sysdig

Prometheus

モニタリングガイド

- Kubernetesを監視する方法 -

Prometheusは、最も人気のあるDockerおよびKubernetes監視ツールの1つになりつつあります。 この電子書籍を読んで、Prometheusの利点、およびそれを使用してKubernetesのサービス、クラスター、およびコンポーネントを監視する方法について学習します。

Prometheusについて

Prometheusは、もともとSoundCloudで構築されたオープンソースのシステム監視およびアラートツールキットであり、DockerおよびKubernetesの最も人気のある監視ツールの1つになりつつあります。

Promethusの台頭は、新しいタイプの監視フレームワークの必要性を生み出した2つの技術シフトに起因する可能性があります。

1. DevOps文化：DevOpsが登場する前は、監視はホスト、ネットワーク、およびサービスで構成されていました。DevOpsの世界では、最先端の開発者は多くの場合CI/CDパイプラインを非常に使いこなしており、自分自身がデバッグを行います。そのため、インフラストラクチャの有機的な部分として、アプリとビジネス関連の両方のメトリクスを簡単に統合できる必要があります。監視は、民主化され、アクセスしやすくなり、スタックの追加レイヤーをカバーするように拡張される必要がありました。
2. コンテナとKubernetes：コンテナベースのインフラストラクチャは、ロギング、デバッグ、高可用性、監視を根本的に変えました。今日のITチームは、膨大な数の揮発的なソフトウェアエンティティ、サービス、仮想ネットワークアドレス、公開されたメトリクスに直面しています。

主な機能

Prometheusをコンテナ化された環境の監視に使用するツールとして位置付けるために、いくつかの機能が統合されています。

1. 多次元データモデル：柔軟で精密な時系列データは、Prometheusクエリ言語を強化します。このモデルは、Kubernetesがラベルを使用してインフラストラクチャメタデータを整理する方法と同様に、キーと値のペアに基づいています。
2. アクセス可能なフォーマットとプロトコル：Prometheusメトリクスの公開は簡単なタスクです。メトリクスは人間が読める形式であり、わかりやすい形式であり、標準のHTTPトランスポートを使用して公開されます。
3. サービスディスカバリー：Prometheusサーバーは定期的にターゲットをスクレイピングするためにアプリケーションやサービスが行う必要がありません。つまり、メトリクスはプッシュされるのではなく、プルされます。 Prometheusサーバーは、いくつかの方法の1つを使用して

ターゲットを自動検出できます。また、一部はコンテナメタデータをフィルターおよび一致するように構成できます。これは、エフェメラルなKubernetesワークロードに最適です。

4. モジュラーおよび高可用性コンポーネント：メトリクスの収集、アラート、グラフィカルな視覚化、およびその他の同様の機能を実行する構成可能なサービスは、冗長性とシャーディングをサポートするように設計されています。

Prometheusは監視の課題をどのように克服するか

プロメテウスは、Kubernetesクラスターの監視が抱える多くの固有の課題を克服するのに役立ちます。

1. **コンテナの可視性**

   コンテナは軽量で、ほとんどが不変のブラックボックスであるため、検索や監視が困難になります。

   Kubernetes APIとkube-state-metrics（Prometheusメトリクスをネイティブに使用）は、Kubernetes内部データ（デプロイメント内の必要な/実行中のレプリカの数、スケジュールできないノードなど）を公開することでこれを解決するのに役立ちますが、Prometheusはより簡単にに行います。

   Prometheusを使用すると、メトリクスポートを公開するだけで済み、複雑さを追加したり、追加のサービスを実行したりすることはありません。多くの場合、サービス自体は既にHTTPインターフェースを提供しており、必要なのは/ metricsなどの追加のパスだけです。

   サービスがPrometheusメトリクスを提供する準備ができておらず、それをサポートするためにコードを変更できない場合は、サービスにバンドルされているPrometheusエクスポーターを、同じポッドのサイドカーコンテナーとしてデプロイするだけです。

2. 変化する揮発的なインフラ

いつでもレポートを開始または停止できる一時的なエンティティは、従来の静的な監視システムの問題です。

Prometheusは、3つの便利な自動検出メカニズムを備えた動的監視ツールです。

- Consul：サービスの検出と構成のためのツール。 Consulは分散され、可用性が高く、非常にスケーラブルです。
- Kubernetes：Kubernetes SD構成では、KubernetesのREST APIからスクレイプターゲットを取得し、常にクラスター状態と同期を維持できます。
- Prometheus Operator：使い慣れたKubernetesラベルクエリに基づいて、監視対象の構成を自動的に生成します。

3. 新しいインフラストラクチャ層

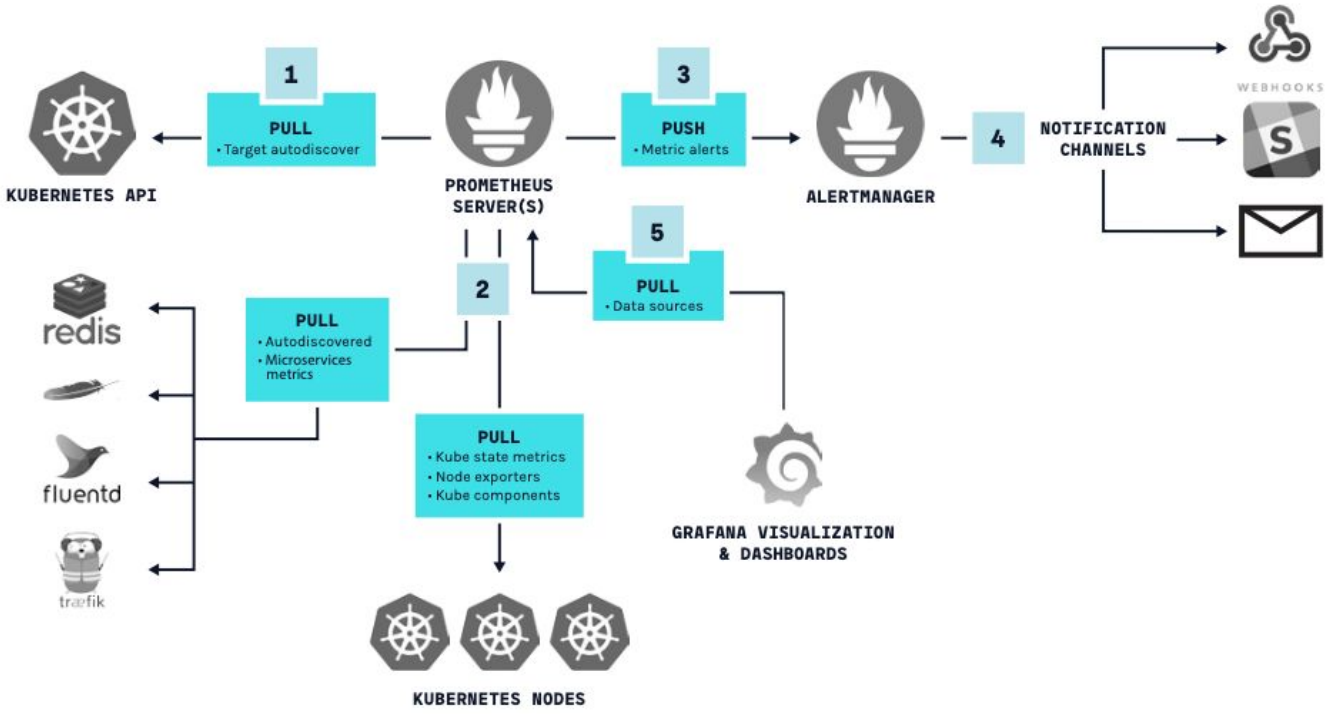
Kubernetesコンポーネントは新しいインフラストラクチャレイヤーを作成し、マイクロサービスパフォーマンス（複数のノードに散在する異なるポッド）、ネームスペース、デプロイメントバージョンなどのグループ化に基づいて監視を編成する必要があります。

PromtheusのラベルベースのデータモデルとPromQLを使用すると、これらの新しいスコープに簡単に適応できます。

PrometheusでKubernetesを監視する方法

Kubernetesクラスタアーキテクチャの概要

以下の図は、Kubernetesクラスター内のコアコンポーネントとそれぞれの機能を示しています。



1. **Kubernetes API**

   Prometheusによるターゲットの自動検出は、いくつかの異なる方法で実現できます。

   - Consul
   - Prometheus Kubernetes SDプラグイン
   - Prometheusオペレーターとそのカスタムリソース定義

2. **Kubernetesノード**

Prometheusは、Kubernetesサービス、ノード、オーケストレーションステータスに関連するメトリクスと同様に、アプリケーションメトリクスを収集する必要があります。

- ノードエクスポーター、従来のホスト関連のメトリクス：CPU、mem、ネットワークなど
- オーケストレーションおよびクラスターレベルのメトリクス用のKube-state-メトリクス：デプロイメント、ポッドメトリクス、リソース予約など
- 内部コンポーネントからのKube-systemメトリクス：kubelet、etcd、DNS、スケジューラなど

3. **アラート**

   Prometheusは、PromQLを使用してアラートをトリガーするルールを構成できます。alertmanagerは、アラート通知、グループ化、禁止などの管理を担当します。

4. **通知デバイス**

   alertmanagerコンポーネントは、アラート通知を配信するようにレシーバー、ゲートウェイを構成します。

5. **ダッシュボード**

   分析とレポートのオープンプラットフォームであるGrafanaは、任意の数のPrometheusサーバーとディスプレイパネルからメトリクスを取得できます。

PrometheusでKubernetesを監視する

Prometheusモニタリングを使用してKubernetesをデプロイするには、多くのコンポーネントを用います。ここでは、Prometheusを使用してエンドユーザーアプリとKubernetesクラスターエンドポイントを監視するためのコア要素について説明します。

Kubernetesサービス

サービスには2つのタイプがあります。Prometheusメトリクスを公開する機能をネイティブに持っているか、Prometheusエンドポイントを提供しているサービスとそうでないサービスです。エクスポー

ターは、サービス統計を収集し、プロメテウスにスクレイピングする準備ができているメトリクスを変換するサービスです。

1. Prometheusメトリクスを公開できるサービスの監視

Traefikは、マイクロサービスおよびコンテナと緊密に統合されるように設計されたリバースプロキシです。 Traefikの一般的な使用例は、イングレスコントローラーまたはエントリポイントとして使用されることです。これは、インターネットとクラスター内の特定のマイクロサービスとの間のブリッジです。

Traefikをインストールするにはいくつかのオプションがあります。また、Kubernetes用のインストールガイドもあります。 Prometheusのサポートを備えた簡単なTraefikのデプロイ麺とをすぐに実行したい場合は、以下のスニペットを使用します。

```
kubectl create -f https://raw.githubusercontent.com/mateobur
prometheus-monitoring-guide/master/traefik-prom.yaml
```

Traefikポッドが実行されたら、サービスIPを表示できます。

```
kubectl get svc

NAME                         TYPE        CLUSTER-IP     EXTERNAL-IP  PORT(S)                  AGE
kubernetes                   ClusterIP   10.96.0.1      <none>       443/TCP                  238d
prometheus-example-service   ClusterIP   10.103.108.86  <none>       9090/TCP                 5h
traefik                      ClusterIP   10.108.71.155  <none>       80/TCP,443/TCP,8080/TCP  35s
```

curlを使用して、Prometheusメトリクスが公開されていることを確認できます。

```
curl 10.108.71.155:8080/metrics
# HELP go_gc_duration_seconds A summary of the GC invocation durations.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"}                              2.4895e-05
go_gc_duration_seconds{quantile="0.25"} 4.4988e-05
...
```

次に、新しいターゲットをprometheus.yml confファイルに追加する必要があります。 Prometheusが自動的にスクレイピングすることに気付くでしょう：

```
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ['localhost:9090']
```

別の静的エンドポイントを追加しましょう。

```
  - job_name: 'traefik'
    static_configs:
    - targets: ['traefik:8080']
```

サービスが別のネームスペースにある場合は、FQDN（traefik.de- fault.svc.cluster.local）を使用する必要があります。もちろん、これは最低限の構成であり、scrape構成は次のような複数のパラメーターをサポートします。

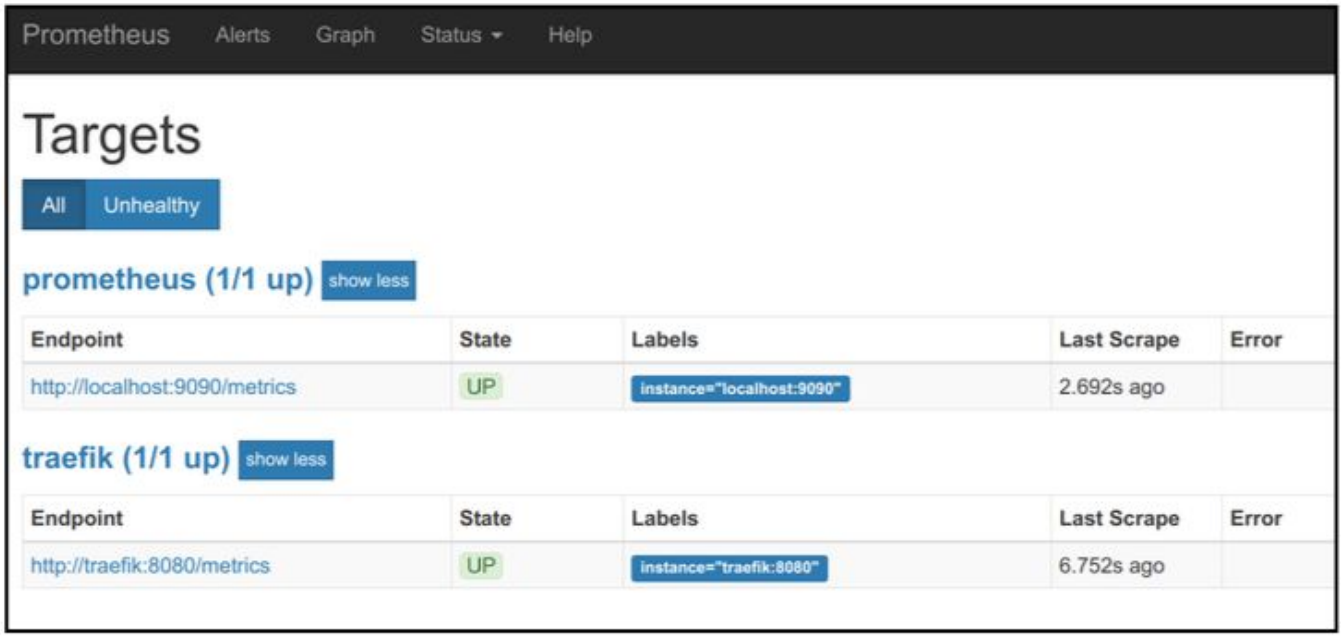
- basic_authおよびbearer_token：エンドポイントでは、リクエストヘッダーで従来のログイン/パスワードスキームまたはベアラートークンを使用して、HTTPS経由の認証が必要になる場合があります。

- kubernetes_sd_configsまたはconsul_sd_configs：さまざまなエンドポイント自動検出メソッド。
- scrape_interval、scrape_limit、scrape_timeout：精度、復元力、システム負荷のトレードオフが異なります。

ConfigMapとデプロイメントにパッチを適用します。

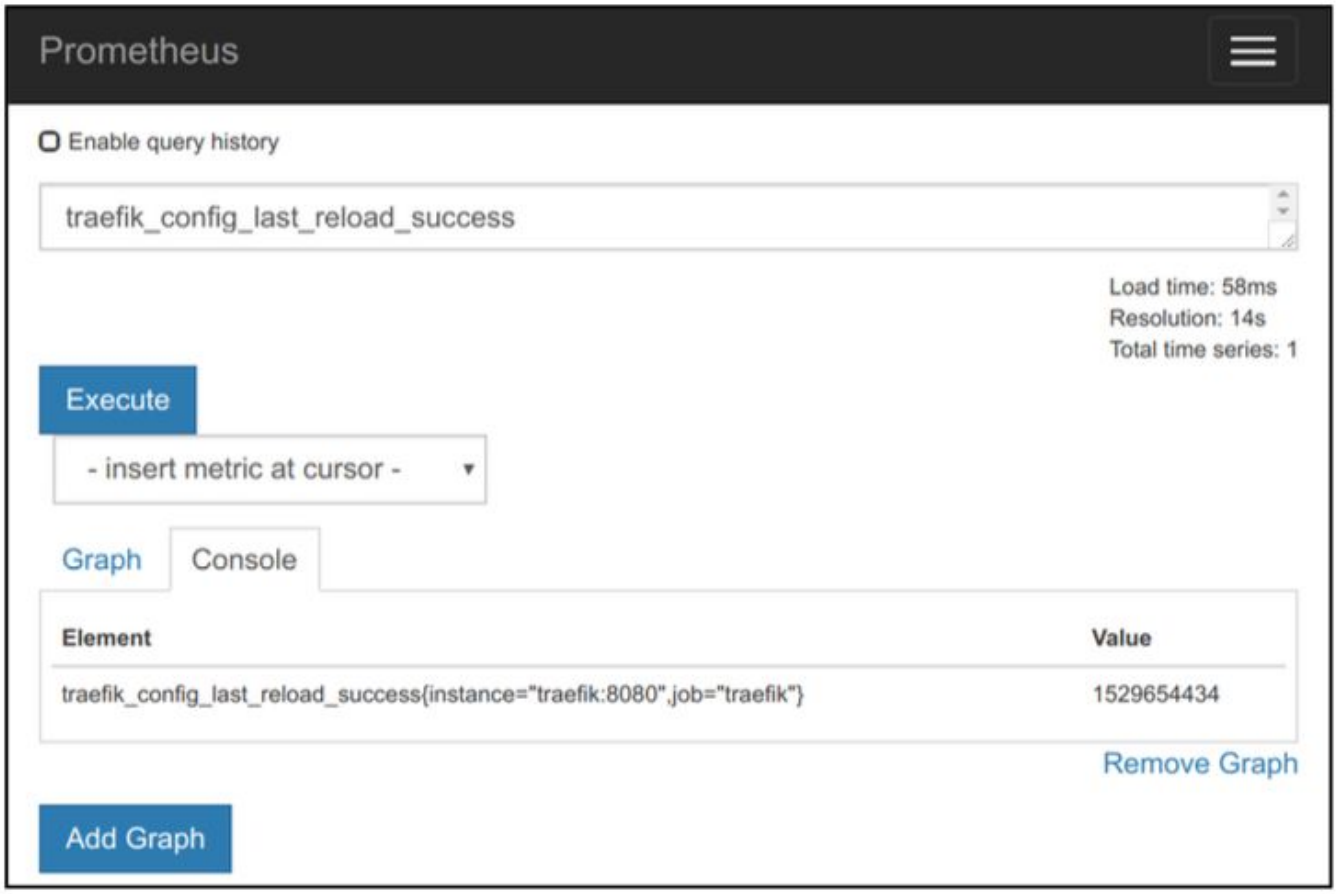
```
kubectl create configmap prometheus-example-cm --from-file=prometheus.
yml -o yaml --dry-run | kubectl apply -f -

kubectl patch deployment prometheus-deployment -p \
   "{\"spec\":{\"template\":{\"metadata\":{\"labels\":{\"date\":\"`date
   +'%s'`\"}}}}}"
```

Prometheus Webインターフェイスで/targets URLにアクセスすると、Traefikエンドポイントが表示されます。



メインのWebインターフェースを使用して、traefikのメトリクスをいくつか見つけて（ごく少数です。この例ではTraefikフロントエンドまたはバックエンドが設定されていないため）、その値を取得

できます。



2. Prometheusエクスポーターを使用したサービスの監視

アプリケーションがネイティブにPrometheusメトリクスを公開していない場合、メインサービスの状態/ログ/その他のメトリクス形式を取得してこの情報をPrometheusメトリクスとして公開するには、Prometheusエクスポーターをバンドルする必要があります。

次のコマンドを使用して、RedisサーバーとPrometheusサイドカーコンテナーを含むポッドをデプロイできます。

```
# Clone the repo if you don't have it already
git clone git@github.com:mateobur/prometheus-monitoring-guide.git
kubectl create -f prometheus-monitoring-guide/redis_prometheus_exporter.yaml
```

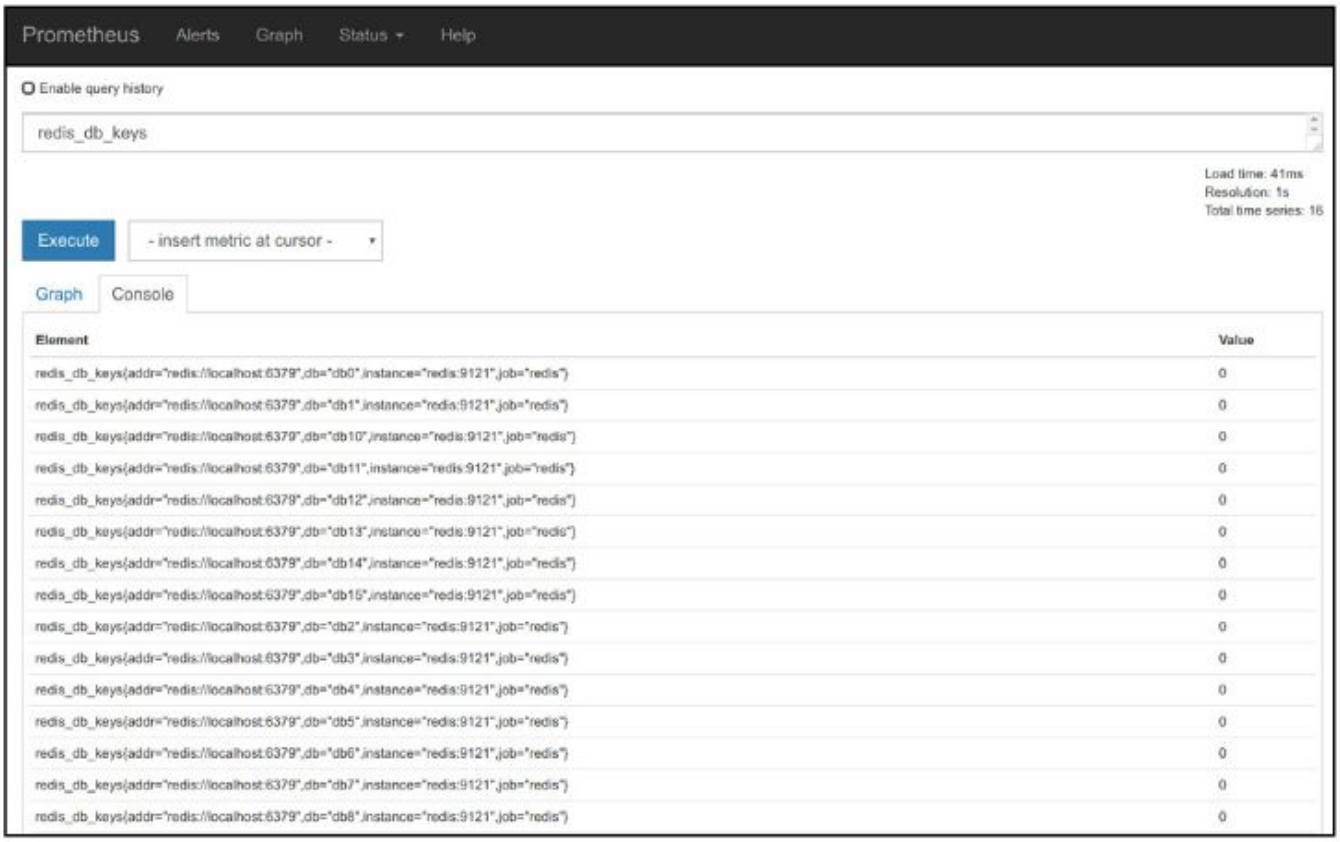
redisポッドを表示すると、内部に2つのコンテナーがあります。

```
kubectl get pod redis-546f6c4c9c-lmf6z
NAME                     READY   STATUS    RESTARTS   AGE
redis-546f6c4c9c-lmf6z   2/2     Running   0          2m
```

ここで、前のセクションで行ったように、Prometheus構成を更新し、リロードするだけです。

```
    - job_name: 'redis'
      static_configs:
        - targets: ['redis:9121']
```

すべてのredisサービスメトリクスを取得するには：



Kubernetesクラスター

クラスターにデプロイされたサービスの監視に加えて、Kubernetesクラスター自体も監視する必要があります。

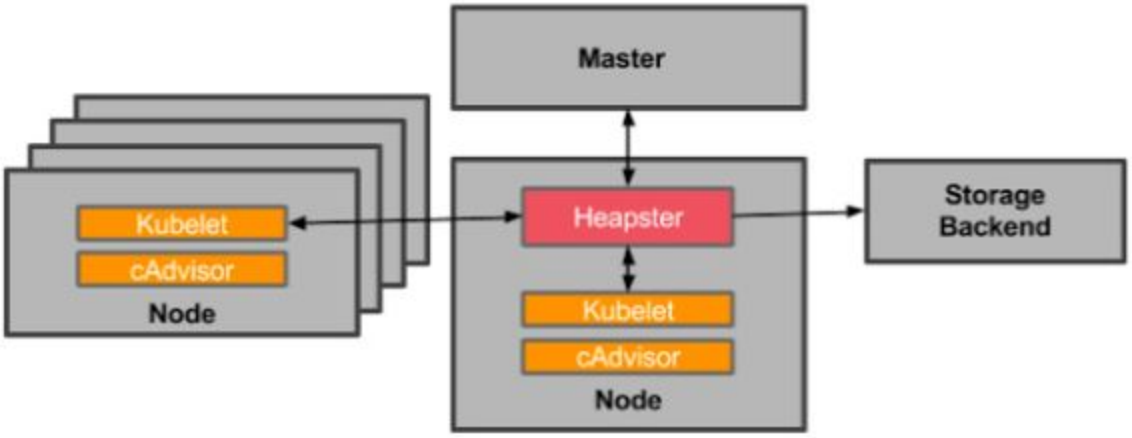
考慮すべきクラスター監視の3つの側面は次のとおりです。

1. Kubernetesホスト（ノード）– CPU、負荷、ディスク、メモリなどの古典的なsysadminメトリクス
2. オーケストレーションレベルのメトリクス-デプロイメントステート、リソース要求、スケジューリング、APIサーバーのレイテンシー時間など。
3. 内部kube-systemコンポーネント–スケジューラ、コントローラーマネージャー、DNSサービスなどの詳細なサービスメトリクス

Kubernetesの内部監視アーキテクチャは最近変更されたため、ここで要約します。詳細については、デザインプロポーザルをご覧ください。

Kubernetesコンポーネント

Heapster：HeapsterはKubernetesクラスターを監視します。 これは、クラスター内でポッドとして実行される監視およびイベントデータのクラスター全体のアグリゲーターです。



Kubelets / cAdvisorエンドポイントとは別に、kube-state-metricsなどの追加のメトリクスソースをHeapsterに追加できます（以下を参照）。

Heapsterは非推奨になりました。Heapsterの代わりはメトリクスサーバーです。

cAdvisor：cAdvisorは、オープンソースのコンテナリソース使用量およびパフォーマンス分析エージェントです。コンテナ専用に作成され、Dockerコンテナをネイティブでサポートします。Kubernetesでは、cAdvisorはKubeletバイナリの一部として実行されます。ノードローカルおよびDockerメトリクスを取得するアグリゲーターは、Kubelet Prometheusエンドポイントを直接スクレイプします。

Kube-state-metrics：kube-state-metricsは、Kubernetes APIサーバーをリッスンし、デプロイメント、ノード、ポッドなどのオブジェクトの状態に関するメトリクスを生成するシンプルなサービスです。 kube-state-metricsは単なるメトリクスエンドポイントであり、他のエンティティがそれをスクレ

イプして長期ストレージ（つまり、Prometheusサーバー）を提供する必要があることに注意することが重要です。

Metrics-server：Metrics Serverは、リソース使用状況データのクラスター全体のアグリゲーターです。これは、デフォルトのHeapsterの代替となることを目的としています。繰り返しますが、メトリクスサーバーは最後のデータポイントのみを提示し、長期保存は担当しません。

したがって：

- Kube-stateメトリクスは、デプロイメント、ポッド、レプリカのステータスなどのオーケストレーションメタデータに焦点を合わせています。
- メトリクスサーバーは、CPU、ファイル記述子、メモリ、リクエストレイテンシなどのリソースメトリクスAPIの実装に重点を置いています。

Kubernetesノード

Kubernetesのノードまたはホストを監視する必要があります。Linuxホストを監視するためのツールがたくさんあります。 このガイドでは、Prometheusノードエクスポーターを使用します。

- プロメテウスプロジェクト自体がホスト
- 次の章で説明するプロメテウスオペレーターを使用すると、自動的にデプロイメントされるものです
- DaemonSetとしてデプロイできるため、クラスターにノードを追加または削除すると自動的にスケーリングされます。

このサービスをデプロイするには、たとえば、このリポジトリでDamonSetを使用するなど、いくつかのオプションがあります。

```
kubectl create ns monitoring
kubectl create -f https://raw.githubusercontent.com/bakins/minikube-
prometheus-demo/master/node-exporter-daemonset.yml
```

またはHelm / Tillerを使用：

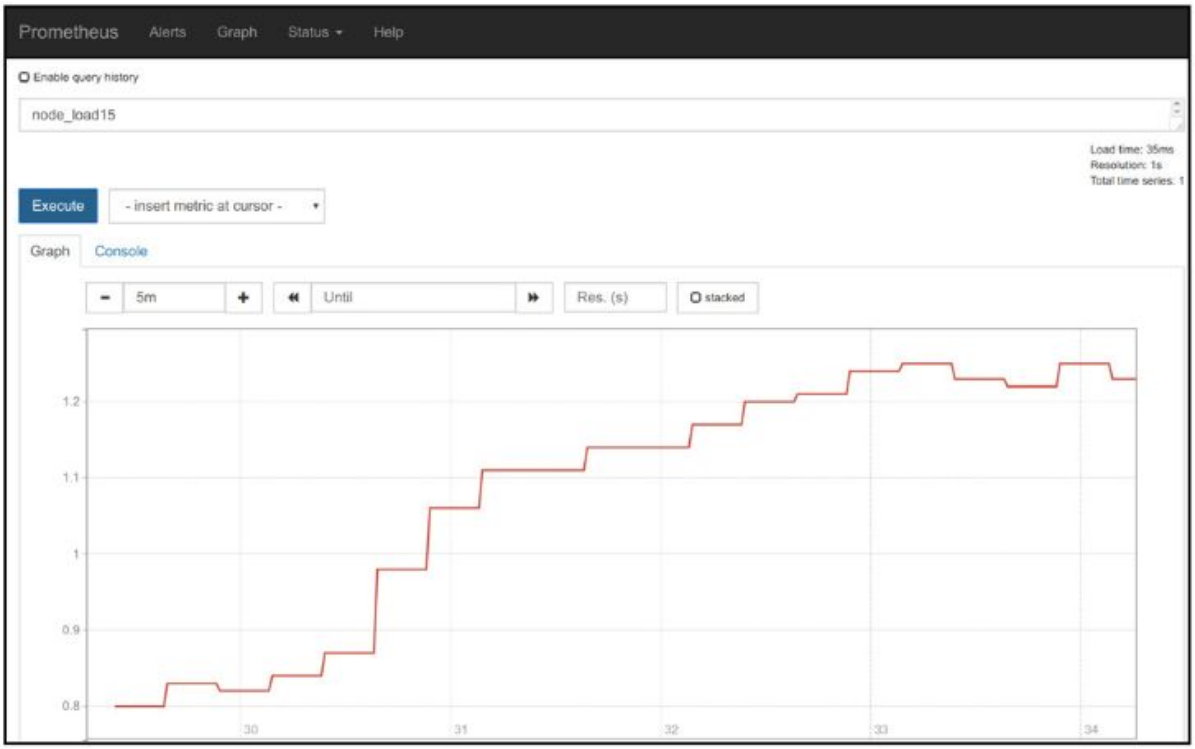
Helmを使用する場合は、先に進む前に、TillerコンポーネントのRBACロールとサービスアカウントを作成することを忘れないでください。

```
helm init --service-account tiller
helm install --name node-exporter stable/prometheus-node-exporter
```

chartをインストールして実行すると、スクレイピングする必要があるサービスを表示できます。

```
kubectl get svc
NAME                                    Type        CLuster-IP     External-IP  Port(s)
node-exporter-prometheus-node-exporter  ClusterIP   10.101.57.207  <none>       9100/TCP
```

前のセクションで行ったようにスクレイプ構成を追加したら、ノードメトリクスの収集と表示を開始できます。



Kube-stateメトリクス

kube-state-metricsのデプロイと監視も非常に簡単なタスクです。 以下の例のように直接デプロイするか、Helmチャートを使用できます。

```
git clone https://github.com/kubernetes/kube-state-metrics.git
kubectl apply -f kube-state-metrics/kubernetes/
...
kubectl get svc -n kube-system

NAME                 TYPE      CLUSTER-IP     EXTERNAL-IP  PORT(S)           AGE
kube-dns             ClusterIP 10.96.0.10     <none>       53/UDP,53/TCP     13h
kube-state-metrics ClusterIP 10.102.12.190 <none>       8080/TCP,8081/TCP 1h
```

Prometheus構成でそのサービス（ポート8080）を単純にスクレイプします。 今回は必ずFQDNを使用してください。

```
- job_name: 'kube-state-metrics'
   static_configs:
   - targets: ['kube-state-metrics.kube-system.svc.cluster.local:8080']
```

内部コンポーネント

プロメテウスを使用して内部パフォーマンスメトリクスを公開できるetcd、kube-scheduler、kube-controllerなどのKubernetesコンポーネントがいくつかあります。

それらを監視することは、次の2つの特性を持つ他のPrometheusエンドポイントを監視することに非常に似ています。

- これらのプロセスがリッスン、httpスキーム、およびセキュリティ（HTTP、HTTPS、RBAC）に使用するネットワークインターフェイスは、デプロイ方法と構成テンプレートによって異なります。
  - 多くの場合、これらのサービスはホスティングノードのローカルホストでのみリッスンしており、プロメテウスポッドから到達するのが困難になっています。

- これらのコンポーネントには、ポッドを指すKubernetesサービスがない場合がありますが、いつでも作成できます。

デプロイメント方法と構成によっては、Kubernetesサービスがローカルホストのみでリッスンしている場合があります。この例では、minikubeを使用して作業を簡単にします。

Minikubeでは、ローカルのシングルノードKubernetes仮想マシンを数分で生成できます。

これは、ホストされているクラスター、GKE、AWSなどでも機能しますが、構成を変更してサービスを再起動するか、追加のネットワークルートを提供することにより、サービスポートに到達する必要があります。

minikubeのインストールは、かなり簡単なプロセスです。最初にバイナリをインストールし、次にすべてのインターフェイスでkube-schedulerサービスを公開するクラスターを作成します。

```
minikube start --memory=4096 --bootstrapper=kubeadm --extra-
config=kubelet.authentication-token-webhook=true --extra-config=kubelet.
authorization-mode=Webhook --extra-config=scheduler.address=0.0.0.0
--extra-config=controller-manager.address=0.0.0.0
```

kube-schedulerポッドを指すサービスを作成します。

```
kind: Service
apiVersion: v1
metadata:
  name: scheduler-service
  namespace: kube-system
spec:
  selector:
    component: kube-scheduler
  ports:
  - name: scheduler
    protocol: TCP
    port: 10251
    targetPort: 10251
```

これで、エンドポイント：scheduler-service.kube-system.svcをスクレイピングできるようになります。

cluster.local：10251

まとめ

Prometheusは、最も人気のあるDockerおよびKubernetesモニタリングツールの1つになりつつあります。この電子書籍を読んで、あなたはその利点に精通しており、おそらく既にPrometheusをインストールし、エンドユーザーアプリとKubernetesクラスターエンドポイントでデプロイしているでしょう。

次のステップに進む準備はできましたか？ Sysdigブログにアクセスして、PromQL言語を使用したメトリクスの集計、アラートの起動、視覚化ダッシュボードの生成など、通常Prometheusサービスと共にデプロイされる追加コンポーネントの情報を確認してください。