにバインドしますが、データディスク中に作成時に入力されたイメージ（手動で指定した誤ったイメージキャッシュ）がない場合、新しく作成したデータディスク内でイメージを取得することにご注意ください。

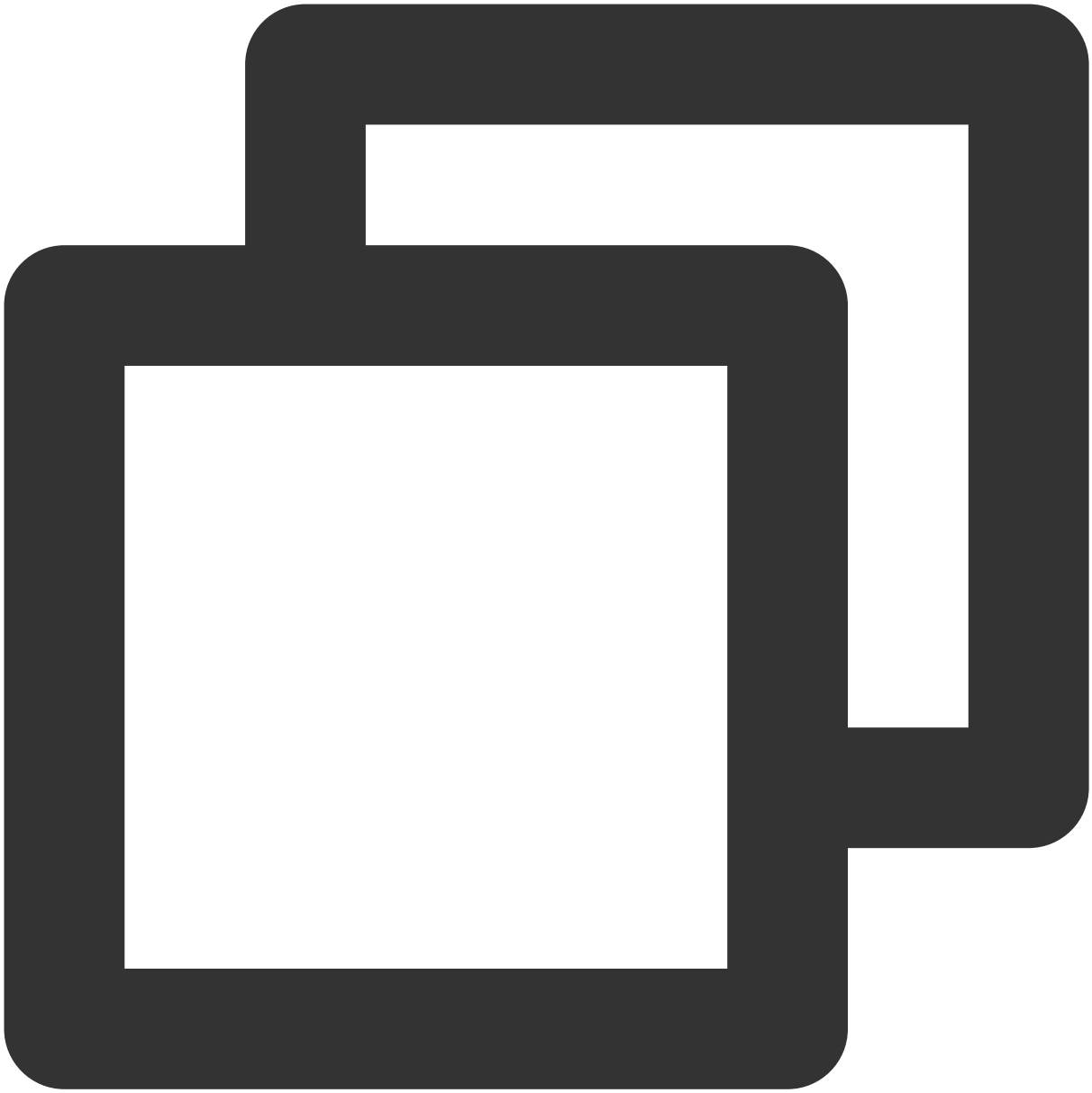


TKEクラスター内のスーパーノード上のPodはPodのAnnotationを指定することによってイメージキャッシュを使用することができます。具体的にはスーパーノード Annotationの説明をご参照ください。
自動マッチング：



```
eks.tke.cloud.tencent.com/use-image-cache: auto
```

手動で指定：



```
eks.tke.cloud.tencent.com/use-image-cache: imc-xxx
```

マッチング結果

インスタンスを作成するイベント内から、マッチングが成功したかどうかを確認することができます。
マッチングが成功した場合、以下のイベントが表示されます。

Started	Started container container
Created	Created container container
Pulled	Container image "nginx" already present on machine
ImageCacheUsed	Image cache imc-7fwlzosl used. Disk disk-dz7qm9i8 attache

このイベントが表示されない場合、適切なイメージキャッシュにマッチングしなかったことを表します。
手動マッチングを選択したのに、マッチングするイメージキャッシュ内にそのイメージがない場合、再度新しく作成したデータディスク内でイメージを取得し、以下のイベントが表示されることにご注意ください。

Started	Started container container
Created	Created container container
Pulled	Successfully pulled image "nginx" in 5.530971711s
Pulling	Pulling image "nginx"
ImageCacheUsed	Image cache imc-kcbee4nj used. Disk disk-d81vy7vw attached

運用保守センター
ログ採集
CRDを使用してCLSにログをキャプチャします
YAMLによってログのキャプチャを設定します

最終更新日：：2023-04-26 18:43:22

ここでは、YAML形式によりCRDを使用してTKE Serverlessクラスターのログキャプチャ機能を設定する方法についてご説明します。

前提条件

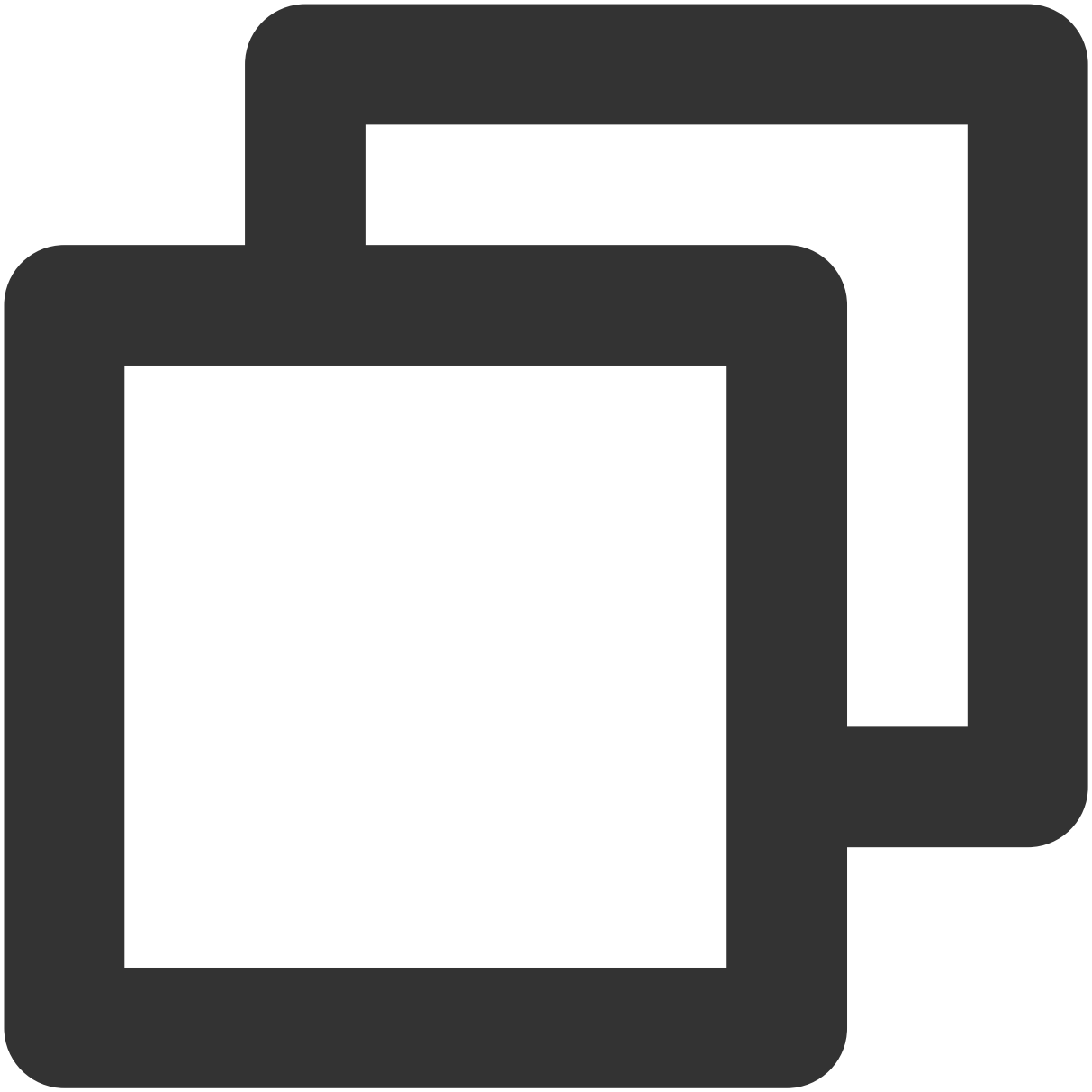
TKEコンソールにログインし、Serverlessクラスターのログキャプチャ機能を有効にします。操作の詳細については、ログキャプチャ機能の有効化をご参照ください。

CRDの作成

LogConfig CRDを定義するだけで、キャプチャ設定を作成することができます。キャプチャコンポーネントは、LogConfig CRDの変更に応じて対応するCloud Log Service(CLS)ログトピックを変更し、バインドされたマシングループを設定します。CRDの形式は次のとおりです。

clsDetailフィールドの説明

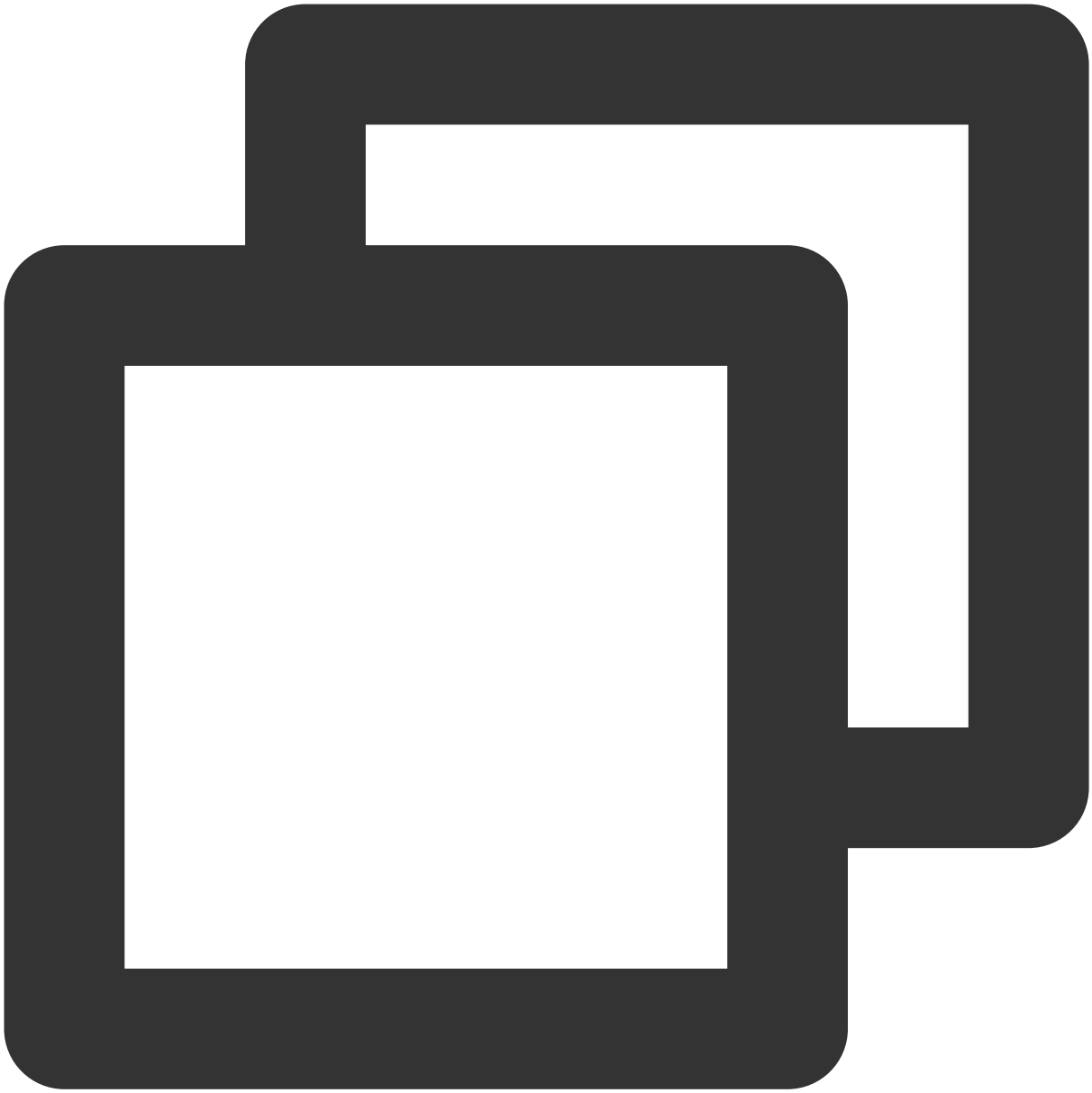
注意：
topic指定後の変更はできません。
選択したキャプチャタイプが「コンテナファイルパス」の場合、対応する「コンテナファイルパス」をソフトリンクにすることはできず、これを行った場合はソフトリンクの実際のパスがキャプチャツール内のコンテナに存在しなくなり、ログキャプチャに失敗します。



```
clsDetail:
  ## ログトピックを自動作成します。ログセットおよびトピックのnameを同時に指定する必要があります
  logsetName: test                    ## CLSログセットのname。このnameのログセットがなけれ
  topicName: test                     ## CLSログトピックのname。このnameのログトピックがな

  # 既存のログセットとログトピックを選択します。ログセットが指定されていて、ログトピックが指定されて
  logsetId: xxxxxx-xx-xx-xx-xxxxxxxx  ## CLSログセットのID。ログセットはCLS内にあらかじめ作
  topicId: xxxxxx-xx-xx-xx-xxxxxxxx   ## CLSログトピックのID。ログトピックはCLS内にあらかじ

  logType: json_log  ## ログキャプチャの形式。json_logはjson形式を表し、delimiter_logは区切
  logFormat: xxx                      ## ログのフォーマット方法
```

```
period: 30                                                    ## ライフサイクル。単位は日、可能
partitionCount:                          ## Integerタイプで、ログトピックのパーティション数で
tags:                                   ## タグ記述リスト。このパラメータを指定することで、タグを
 - key: xxx                              ## タグkey
 value: xxx                             ## タグvalue
autoSplit: false                                              ## booleanタイプ、自動分割を有効
maxSplitPartitions:
storageType: hot                       ## ログトピックのStorageタイプ。オプション値はhot（標準
excludePaths:                            ## ブラックリストパスリストのキャプチャ
  - type: File                                                        ##  タイプ、オプショ
      value: /xx/xx/xx/xx.log                ## typeの対応する値
indexs:                                 ## topic作成時にカスタマイズ可能なインデックス方式およ
  - indexName:     ## キー値またはメタフィールドインデックスのフィールドを設定する必要がありま
      indexType:     ## フィールドタイプ。現在はlong、text、doubleタイプをサポートしていま
      tokenizer:     ## フィールドの区切り文字。この中のそれぞれの文字が区切り文字を表します。
      sqlFlag:     ## boolean フィールドで分析機能を有効にしているかどうか
      containZH:   ## boolean 中国語が含まれるかどうか
region: ap-xxx                           ## topicの所在リージョン。クロスリージョン配信に使用
userDefineRule: xxxxxx                   ## ユーザー定義キャプチャルール。Json形式でシリアライ
extractRule: {}                          ## 抽出、フィルタリングルール。ExtractRuleを設定した
```

inputDetailフィールドの説明



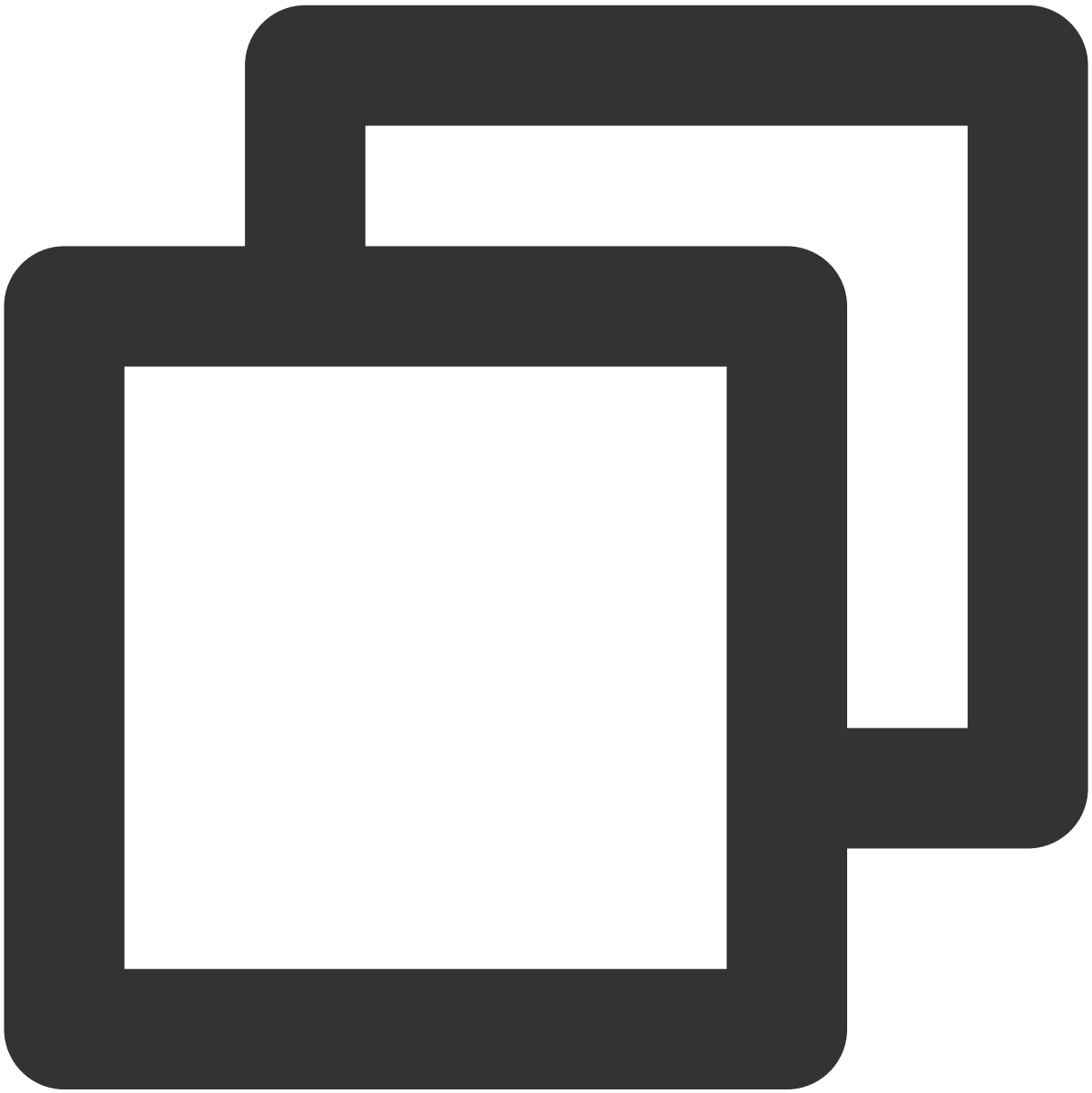
```
inputDetail:
  type: container_stdout  ## ログキャプチャのタイプ。container_stdout（コンテナ標準出力）、

  containerStdout:        ## コンテナ標準出力
    namespace: default    ## キャプチャコンテナのkubernetesネームスペース。複数のネームスペ
    excludeNamespace: nm1,nm2  ## キャプチャコンテナのkubernetesネームスペースを除外します。
    nsLabelSelector: environment in (production),tier in (frontend) ## ネームスペース
    allContainers: false       ## 指定のネームスペース内のすべてのコンテナの標準出力をキャプチ
    container: xxx             ## ログをキャプチャするコンテナ名。空の場合は、コンテナに合致
    excludeLabels:  ## キャプチャに指定のlabelを含むPodを含めません。workload、namespace、
      key2: value2  ## 同一のkey下にある複数のvalue値のpodのマッチングをサポートします。例えば
```

```
    includeLabels:  ## 指定のlabelを含むPodをキャプチャします。workload、namespace、exclu
      key: value1  ## キャプチャルールでキャプチャされたログにはmetadataが含まれ、コンシューマ

    metadataLabels:           ## 具体的にどのpod labelがメタデータとしてキャプチャされるかを
    - label1
    customLabels:             ## ユーザー定義のmetadata
      label: l1

    workloads:
    - container: xxx  ## キャプチャするコンテナ名。指定しない場合は、workload Pod内のすべての
      kind: deployment  ## workloadタイプ。deployment、daemonset、statefulset、job、cr
      name: sample-app  ## workloadの名前
      namespace: prod  ## workloadのネームスペース

  containerFile:  ## コンテナ内のファイル
    namespace: default      ## キャプチャコンテナのkubernetesネームスペース。1つのネームスペ
    excludeNamespace: nm1,nm2   ## キャプチャコンテナのkubernetesネームスペースを除外します
    nsLabelSelector: environment in (production),tier in (frontend) ## ネームスペース
    container: xxx          ## ログをキャプチャするコンテナ名。*の場合は、コンテナに合致するロ
    logPath: /var/logs      ## ログフォルダ。ワイルドカードはサポートしていません
    filePattern: app_*.log  ## ログファイル名。ワイルドカード*と?をサポートしています。*は複数
    customLabels:  ## ユーザー定義のmetadata
      key: value
    excludeLabels:  ## キャプチャに指定のlabelを含むPodを含めません。workloadとは同時に指定で
      key2: value2  ## 同一のkey下にある複数のvalue値のpodのマッチングをサポートします。例えば

    includeLabels:  ## 指定のlabelを含むPodをキャプチャします。workloadとは同時に指定できませ
      key: value1  ## キャプチャルールでキャプチャされたログにはmetadataが含まれ、コンシューマ
    metadataLabels:       ## 具体的にどのpod labelがメタデータとしてキャプチャされるかを指定し
    - label1              ## pod label
    workload:
      container: xxx    ## キャプチャするコンテナ名。指定しない場合はworkload Pod内のすべての
      name: sample-app  ## workloadの名前
```

ログ解析形式

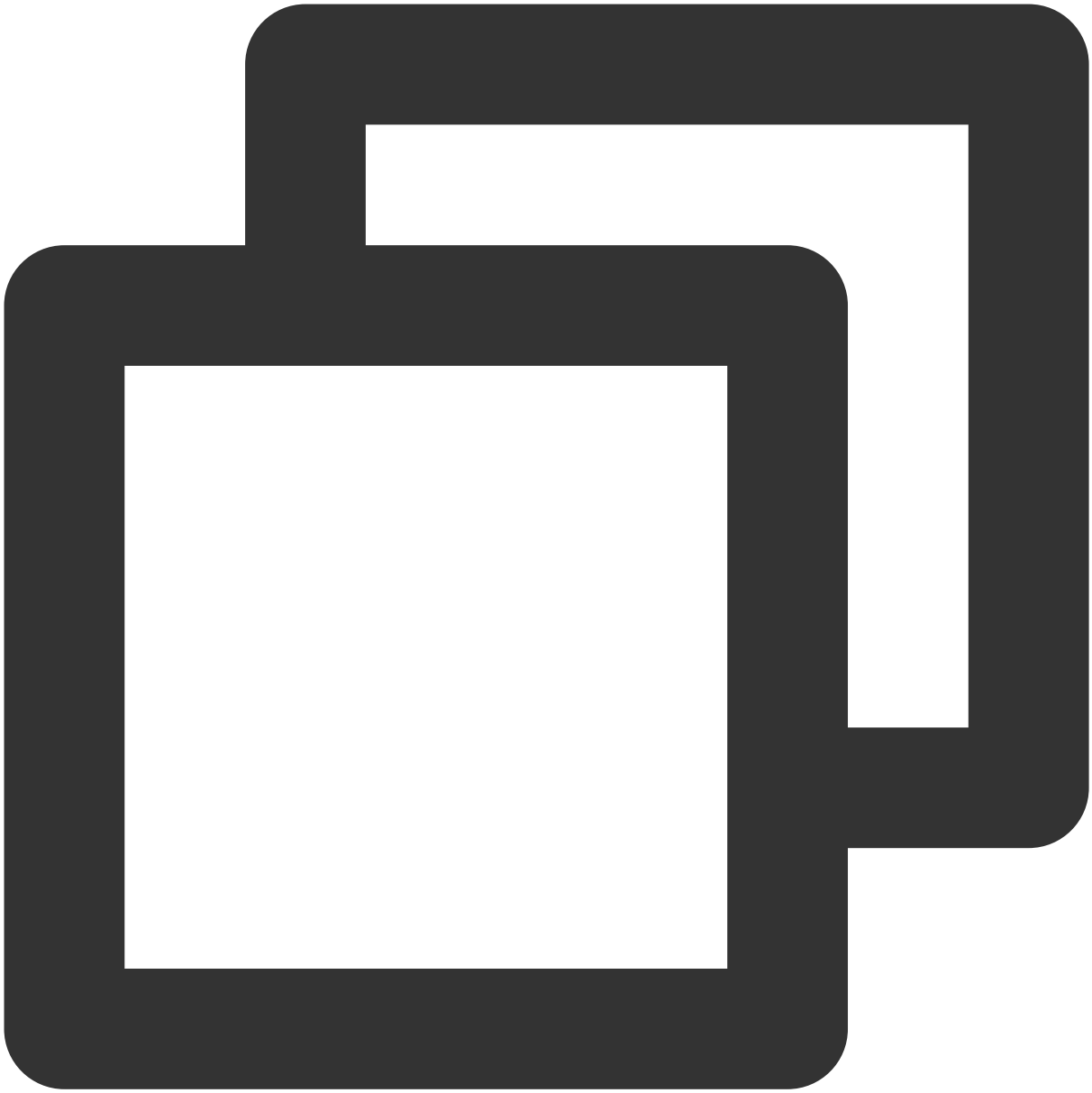
シングルライン全文形式
マルチライン全文形式
シングルライン-完全な正規表現形式
マルチライン-完全な正規表現形式
JSON形式
区切り文字形式

シングルライン全文ログとは、1行のログ内容が完全なログのことです。CLSがキャプチャする際に、改行文字 `\\n` を1行のログの末尾として使用します。構造化管理の一元化を図るため、各ログにはデフォルトのキー値 `__CONTENT__` がありますが、ログデータ自体はログ構造化処理が行われなくなり、ログフィールドも抽出されません。ログ属性の時間項目は、ログキャプチャの時間によって決まります。詳細については、シングルライン全文形式をご参照ください。
次のような1行のログのオリジナルデータがあると仮定します。



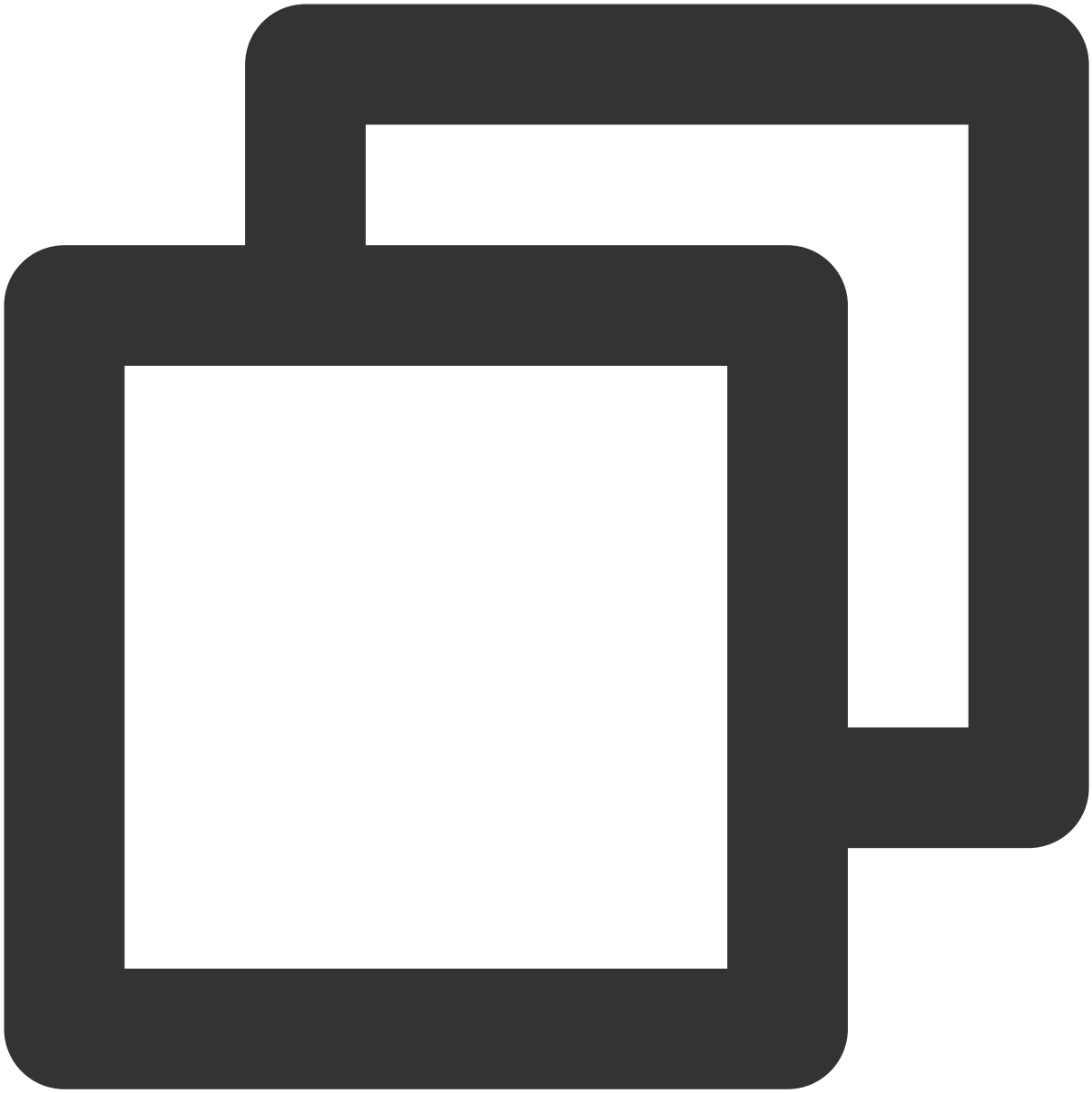
```
Tue Jan 22 12:08:15 CST 2019 Installed: libjpeg-turbo-static-1.2.90-6.el7.x86_64
```

LogConfigの設定の参照例は次のとおりです。



```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
spec:
   clsDetail:
     topicId: xxxxxx-xx-xx-xx-xxxxxxxx
     # シングルラインログ
     logType: minimalist_log
```

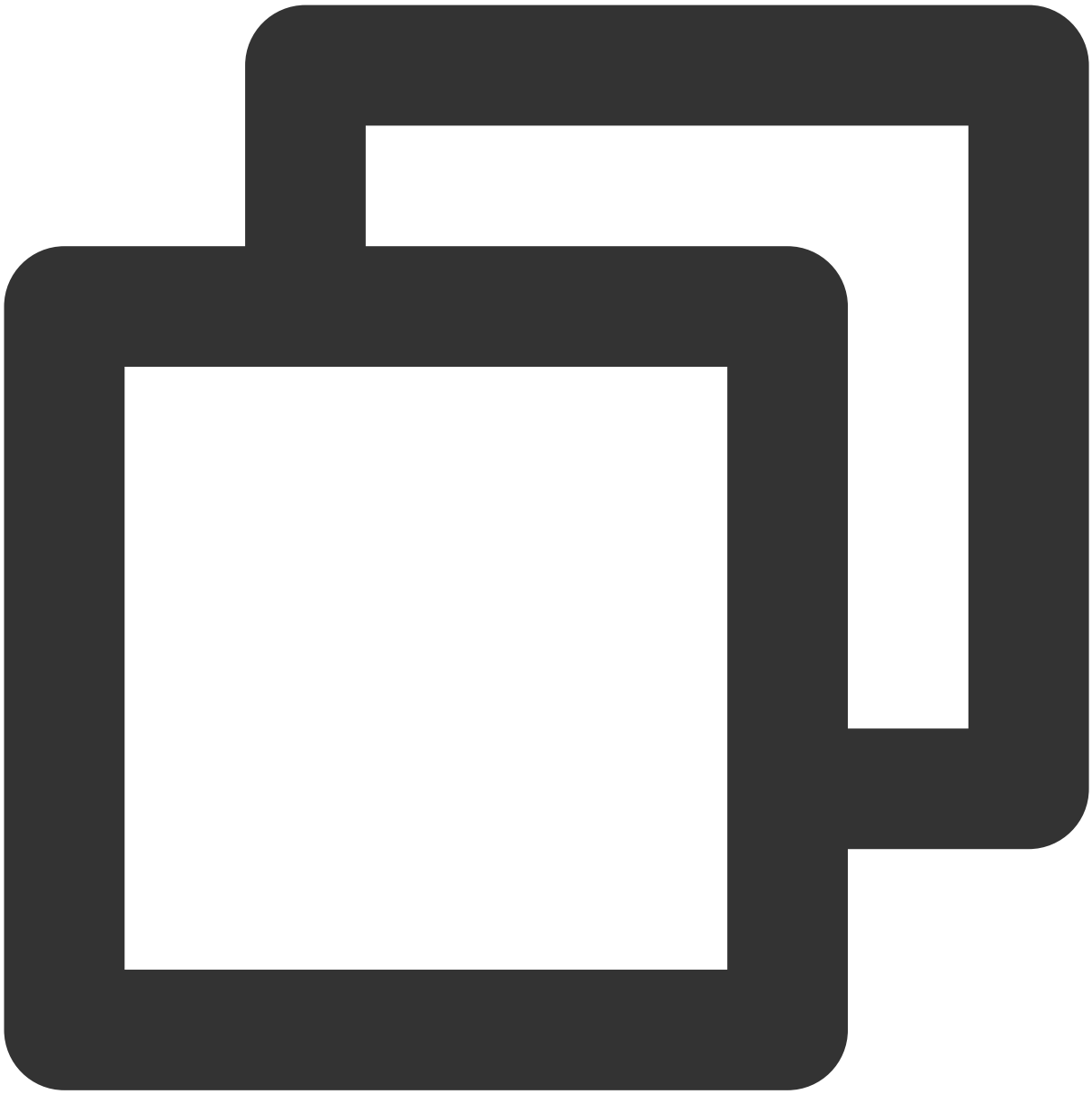
CLSにキャプチャされるデータは次のとおりです。



```
__CONTENT__:Tue Jan 22 12:08:15 CST 2019 Installed: libjpeg-turbo-static-1.2.90-6.e
```

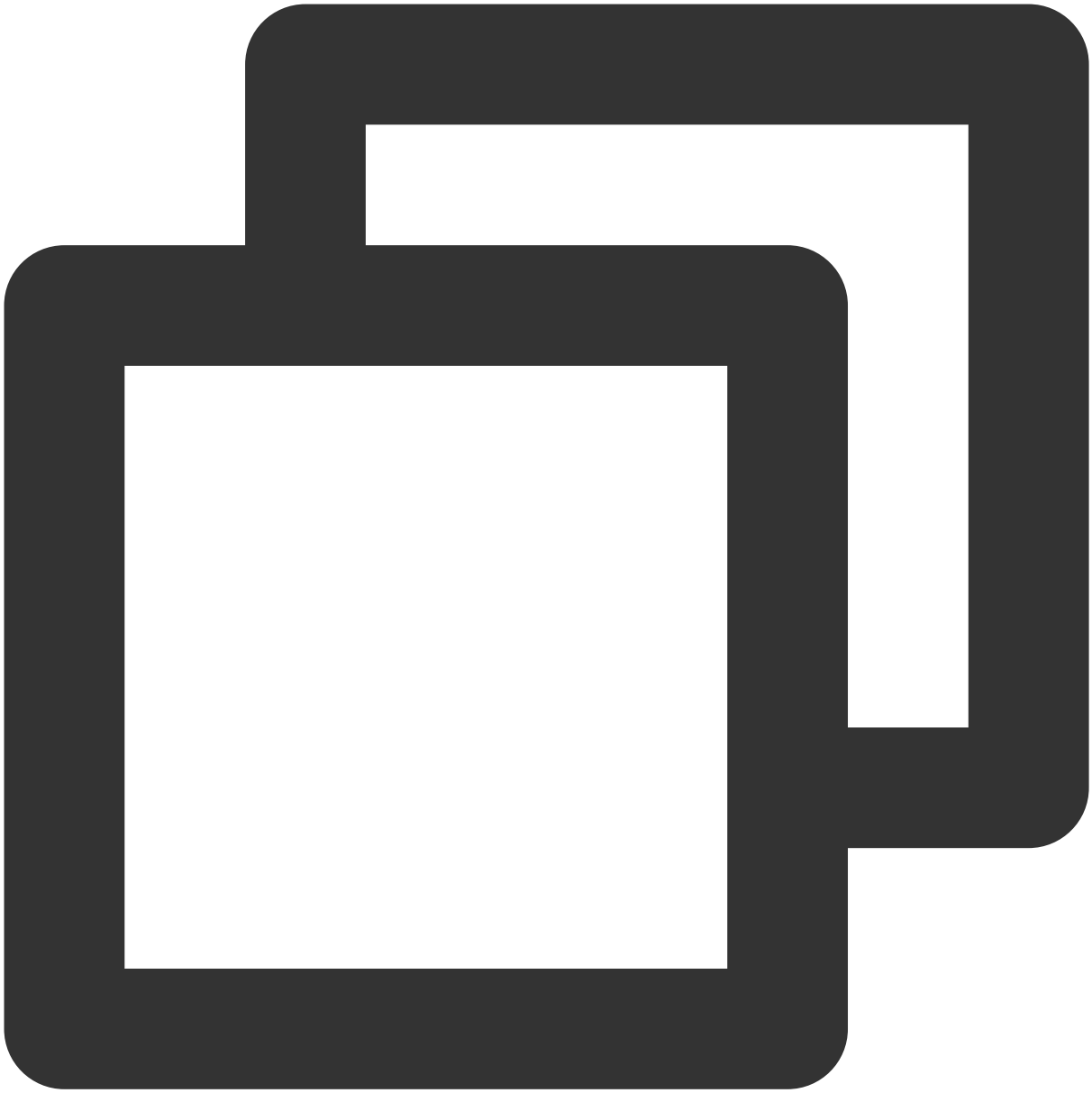
マルチライン全文ログとは、1行の完全なログデータで、複数行にまたがる可能性があるもののことです（例：Java stacktrace）。この場合、ログの終了識別子として改行文字 `n` を使用することはできません。ログシステムが各ログを明確に区別するために、最初の行の正規表現が使用されてマッチングが行われます。特定の行のログがあらかじめ設定された正規表現にマッチする場合、それがログの先頭になり、次の行に最初に出現したものがそのログの終了識別子となります。マルチライン全文でも、デフォルトのキー値 `__CONTENT__` が設定されますが、ログデータ自体はログ構造化処理が行われなくなり、ログフィールドも抽出されません。ログ属性の時間項目は、ログキャプチャの時間によって決まります。詳細については、マルチライン全文形式をご参照ください。

次のような1行のマルチラインログのオリジナルデータがあると仮定します。



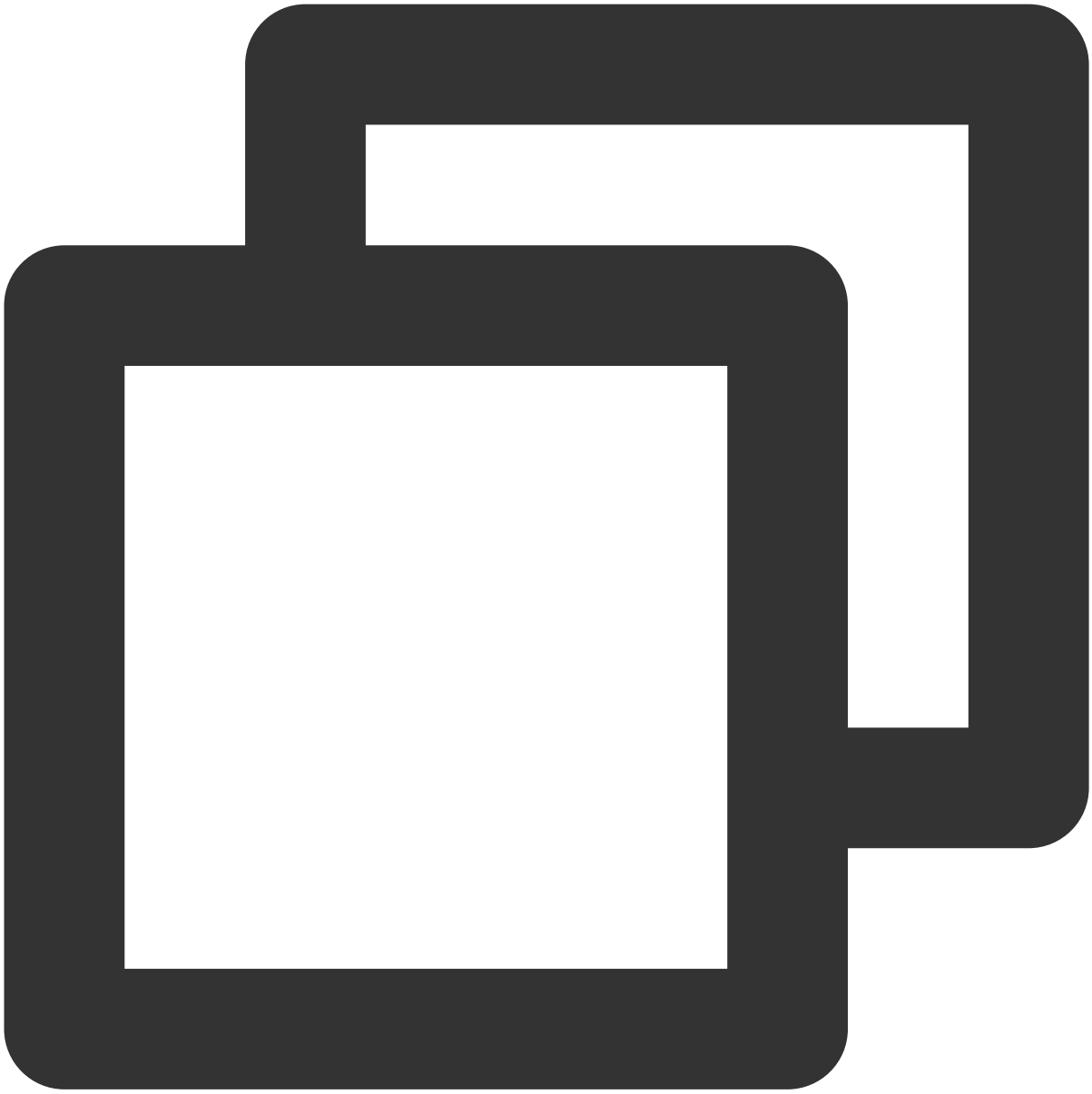
```
2019-12-15 17:13:06,043 [main] ERROR com.test.logging.FooFactory:
java.lang.NullPointerException
     at com.test.logging.FooFactory.createFoo(FooFactory.java:15)
     at com.test.logging.FooFactoryTest.test(FooFactoryTest.java:11)
```

LogConfigの設定の参照は次のとおりです。



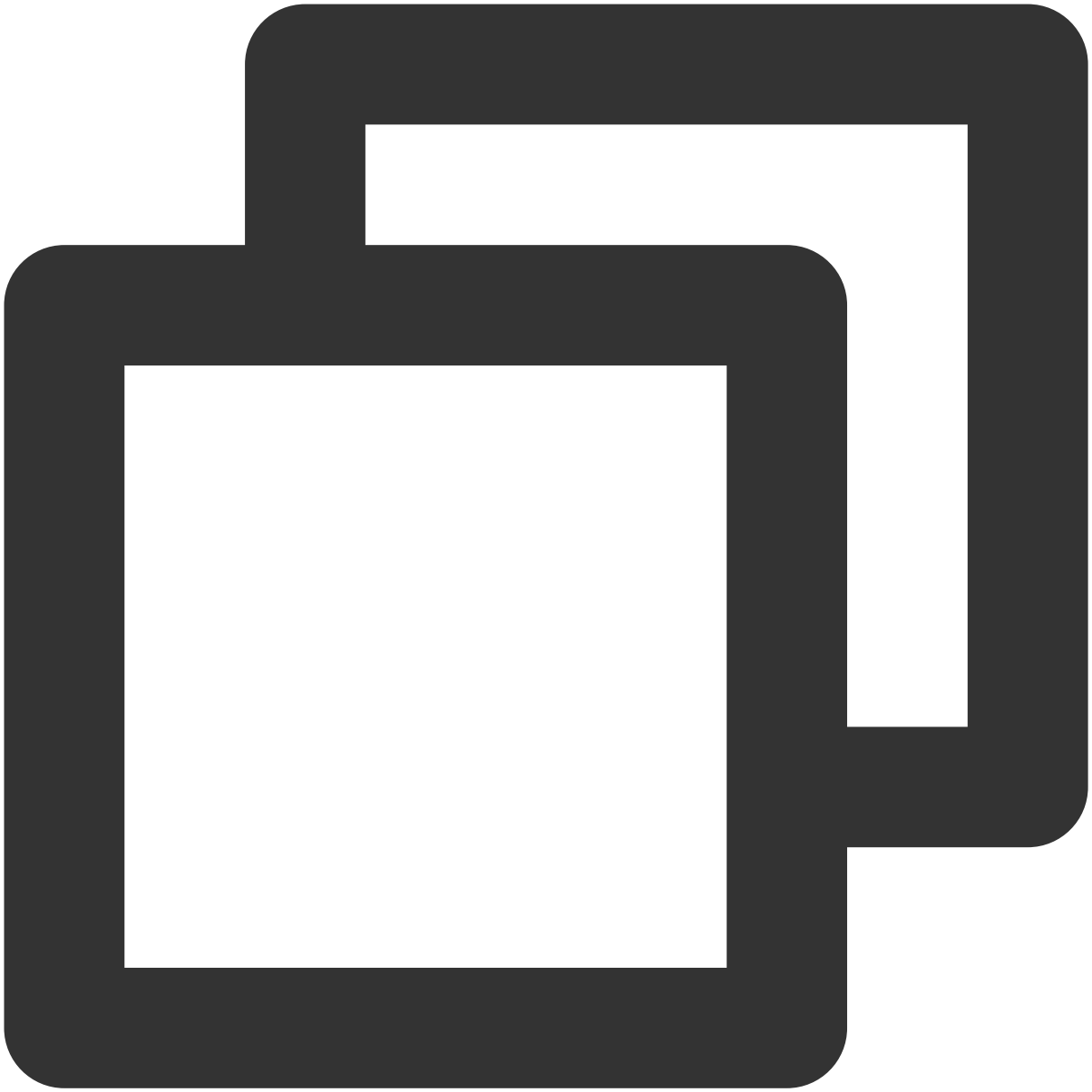
```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
spec:
   clsDetail:
     topicId: xxxxxx-xx-xx-xx-xxxxxxxx
     #マルチラインログ
     logType: multiline_log
     extractRule:
       #日時で始まる行のみを新しい1行のログの先頭とみなし、それ以外は改行文字\\nを加え、現在のロク
       beginningRegex: \\d{4}-\\d{2}-\\d{2}\\s\\d{2}:\\d{2}:\\d{2},\\d{3}\\s.+
```

CLSにキャプチャされるデータは次のとおりです。



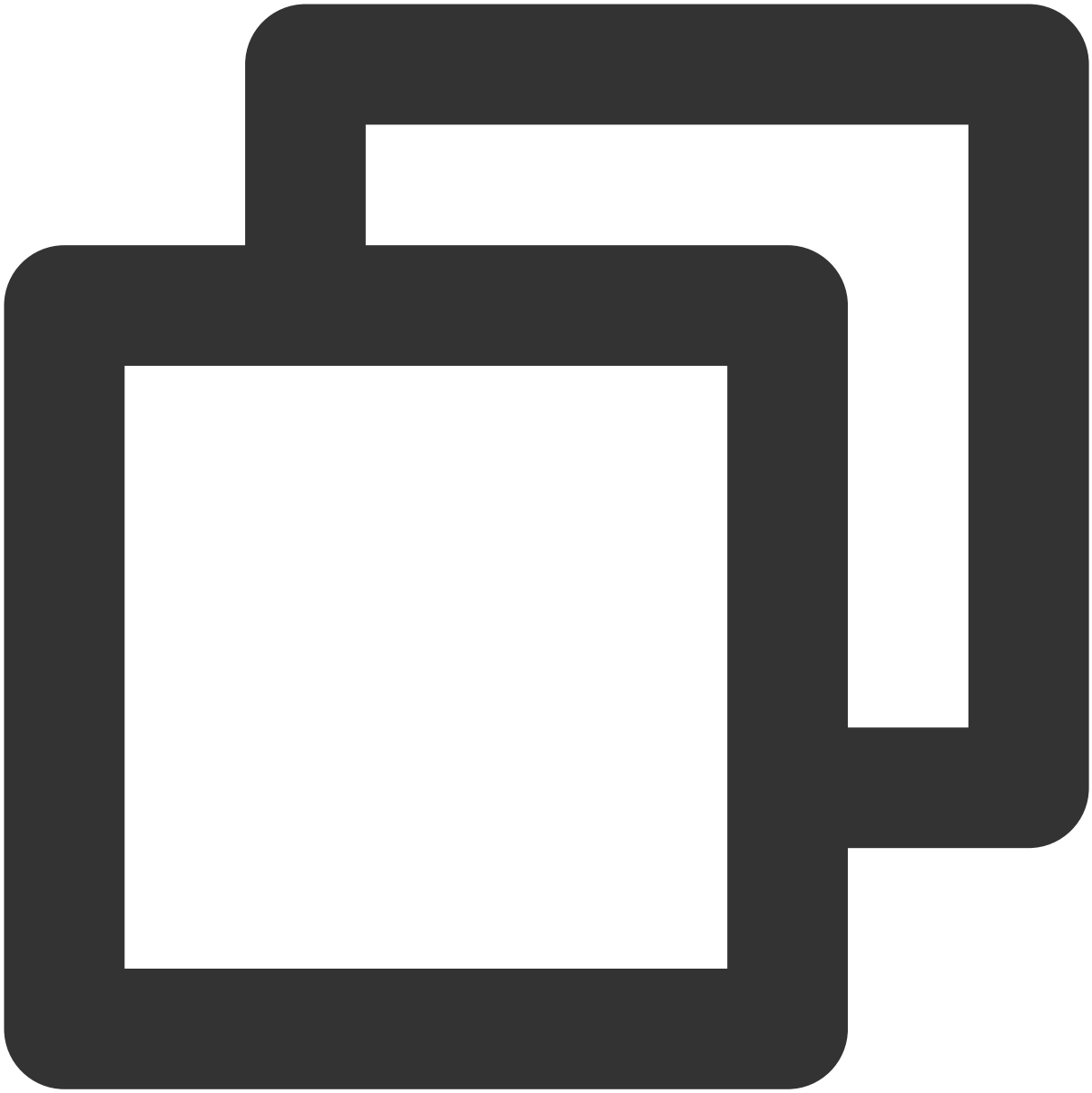
```
\\_\\_CONTENT__:2019-12-15 17:13:06,043 [main] ERROR com.test.logging.FooFactory:\\
```

完全な正規形式とは、通常、構造化処理されたログに使われ、完全な1行のログから複数のkey-valueを正規の方法で抽出するログ解析モードのことです。詳細については、完全な正規形式をご参照ください。
次のような1行のログのオリジナルデータがあると仮定します。



```
10.135.46.111 - - [22/Jan/2019:19:19:30 +0800] "GET /my/course/1 HTTP/1.1" 127.0.0.
```

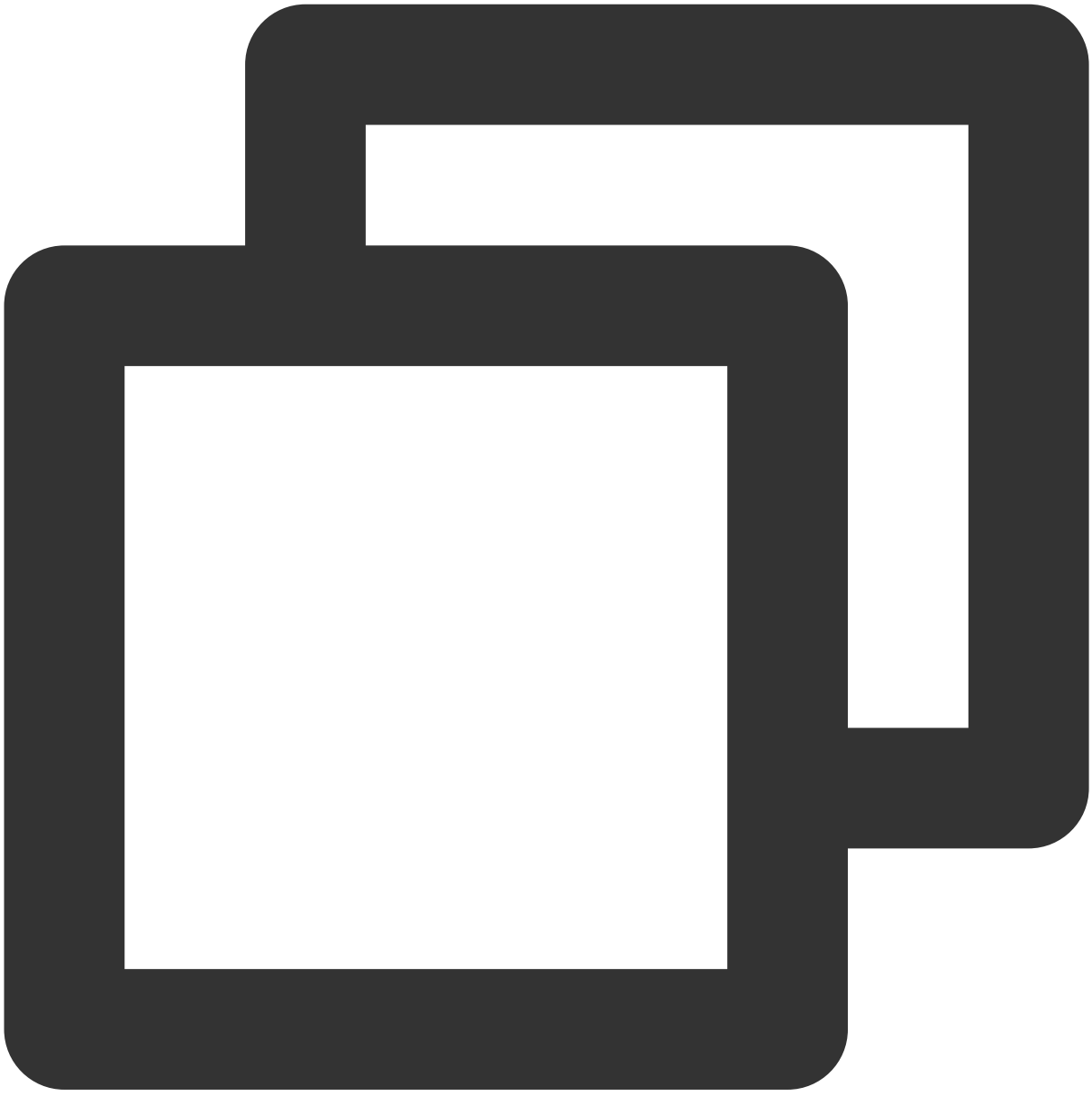
LogConfigの設定の参照は次のとおりです。



```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
spec:
   clsDetail:
     topicId: xxxxxx-xx-xx-xx-xxxxxxxx
     # 完全な正規表現形式
     logType: fullregex_log
     extractRule:
       # 正規表現、()のキャプチャグループに基づき、対応するvalueを抽出します
       logRegex: (\\S+)[^\\[]+(\\[[^:]+:\\d+:\\d+:\\d+\\s\\S+)\\s"(\\w+)\\s(\\S+)\\
       beginningRegex: (\\S+)[^\\[]+(\\[[^:]+:\\d+:\\d+:\\d+\\s\\S+)\\s"(\\w+)\\s(\
```

```
        # 抽出されたkeyリスト、抽出されたvalueと逐一対応します
        keys:   ['remote_addr','time_local','request_method','request_url','http_pro
```

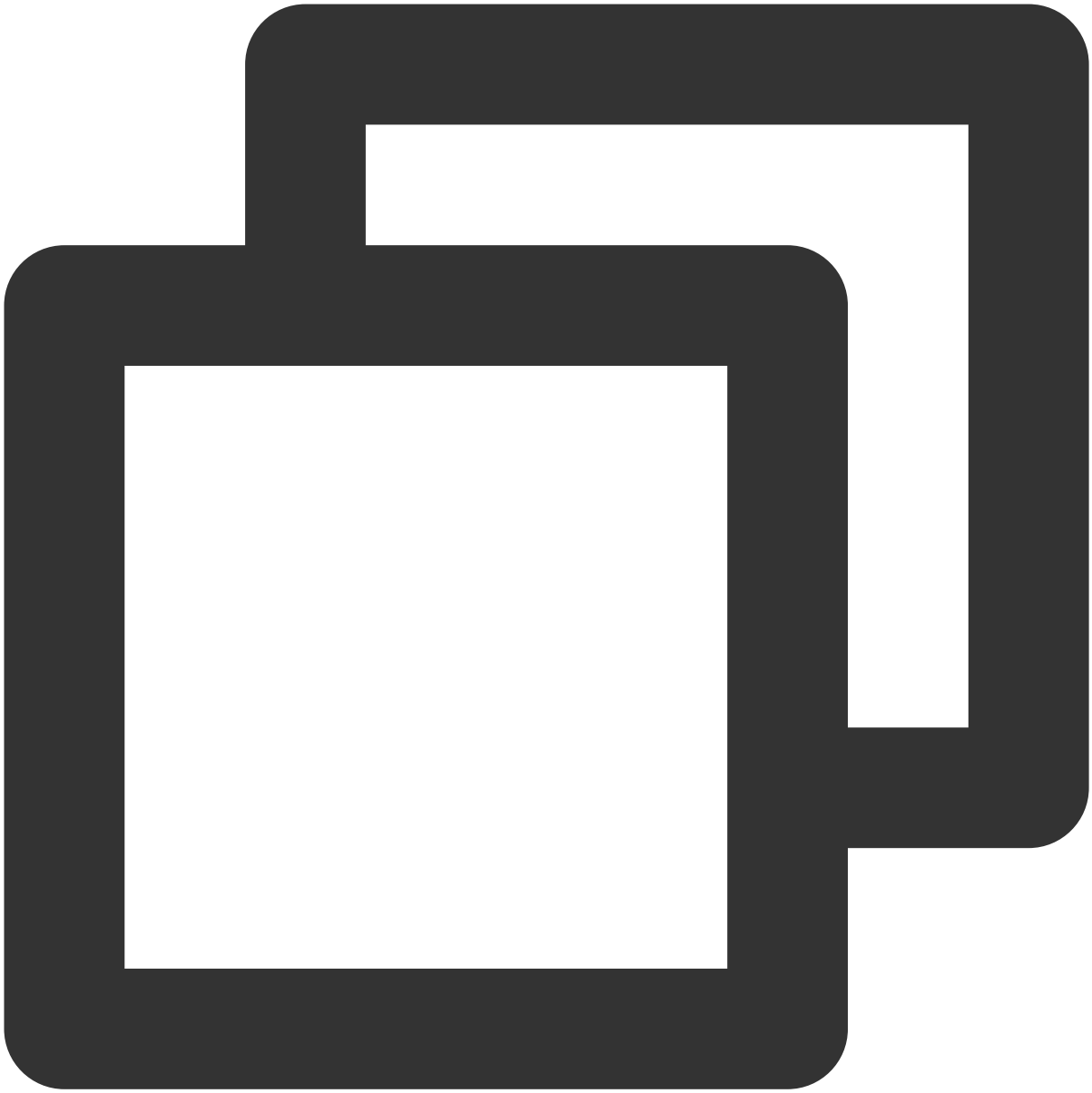
CLSにキャプチャされるデータは次のとおりです。



```
body_bytes_sent: 9703
http_host: 127.0.0.1
http_protocol: HTTP/1.1
http_referer: http://127.0.0.1/course/explore?filter%5Btype%5D=all&filter%5Bprice%5
http_user_agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firef
remote_addr: 10.135.46.111
```

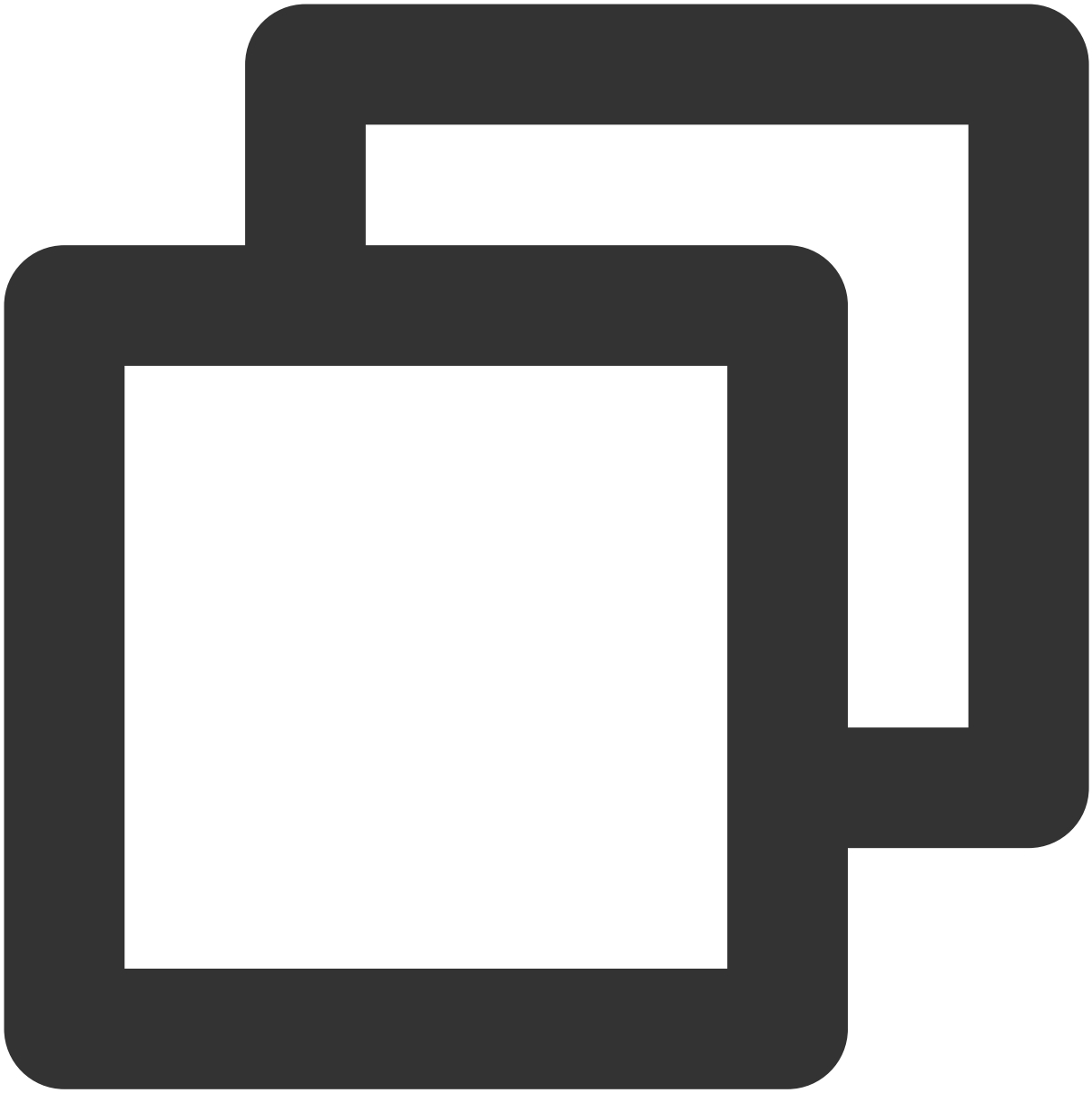
```
request_length: 782
request_method: GET
request_time: 0.354
request_url: /my/course/1
status: 200
time_local: [22/Jan/2019:19:19:30 +0800]
upstream_response_time: 0.354
```

マルチライン-完全な正規表現モードは、ログテキスト内で複数行にまたがる1行の完全なログデータ（例：Javaプログラムログ）に適した、正規表現によって複数のkey-valueキー値として抽出できるログ解析モードです。key-valueの抽出が不要な場合は、マルチライン全文形式を参照して設定を行ってください。詳細については、マルチライン-完全な正規表現形式をご参照ください。
次のような1行のログのオリジナルデータがあると仮定します。



```
[2018-10-01T10:30:01,000] [INFO] java.lang.Exception: exception happened
   at TestPrintStackTrace.f(TestPrintStackTrace.java:3)
   at TestPrintStackTrace.g(TestPrintStackTrace.java:7)
   at TestPrintStackTrace.main(TestPrintStackTrace.java:16)
```

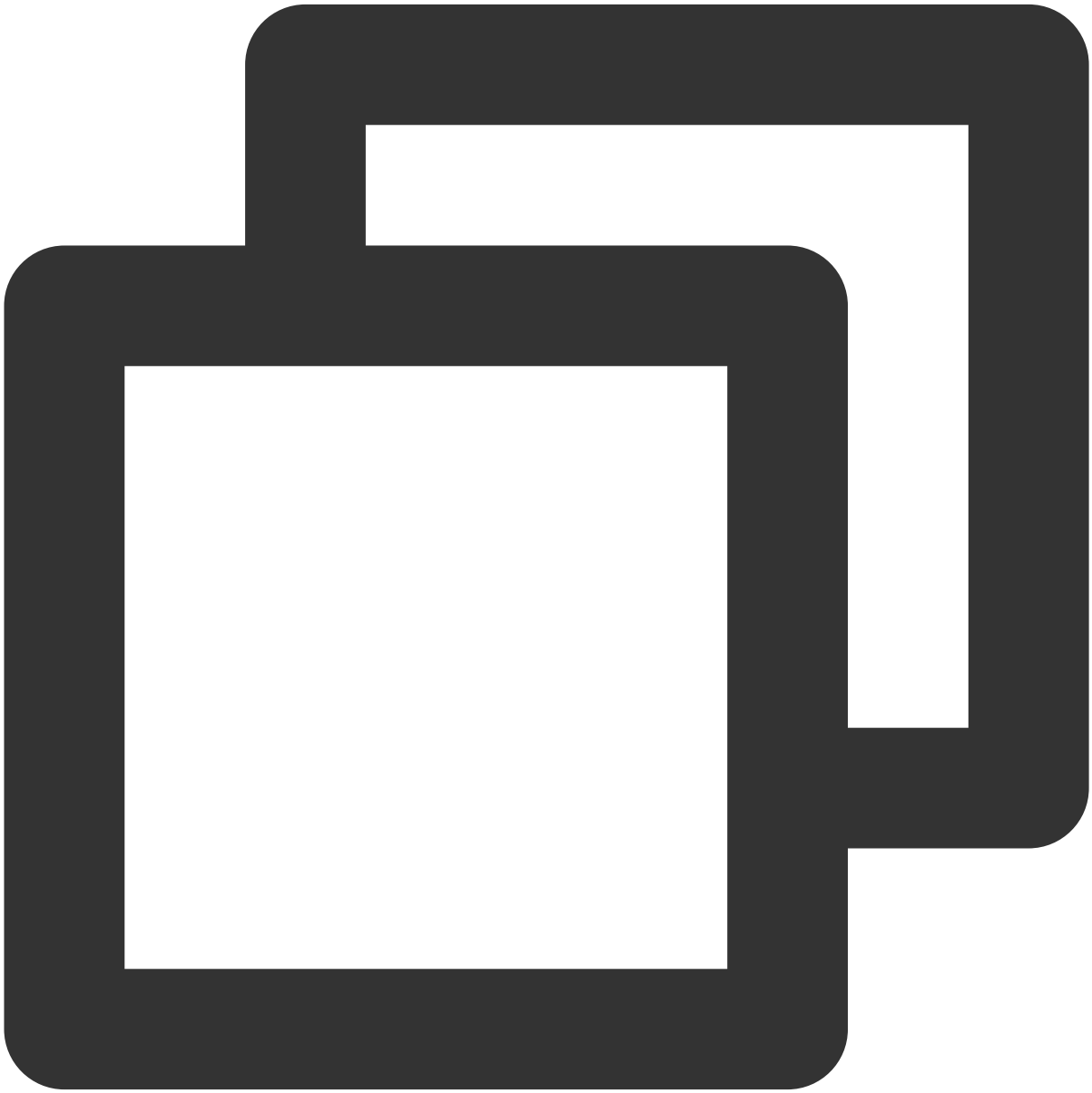
LogConfigの設定の参照は次のとおりです。



```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
spec:
  clsDetail:
    topicId: xxxxxx-xx-xx-xx-xxxxxxxx
        #マルチライン-完全な正規表現形式
        logType: multiline_fullregex_log
        extractRule:
      #行頭の完全な正規表現です。日時で始まる行のみを新しい1行のログの先頭とみなし、それ以外は改行
          beginningRegex: \\[\\d+-\\d+-\\w+:\\d+:\\d+,\\d+\\]\\s\\[\\w+\\]\\s.*
      #正規表現、()のキャプチャグループに基づき、対応するvalueを抽出します
```

```
        logRegex: \\[(\\d+-\\d+-\\w+:\\d+:\\d+,\\d+)\\]\\s\\[(\\w+)\\]\\s(.*)
              # 抽出されたkeyリスト、抽出されたvalueと逐一対応します
              keys:
        - time
        - level
        - msg
```

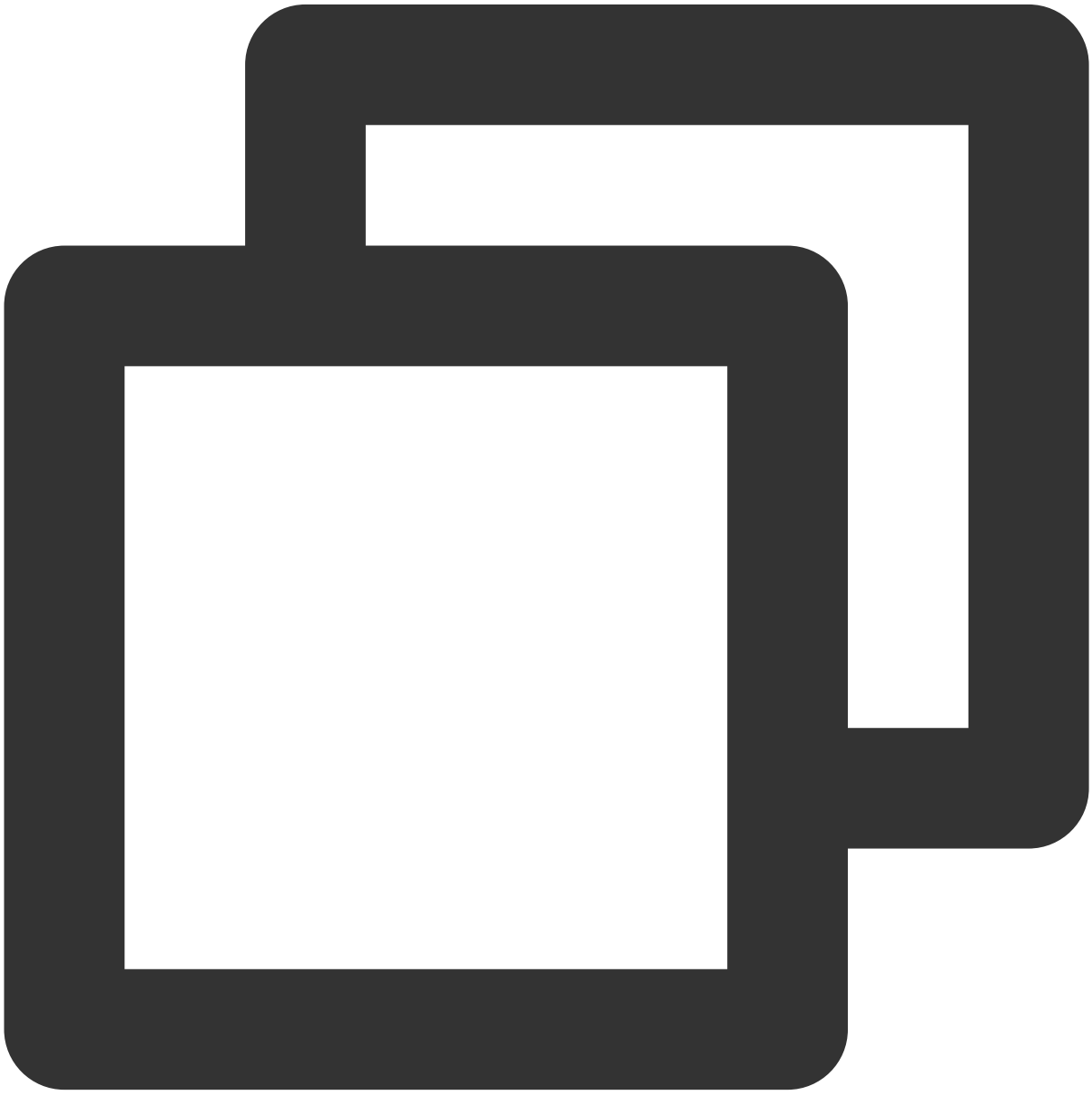
抽出されたkeyに基づき、CLSにキャプチャされたデータは次のとおりです



```
time: 2018-10-01T10:30:01,000`
level: INFO`
```

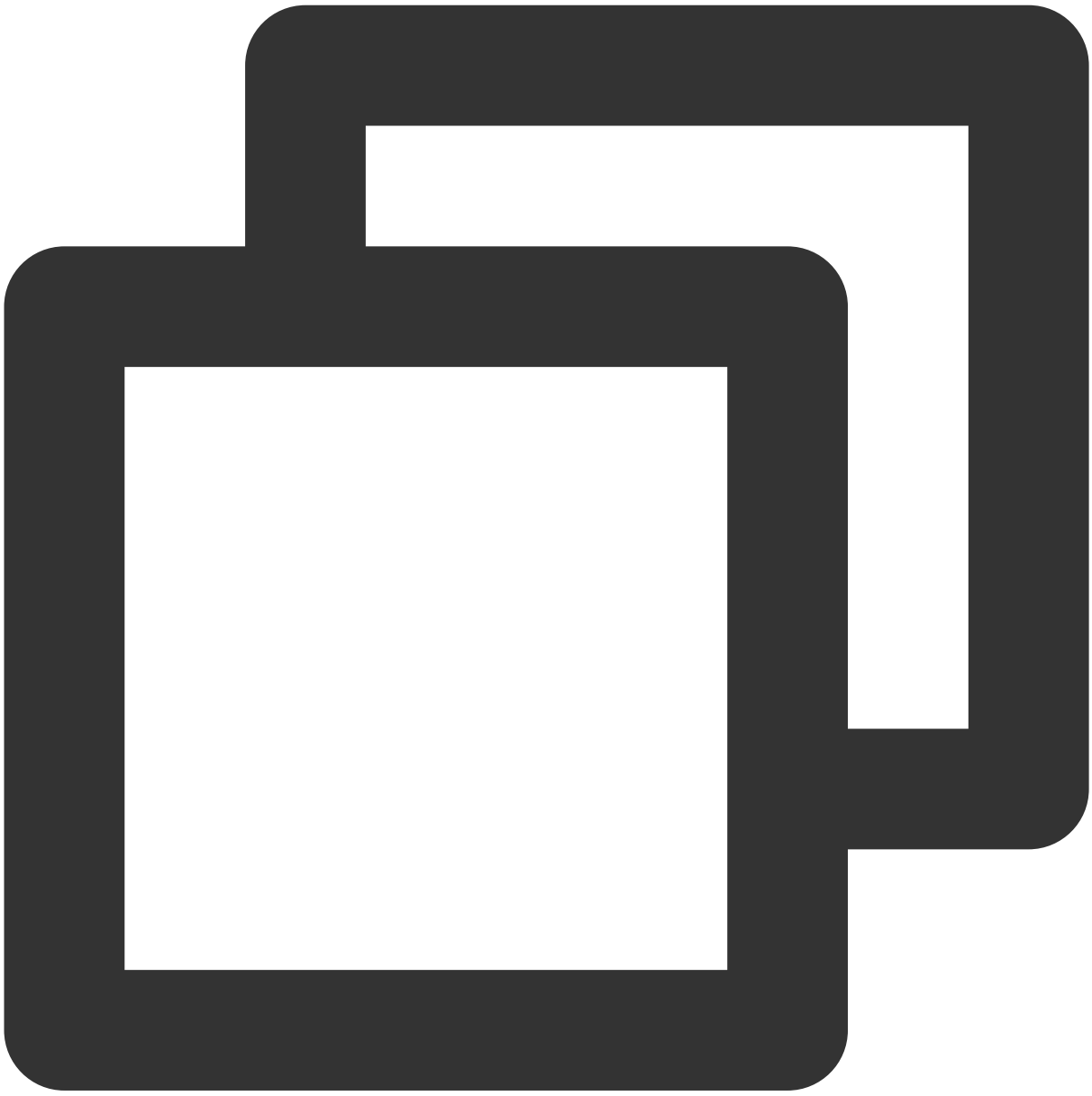
```
msg:java.lang.Exception: exception happened
    at TestPrintStackTrace.f(TestPrintStackTrace.java:3)
    at TestPrintStackTrace.g(TestPrintStackTrace.java:7)
    at TestPrintStackTrace.main(TestPrintStackTrace.java:16)
```

JSON形式のログは、第1階層のkeyを対応するフィールド名として自動的に抽出します。第1階層のvalueは対応するフィールドの値とします。このような方法によりログ全体の構造化処理を行い、それぞれの完全なログは改行文字 `\\n` を終了識別子とします。詳細については、JSON形式をご参照ください。
次のような1行のJSONログのオリジナルデータがあると仮定します。



```
{"remote_ip":"10.135.46.111","time_local":"22/Jan/2019:19:19:34 +0800","body_sent":
```

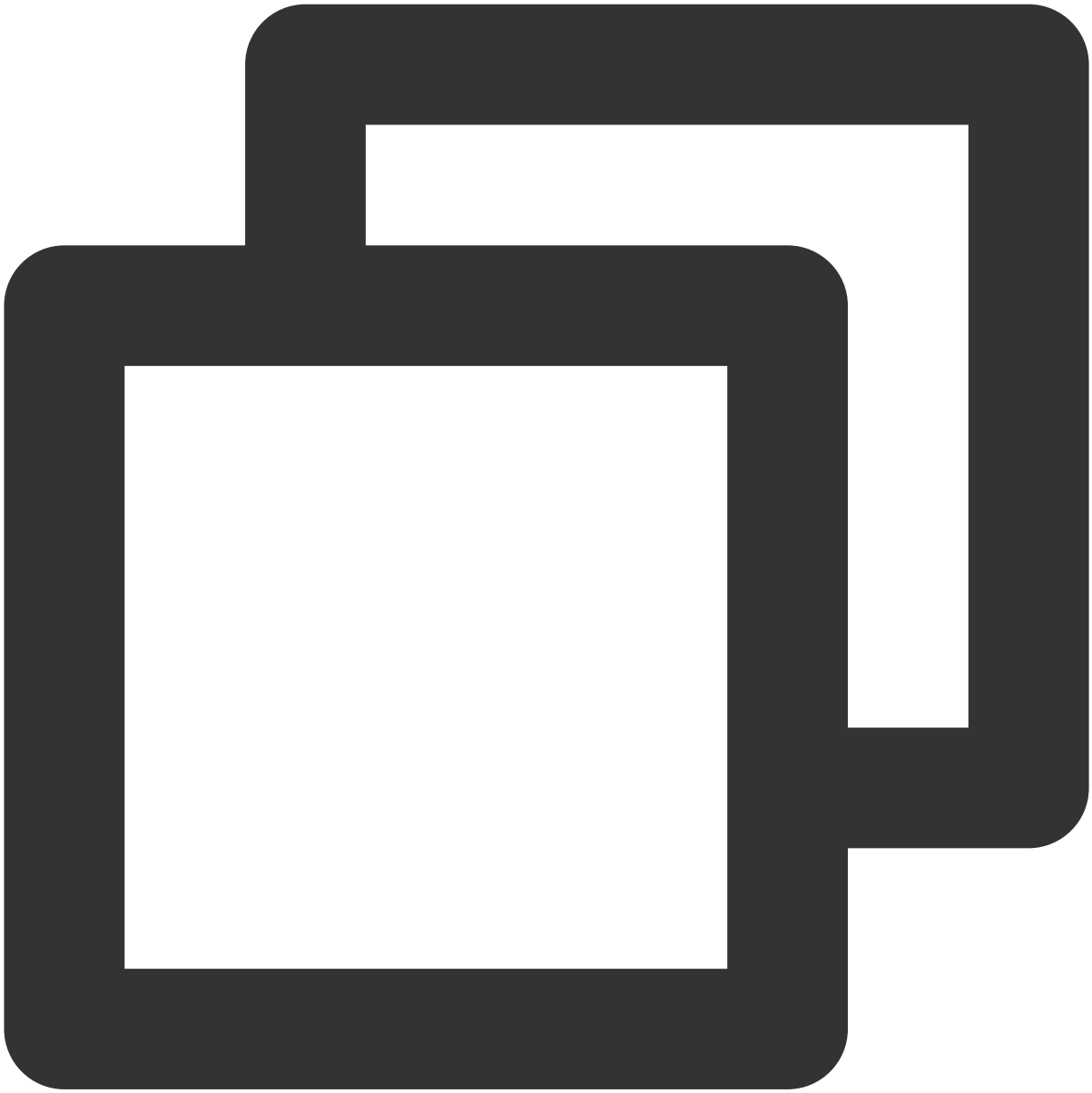
LogConfigの設定の参照は次のとおりです。



```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
spec:
   clsDetail:
     topicId: xxxxxx-xx-xx-xx-xxxxxxxx
     # JSON形式ログ
```

```
      logType: json_log
```

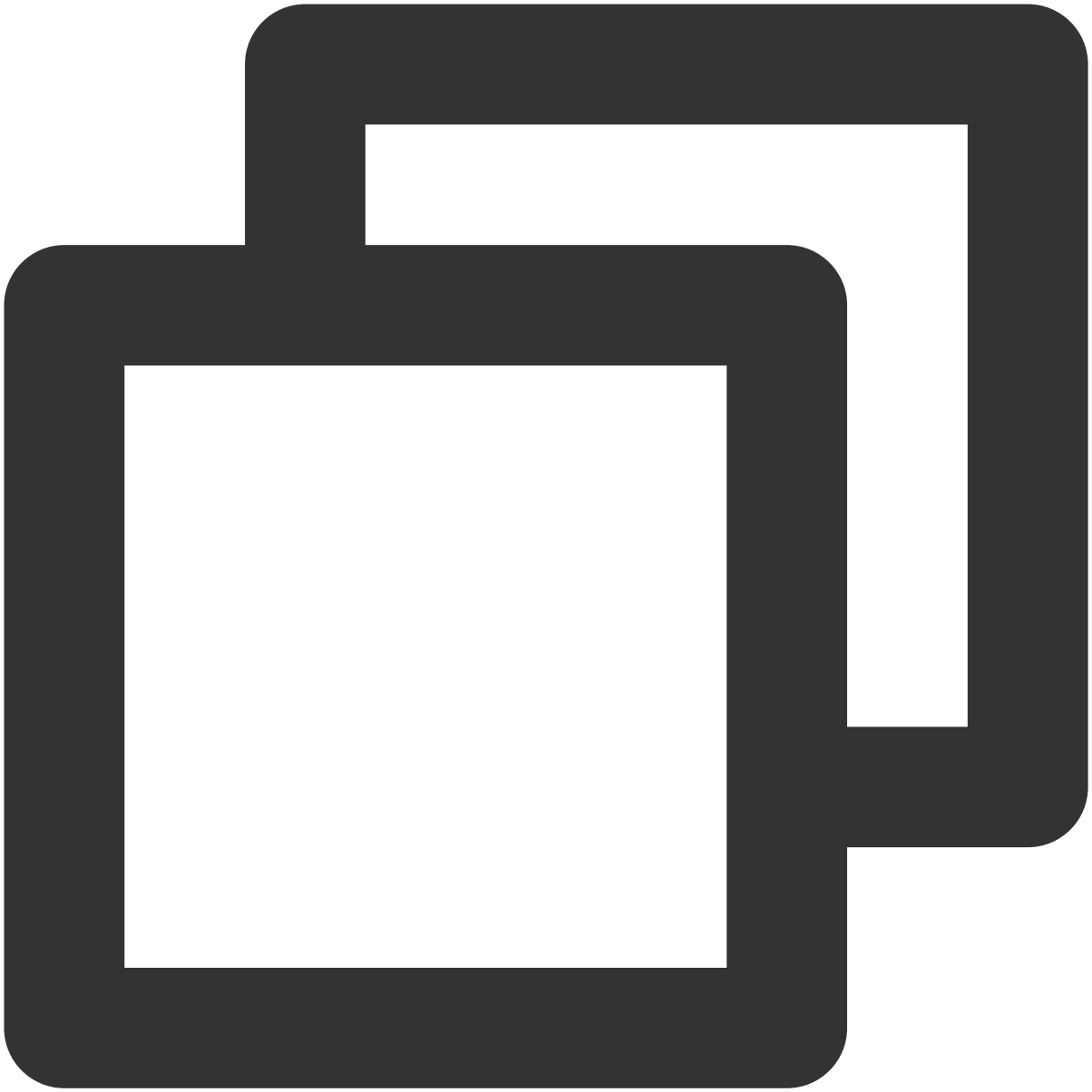
CLSにキャプチャされるデータは次のとおりです。



```
agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0
body_sent: 23
http_host: 127.0.0.1
method: POST
referer: http://127.0.0.1/my/course/4
remote_ip: 10.135.46.111
request: POST /event/dispatch HTTP/1.1
```

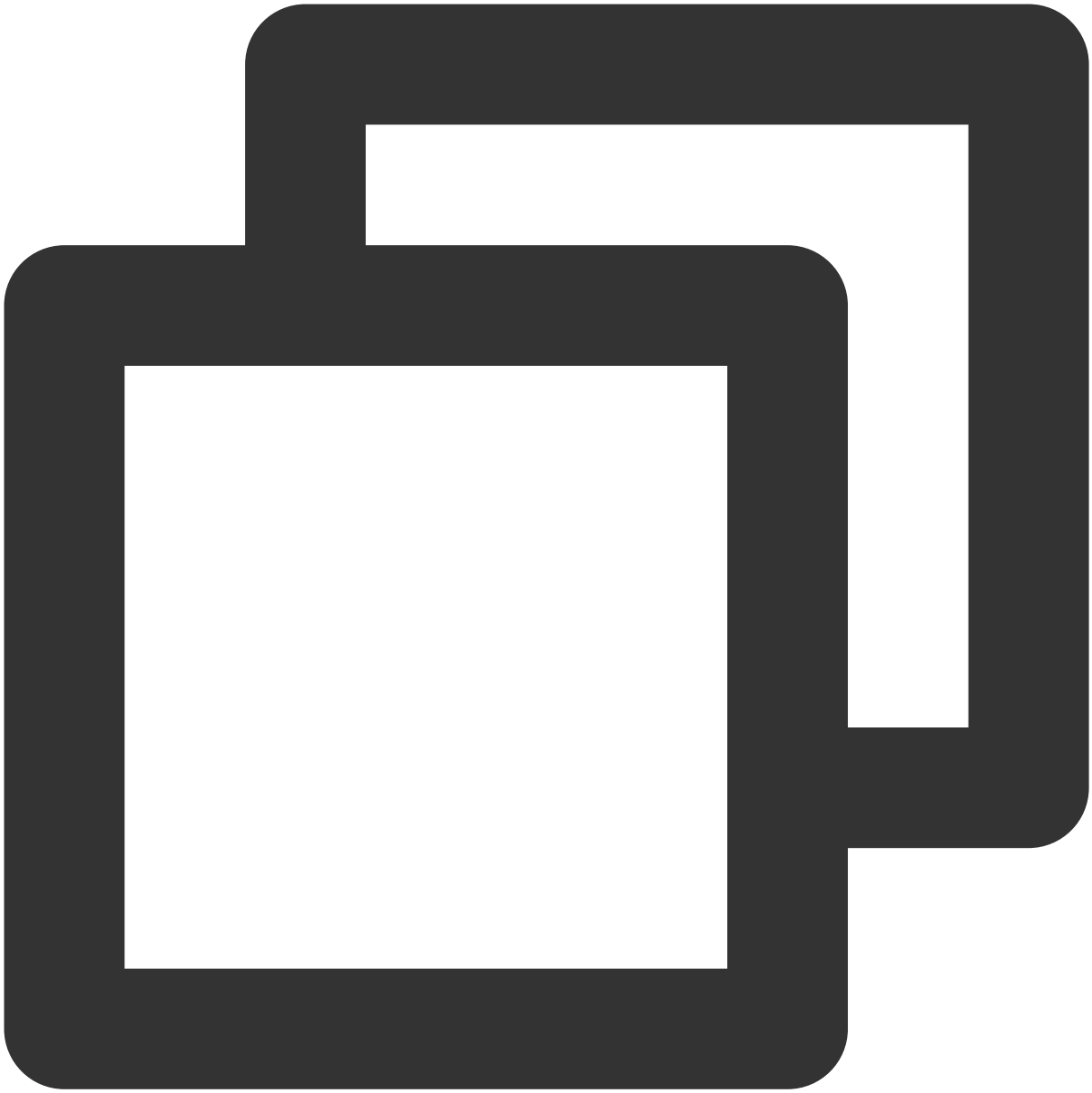
```
response_code: 200
responsetime: 0.232
time_local: 22/Jan/2019:19:19:34 +0800
upstreamhost: unix:/tmp/php-cgi.sock
upstreamtime: 0.232
url: /event/dispatch
xff: -
```

区切り文字ログとは、1行のログデータを指定された区切り文字に従ってログ全体の構造化処理を行い、それぞれの行の完全なログは、改行文字 `\\n` を終了識別子とするログのことです。CLSが区切り文字ログ処理を行う場合、各区切りのフィールドに固有のkeyを定義する必要があります。詳細については、区切り文字形式をご参照ください。
次のようなオリジナルログがあると仮定します。



```
10.20.20.10 ::: [Tue Jan 22 14:49:45 CST 2019 +0800] ::: GET /online/sample HTTP/1.
```

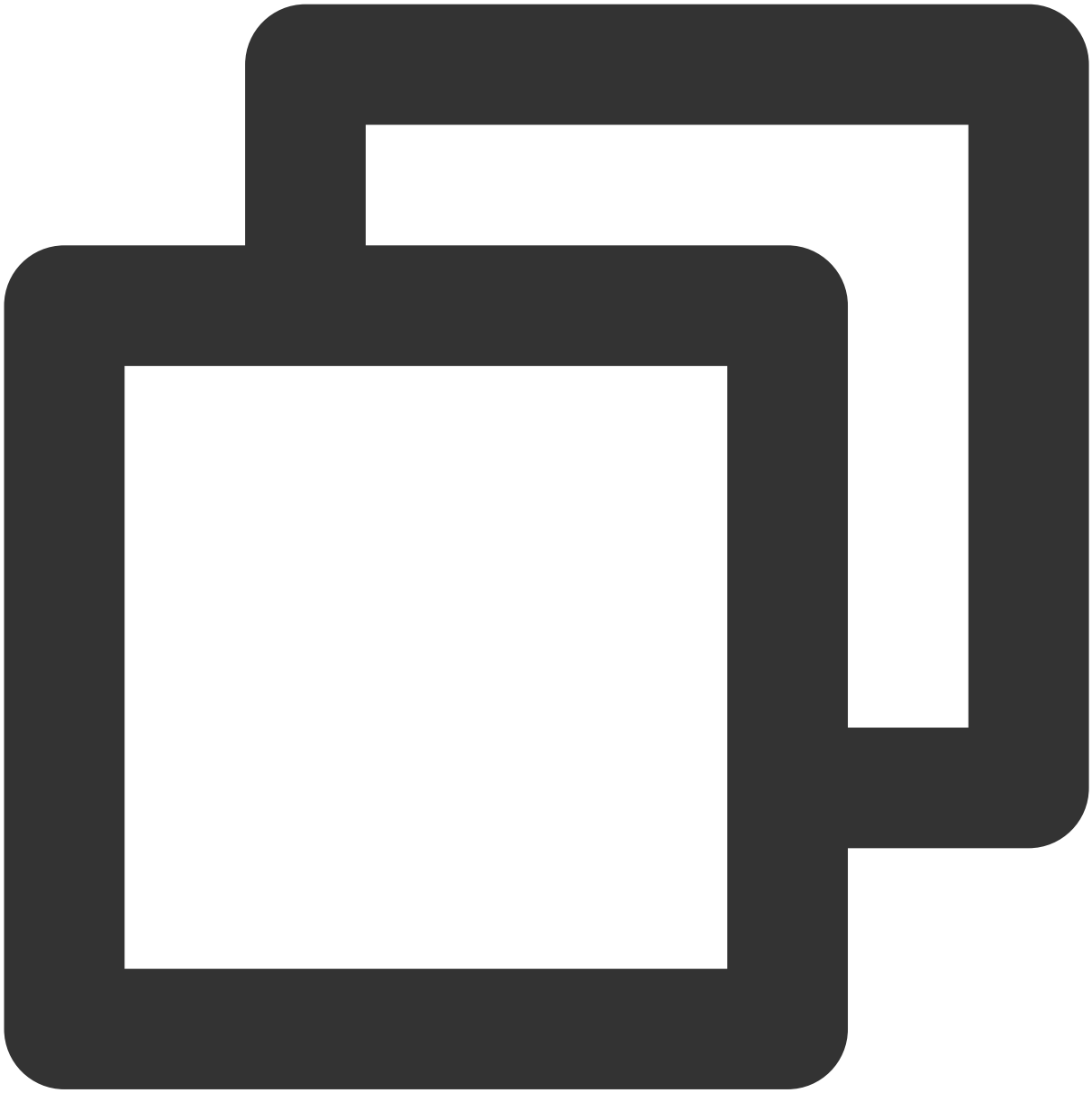
LogConfigの設定の参照は次のとおりです。



```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
spec:
   clsDetail:
     topicId: xxxxxx-xx-xx-xx-xxxxxxxx
     #区切り文字ログ
     logType: delimiter_log
     extractRule:
       #区切り文字
       delimiter: ':::'
       #抽出されたkeyリスト、分割されたフィールドと逐一対応します
```

```
        keys: ['IP','time','request','host','status','length','bytes','referer']
```

CLSにキャプチャされるデータは次のとおりです。



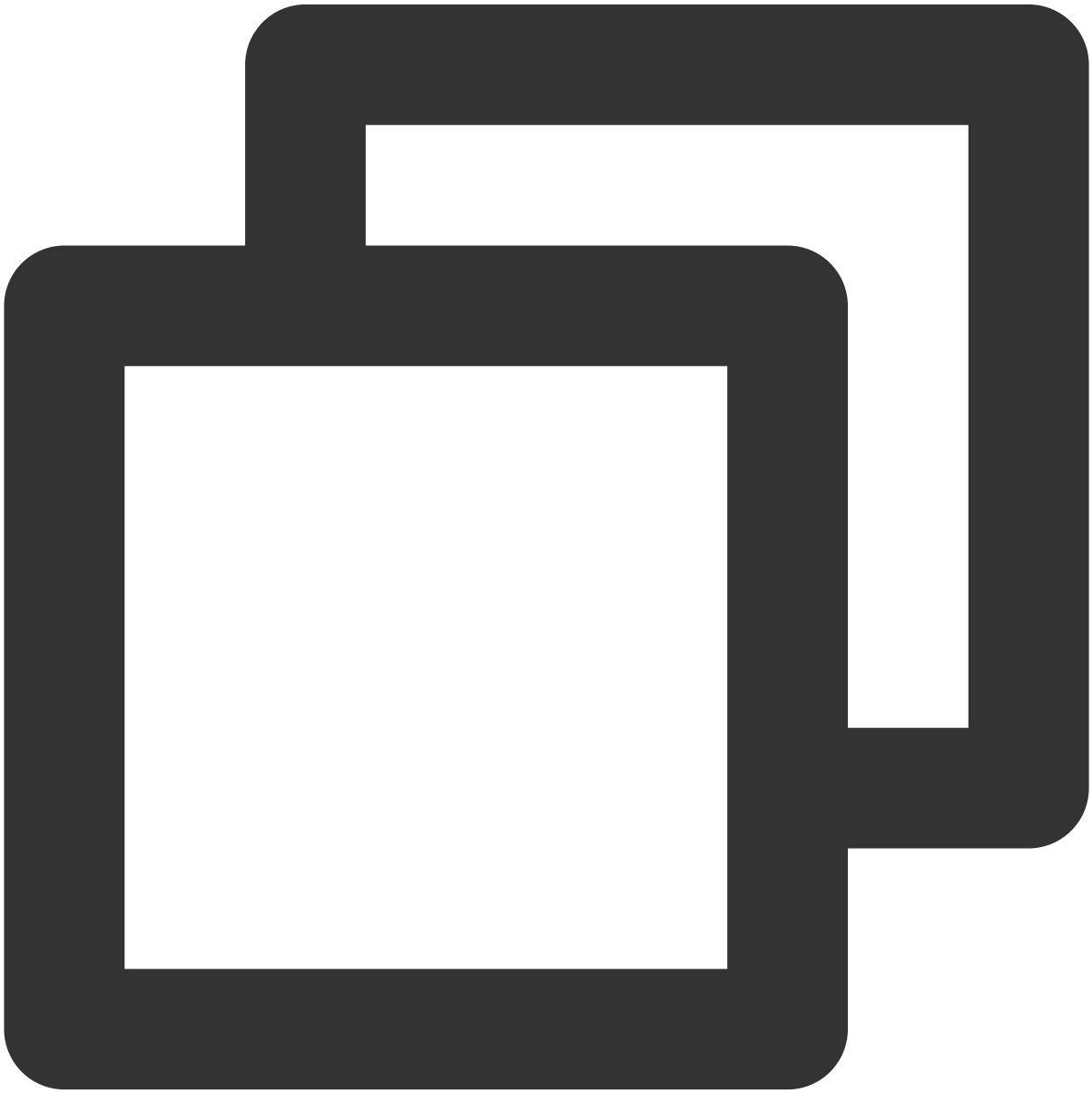
```
IP: 10.20.20.10
bytes: 35
host: 127.0.0.1
length: 647
referer: http://127.0.0.1/
request: GET /online/sample HTTP/1.1
status: 200
```

```
time: [Tue Jan 22 14:49:45 CST 2019 +0800]
```

キャプチャログのタイプ

コンテナの標準出力

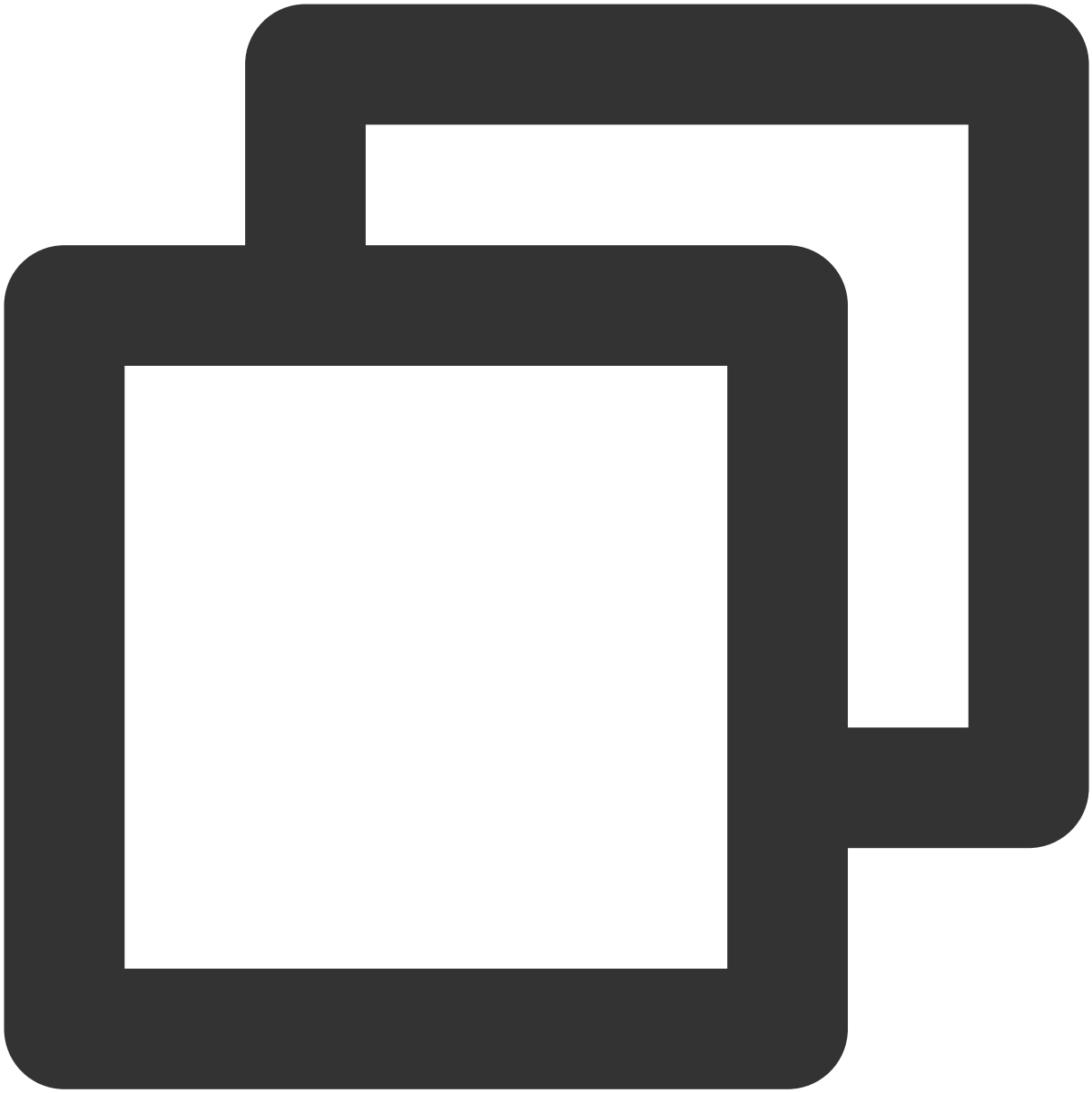
例1：defaultのネームスペースにあるすべてのコンテナの標準出力をキャプチャします



```
apiVersion: cls.cloud.tencent.com/v1
```

```
kind: LogConfig
spec:
   inputDetail:
     type: container_stdout
     containerStdout:
       namespace: default
       allContainers: true
 ...
```

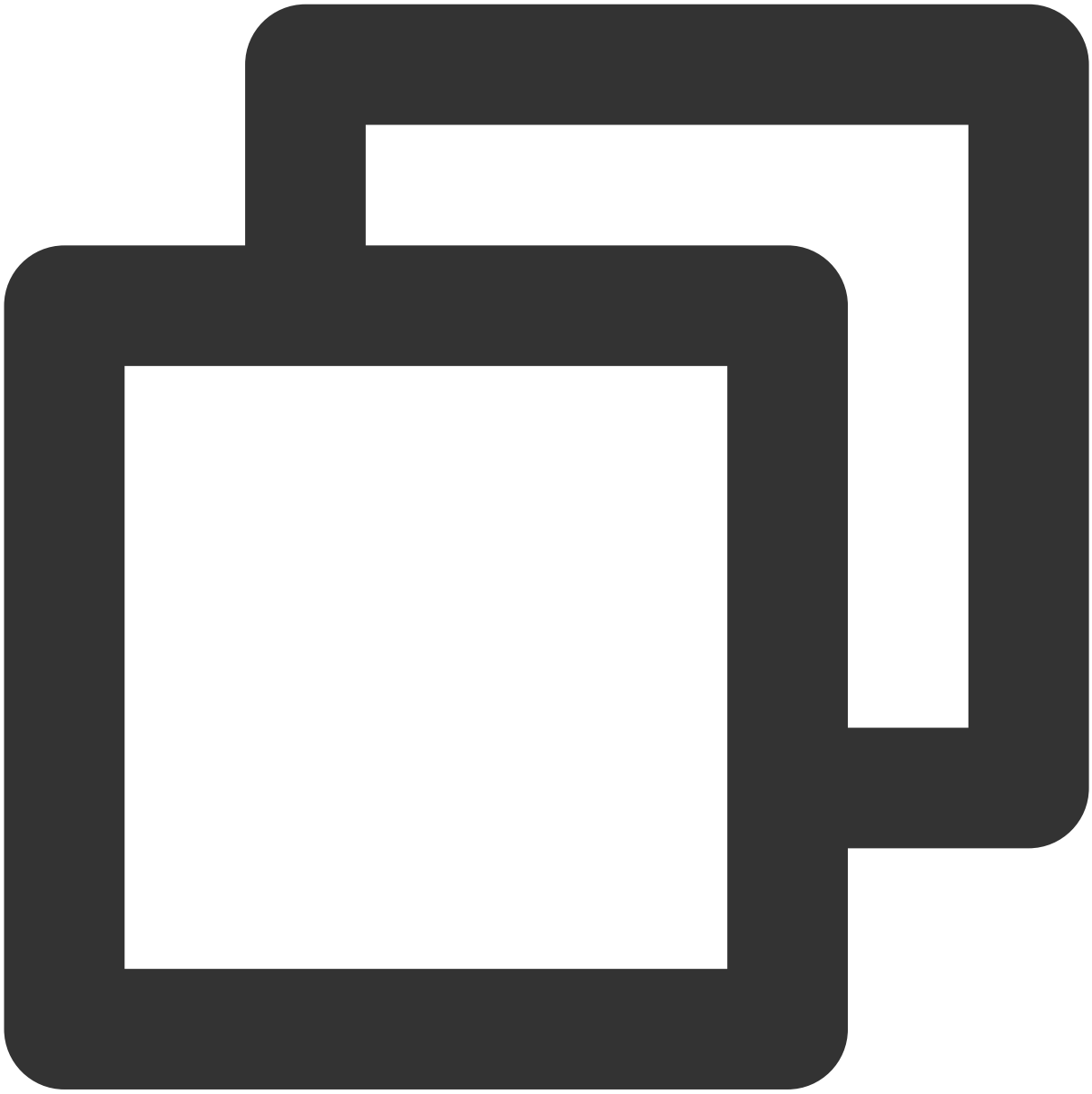
例2：productionのネームスペースのingress-gateway deploymentに属するpodのコンテナの標準出力をキャプチャします



```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
spec:
   inputDetail:
     type: container_stdout
     containerStdout:
       allContainers: false
       workloads:
       - namespace: production
         name: ingress-gateway
         kind: deployment
```

```
   ...
```

例3：productionのネームスペースにおけるpodタグに「k8s-app=nginx」を含むpodのコンテナの標準出力をキャプチャします

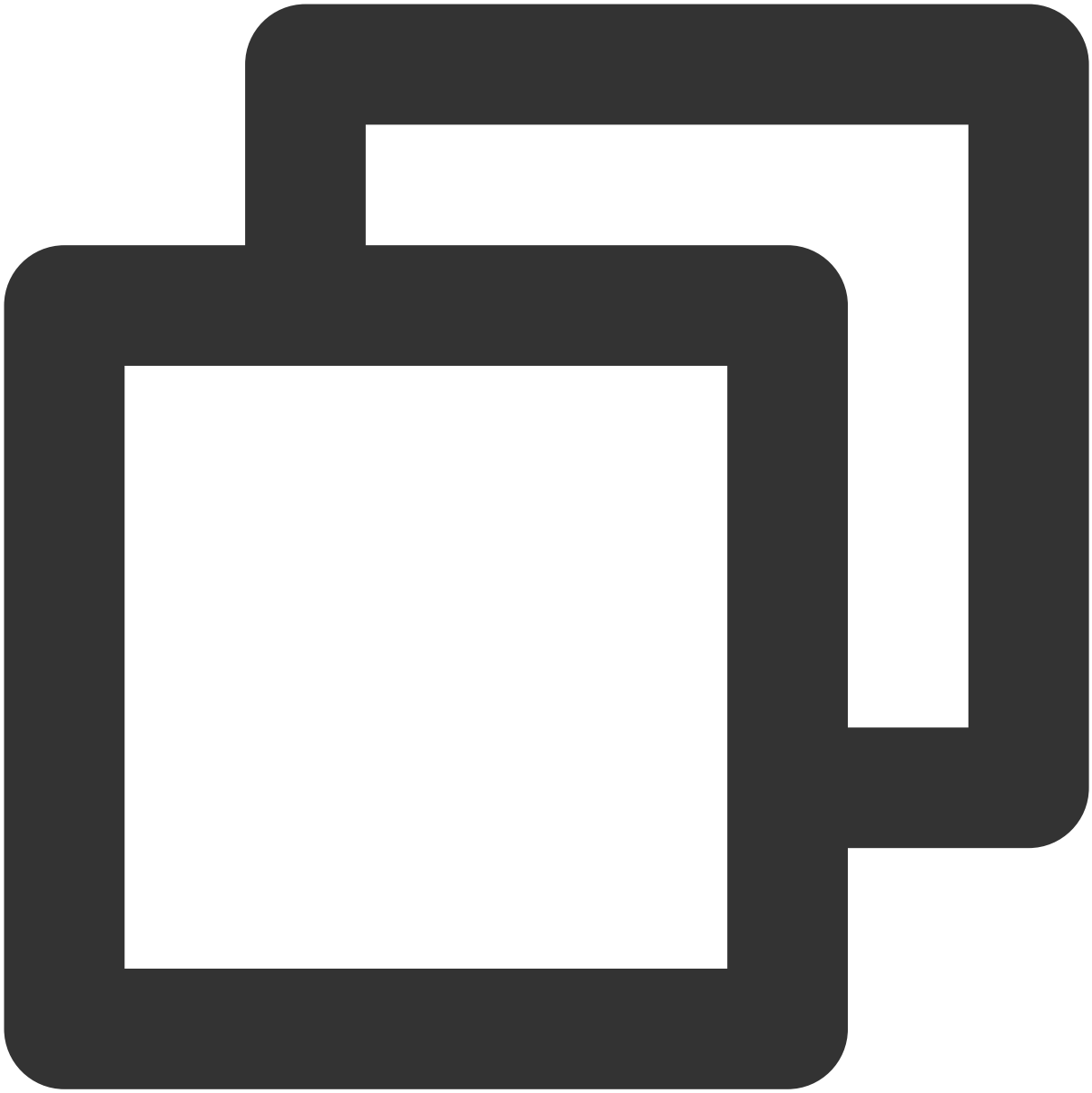


```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
spec:
   inputDetail:
     type: container_stdout
```

```
      containerStdout:
        namespace: production
        allContainers: false
        includeLabels:
          k8s-app: nginx
  ...
```

コンテナファイル

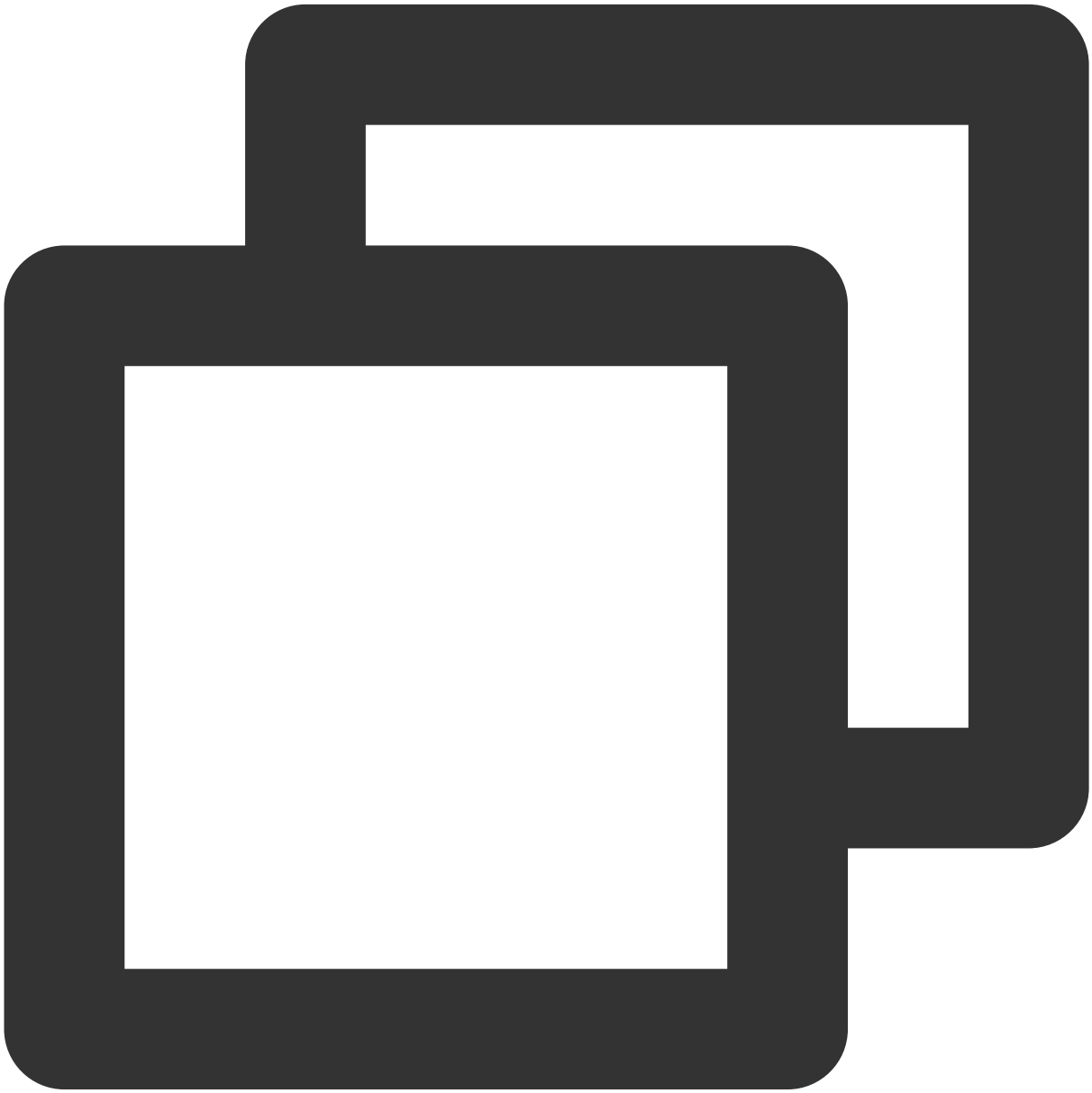
例1：productionネームスペースにおけるingress-gateway deploymentに属するpodのnginxコンテナの `/data/nginx/log/` パスの下にある `access.log` というファイルをキャプチャします



```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
spec:
   topicId: xxxxxx-xx-xx-xx-xxxxxxxx
   inputDetail:
     type: container_file
     containerFile:
       namespace: production
       workload:
         name: ingress-gateway
         type: deployment
```

```
        container: nginx
        logPath: /data/nginx/log
        filePattern: access.log
    ...
```

例2：productionネームスペースのpodタグに「k8s-app=ingress-gateway」が含まれるpodのnginxコンテナの `/data/nginx/log/` パスの下にある `access.log` というファイルをキャプチャします



```
apiVersion: cls.cloud.tencent.com/v1
kind: LogConfig
```

```
spec:
   inputDetail:
     type: container_file
     containerFile:
       namespace: production
       includeLabels:
         k8s-app: ingress-gateway
       container: nginx
       logPath: /data/nginx/log
       filePattern: access.log
   ...
```

メタデータ（Metadata）

コンテナ標準出力(container_stdout)およびコンテナファイル(container_file)は、オリジナルのログ内容のほか、コンテナシナリオに関するメタデータ（ログを生成したコンテナIDなど）とともにCLSへレポートする必要があります。これにより、ユーザーがログを確認する際、ソースを追跡したり、コンテナのマーカーや特性（コンテナ名やlabelsなど）に基づいて検索しやすくなります。
メタデータは下表のとおりです

フィールド名	意味
cluster_id	ログが属するクラスターIDです。
container_name	ログが属するコンテナ名です。
image_name	ログが属するコンテナのイメージ名IPです。
namespace	ログが属するpodのnamespaceです。
pod_uid	ログが属するpodのユーザーIDです。
pod_name	ログが属するpod名です。
pod_ip	ログが属するpodのIPです。
pod_lable_{label name}	ログが属するpodのlabelです（例えば1つのpodに、app=nginx、env=prodという2つのlabelがある場合、アップロードされたログにはpod_label_app:nginx、pod_label_env:prodという2つのmetedataが添付されます）。

審査の管理
クラスター審査

最終更新日：：2023-04-28 15:30:11

説明：
ログサービスCLSはTKE Serverlessクラスターによって生成されたすべての審査、イベントデータに**無料サービス**を2021年12月31日まで提供します。ログセットの自動作成を選択するか、または既存のログセット内でログトピックの自動作成を選択してください。

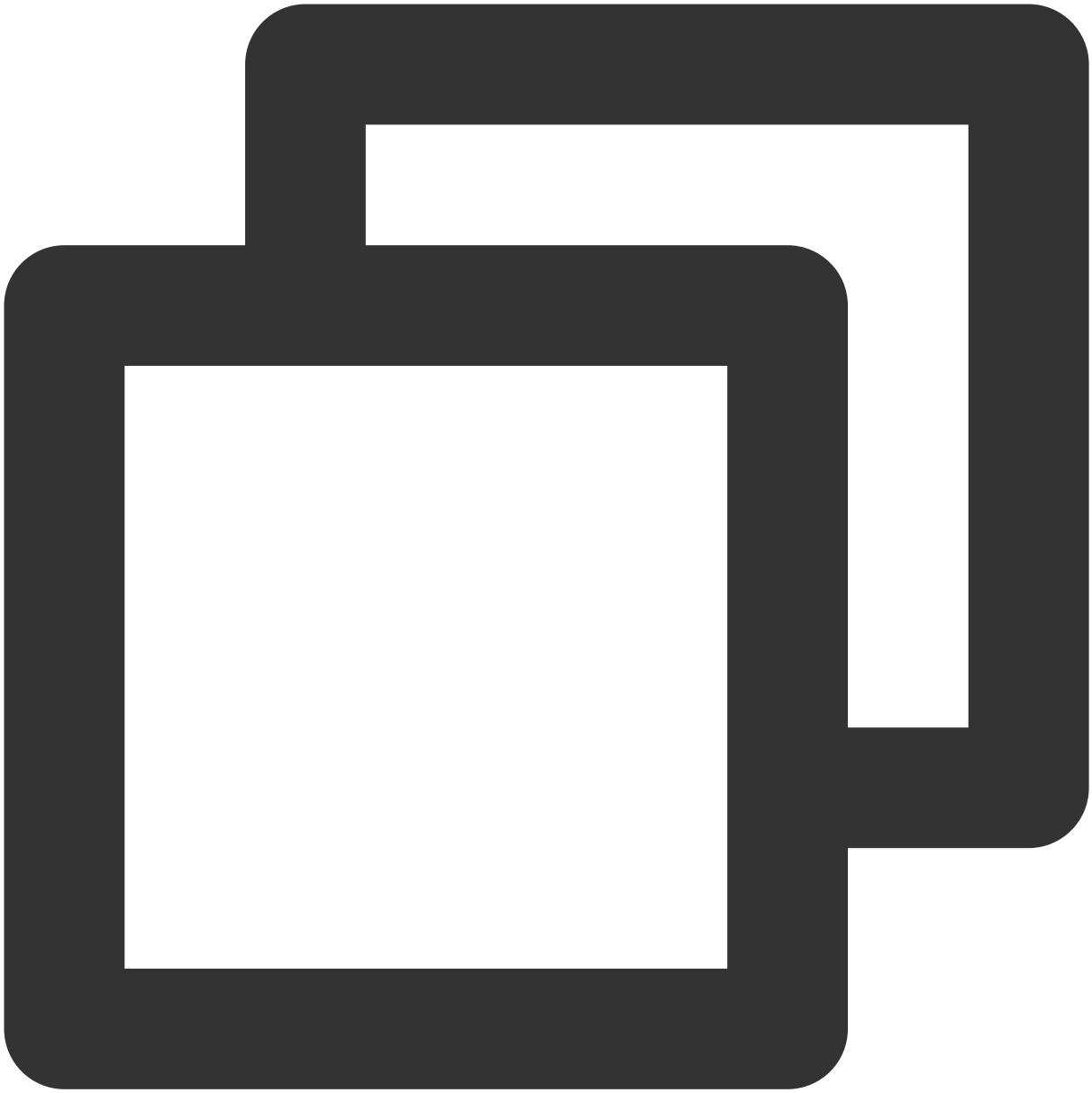
概要

クラスター審査はKubernetes Auditがkube-apiserverに対して生成したポリシーの設定が可能なJSON構造ログの記録保存および検索機能に基づきます。本機能はkube-apiserverに対するアクセスイベントを記録し、順番にユーザー、管理者またはシステムコンポーネントがクラスターに影響を与えるイベントを記録します。

機能のメリット

クラスター審査機能はmetricsとは異なる別のクラスター監視ディメンションを提供します。クラスター審査を有効化すると、Kubernetesは毎回クラスター操作に対する審査ログを記録することができます。各審査ログはJSON形式の構造化レコードで、メタデータ（metadata）、リクエスト内容（requestObject）および応答内容（responseObject）の3つの部分が含まれます。このうち、メタデータ（誰が開始したリクエストであるか、どこから開始されたか、アクセスのURIなどのリクエストのコンテキスト情報が含まれます）は必ず存在し、リクエストおよび応答内容が存在するかどうかは審査レベルで決まります。ログによって以下の内容を理解することができます。
クラスター内で発生したイベント。
イベントの発生時間および発生オブジェクト。
イベントのトリガー時間、トリガー位置および観察点。
イベントの結果および後続の処理行動。

審査ログ事例の閲覧



```
{
  "kind":"Event",
  "apiVersion":"audit.k8s.io/v1",
  "level":"RequestResponse",
  "auditID":0a4376d5-307a-4e16-a049-24e017******,
  "stage":"ResponseComplete",
  // 何が起きたか
  "requestURI":"/apis/apps/v1/namespaces/default/deployments",
  "verb":"create",
  // 誰が開始したか
  "user":{
```

```
    "username":"admin",
      "uid":"admin",
      "groups":[
        "system:masters",
        "system:authenticated"
      ]
  },
  // どこで開始されたか
  "sourceIPs":[
    "10.0.6.68"
  ],
  "userAgent":"kubectl/v1.16.3 (linux/amd64) kubernetes/ald64d8",
  // 何が起きたか
  "objectRef":{
    "resource":"deployments",
    "namespace":"default",
    "name":"nginx-deployment",
    "apiGroup":"apps",
    "apiVersion":"v1"
  },
  // どのような結果か
  "responseStatus":{
    "metadata":{
    },
    "code":201
  },
  // 具体的な情報をリクエストして返す
  "requestObject":Object{...},
  "responseObject":Object{...},
  // いつ開始/終了したか
  "requestReceivedTimestamp":"2020-04-10T10:47:34.315746Z",
  "stageTimestamp":"2020-04-10T10:47:34.328942Z",
  // リクエストが受信/拒絶された原因は何か
  "annotations":{
    "authorization.k8s.io/decision":"allow",
    "authorization.k8s.io/reason":""
  }
}
```

TKE Serverlessクラスターの審査ポリシー

審査レベル（level）

一般的なログと異なり、Kubernetes審査ログのレベルはよりverbose設定に近く、記録情報の詳細度を示すために使用されます。計4つのレベルがあります。以下の表の内容をご参照ください。

パラメータ	説明
None	記録しません。
Metadata	リクエストのメタデータ（ユーザー、時間、リソース、操作など）を記録します。リクエストおよび応答のメッセージボディは含まれません。
Request	メタデータ以外に、リクエストメッセージボディが含まれます。応答のメッセージボディは含まれません。
RequestResponse	すべての情報を記録します。メタデータおよびリクエスト、応答のメッセージボディが含まれます。

審査段階（stage）

ログの記録は異なる段階で発生します。以下の表の内容をご参照ください：

パラメータ	説明
RequestReceived	リクエストを受信すると記録する。
ResponseStarted	watchのような長時間接続リクエストのみ、リターンメッセージヘッダの送信が完了した後に記録します。
ResponseComplete	リターンメッセージが全て送信完了した後に記録します。
Panic	内部サーバーにエラーが発生し、リクエストが完了していません。

監査ポリシー

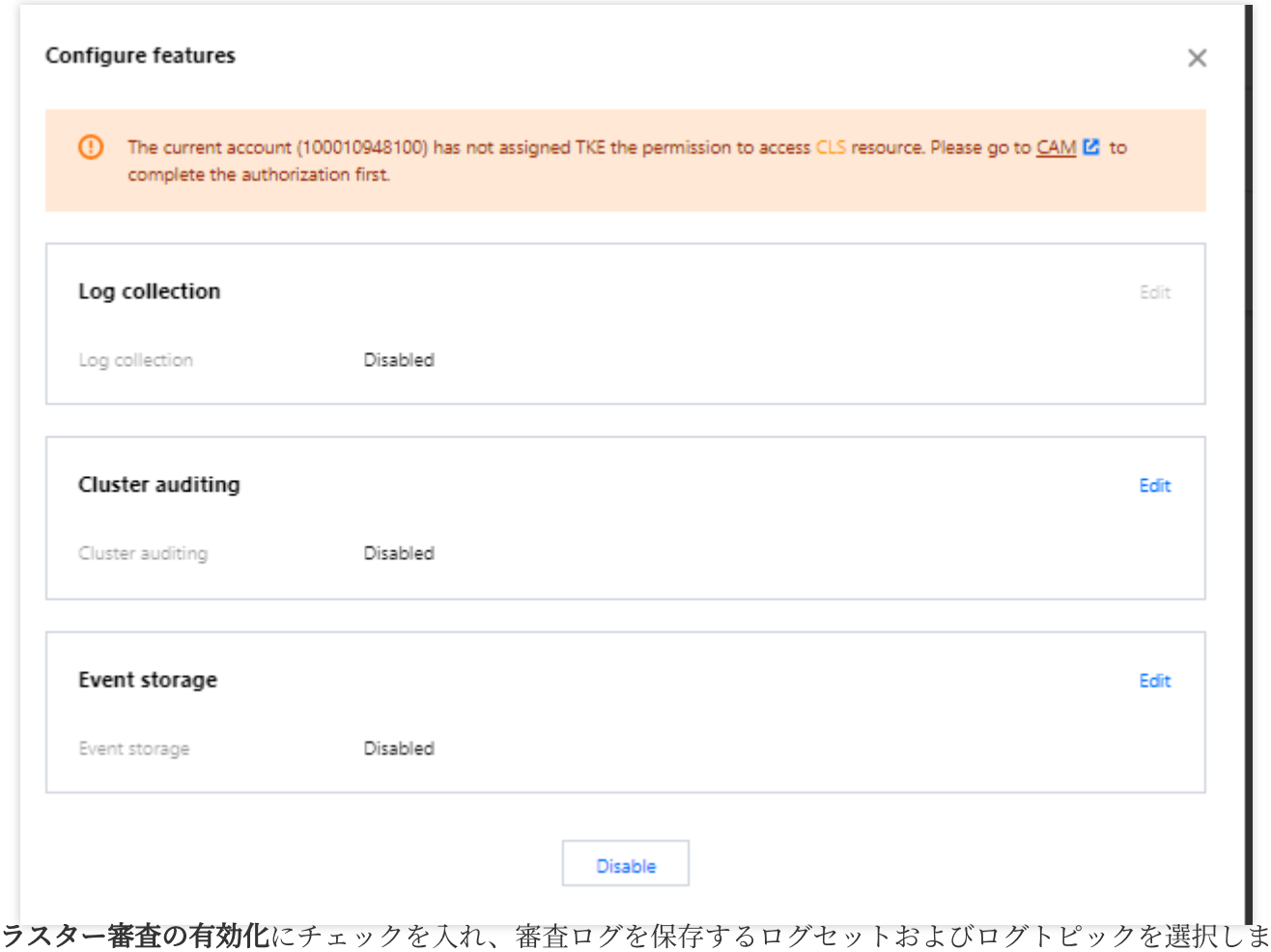
TKE Serverlessはデフォルトでリクエストを受信して審査ログを記録し、且つ大部分の操作はRequestResponseレベルの審査ログに記録されます。ただし以下のような状況が存在する場合があります。
get、listおよびwatchはRequestレベルのログを記録します。
secretsリソース、configmapsリソースまたはtokenreviewsリソースに対するリクエストはMetadataレベルで記録されます。
以下のリクエストはログの記録を行いません。
`system:kube-proxy` が送信したendpointsリソース、servicesリソースまたはservices/statusリソースを監視するリクエスト。
`system:unsecured` が送信した kube-systemネームスペース内の configmapsリソースに対するgetリクエスト。
kubeletが送信したnodesリソースまたはnodes/statusリソースに対するgetリクエスト。

`system:kube-controller-manager` 、 `system:kube-scheduler` または `system:serviceaccount:endpoint-controller` が送信したkube-system ネームスペース内のendpointsリソースに対するgetおよびupdate リクエスト。
`system:apiserver` が送信した namespaces リソース、namespaces/statusリソースまたはnamespaces/finalizeリソースに対するgetリクエスト。
`/healthz*` 、 `/version` または `/swagger*` にマッチングするURLに対して送信されるリクエスト。

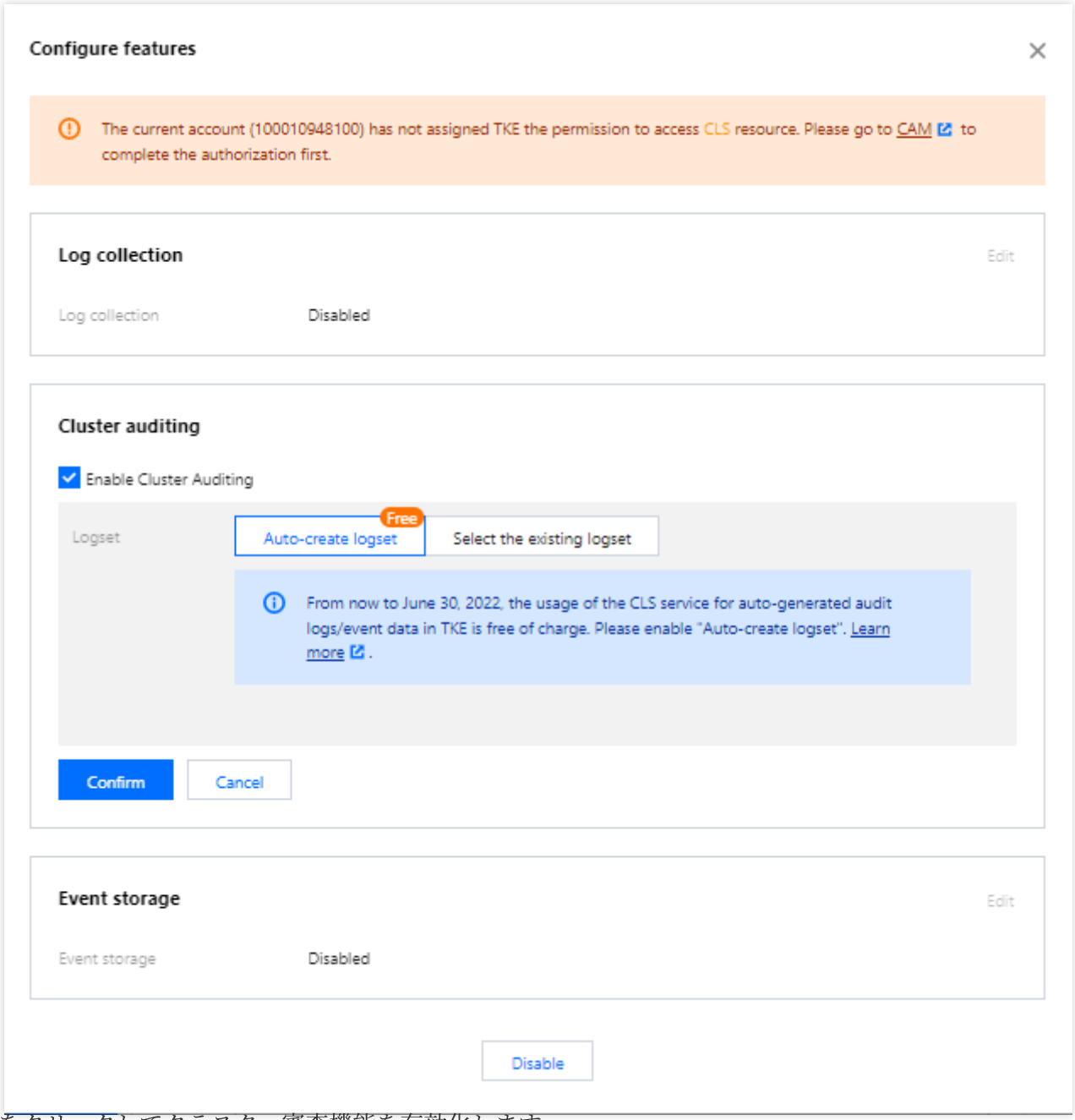
操作手順

クラスター審査を有効化する

注意
クラスター審査の機能を有効化するにはkube-apiserverを再起動する必要があります。頻繁にオンオフしないことをお勧めします。
1. TKEコンソールにログインします。
2. 左側ナビゲーションバーの**運用保守機能管理**を選択し、「機能管理」ページに進みます。
3. 「機能管理」ページ上方でリージョンを選択し、「Serverlessクラスター」タイプを選択します。
4. 下方のクラスターリストでクラスター審査を有効化したいクラスターを見つけ、右側操作バーで**設定**をクリックします。
5. ポップアップした「機能設定」ウィンドウで、「クラスター審査」機能右側の**編集**をクリックします。下図に示すとおりです。



6. **クラスター審査の有効化**にチェックを入れ、審査ログを保存するログセットおよびログトピックを選択します。**ログセットの自動作成**を選択することを推奨します。下図に示すとおりです。



7. **OK**をクリックしてクラスター審査機能を有効化します。