記事
Tomohiro Iwamoto · 2020年12月10日 17m read

Open Exchange

Google Kubernetes Engine にデプロイされた IRIS ベースのサービスに TLS と DNS を追加する

この記事は、GitHub Actions を使って GKE に InterSystems IRIS Solution をデプロイする
の継続記事で、そこではGitHub Actions パイプラインを使って、 Terraform で作成された
Google Kubernetes クラスタにzpm-registry をデプロイしています。
繰り返しにならないよう、次の項目を満たしたものを開始点とします。

訳者注)
??????????????????????????????????GKE??????????????????????
??????????????????

- リポジトリ Github Actions + GKE + zpm example をフォーク済みで、フォークでの Actions を許可していること。 この記事を通して、ルートディレクトリは <rootrepodir> として参照されます。
- Terraform ファイルのプレースホルダを置換済みであること。
- GitHub Actions を使って GKE に InterSystems IRIS Solution をデプロイする の唯一の表に記載されているすべてのシークレットを（GitHub Actions Secrets ページで）作成済みであること。
- コピーペースト作業を単純化するために、すべてのコードサンプルが、GitHub-GKE-TLS リポジトリに保存されるようになっていること。

上記のすべてを満たしていることを前提に、先に進みましょう。

????

前回、zpm-registry への接続は、次のように行いました。
curl -XGET -u system:SYS 104.199.6.32:52773/registry/packages/-/all

IP アドレスの前にプロトコルがない場合は HTTP
を使用してい
ることを示します。つまり
、トラフィックは非暗号化であるため、かの悪名高いイブ
によってパスワードを盗み聞きされる可能性があります。

イブが盗聴できない
ようにするには、トラフィックを暗号化する、つまり HTTPS を使用する必要があります。
それは可能なことなのでしょうか。
104.199.6.32:52773/registry/packages → https://104.199.6.32:52773/registry/packages

概して言えば、可能です。 IP アドレスの証明書を取得すればよいのですが、

このソリューションには欠点があります。 詳細については、Using an IP Address in an SSL Certificate と SSL for IP Address を読んでいただきたいのですが、 要約すると、静的 IP アドレスが必要であり、その機能を備えた証明書プロバイダーは無料ではありません。 メリットについても、疑わしい部分があります。

一般的な無料プロバイダーは、9 Best Free SSL Certificate Sources にも紹介されているように、幸いにも存在はします。 その 1 つは Let ’ s Encrypt ですが、証明書を発行するのはドメイン名に対してのみです。ちなみに、証明書に IP アドレスを追加する計画は上がってはいます。

したがって、トラフィックを暗号化するには、まず、ドメイン名を取得する必要があります。 ドメイン名をすでにお持ちの方は、次のセクションにスキップしてください。

ドメイン名の取得

ドメインレジストラからドメイン名を購入します。 かなりの数のレジストラが存在し、価格はドメイン名やサービスレベルによって異なります。 開発の目的には、たとえば .dev のドメインを使うことができますし、おそらく安価でもあります。

こ
こで
は、ドメ
イン登録プロセス
には触れませんが、それほど複雑な
ものではありません。
これ以降では、登録済みのドメイン名を *example.com* として説明することにします。 これを自分のドメインに置き換えてください。

ドメイン登録プロセスが完了すると、ドメイン名とドメインゾーンを得られますが、これらは別々のものとして考える必要があります。 Definition - Domains vs. Zones の説明と短い DNS Zones の動画を見て、この違いを理解してください。

簡単に説明すると、ドメインを組織と考えた場合、ゾーンはその組織内の部署として捉えることができます。 このチュートリアルでは、ドメインは example.com であり、ゾーンも同様に example.com と呼んでいます。 （小さな組織では、部署が 1 つしかない場合があります。） 各 DNS ゾーンには、ゾーン内の IP アドレスとドメイン名を認識する特別なサーバーが必要です。 これらのリンクは、リソースレコード（RR）と呼ばれており、それぞれに種類が異なる場合があります（DNS Record types をご覧ください）。 最も広く使用されているのは、A レコードです。

「ネームサーバー」は、ドメインレジストラが提供する特別なサーバーです。 たとえば、私が契約しているレジストラからは、次の 2 つのネームサーバーが提供されています。

ns39.domaincontrol.com

ns40.domaincontrol.com

次のようなリソースレコード（サーバードメイン名 = IP

アドレス）を作成する必要があります。
zpm.example.com = 104.199.6.32

これは、ゾーン *example.com* に A レコードを作成して行います。
このレコードを保存すると、世界中にあるほかの DNS
サーバーによってこの更新内容が認識され、最終的に、 zpm-registry
を *zpm.example.com:52773/registry/* という名前で参照できるようになります。

ドメインと kubernetes

覚えているかと思いますが、zpm サービスの IP アドレスは、Kubernetes Service（Load Balancer タイプ）をデプロイ中に作成されました。 zpm-registry
を試して削除した後にもう一度 zpm-registry をデプロイすることにした場合は、別の IP
アドレスが割り当てられる可能性があります。 その場合は、もう一度 DNS レジストラの Web
コンソールにアクセスして、zpm.example.com の IP アドレスを新たに設定してください。

これには、もう 1 つの方法があります。 Kubernetes のデプロイ中に、External DNS
という、Kubernetes Services または Ingress から新しく作成された IP
アドレスを取得して対応する DNS
レコードを作成する、ヘルパーツールをデプロイできます。
このツールはすべてのレジストラをサポートしているわけではありませんが、DNS
ゾーン、ネームサーバーの提供、およびリソースレコードの保存を行える、Google Cloud DNS
をサポートしています。

Google Cloud DNS を使用するには、レジストラの Web コンソールで、DNS ゾーンの
example.com の管理を Google Cloud DNS に移行する必要があります。
これを行うには、ドメインレジストラが提供したネームサーバーを Google Cloud DNS
が提供するものに変更します。 Google コンソールに *example.com* ゾーンを作成し、Google
が提供するネームサーバーをコピーして貼り付けてください。
詳細は以下を参照してください。

コードに外部 DNS を追加して Google Cloud DNS を作成する方法を見てみましょう。
ただし、スペースを節約するために、ここではコードの一部のみを記載します。
前述のとおり、完全なサンプルは、 GitHub GKE TLS リポジトリにあります。

外部 DNS の追加

このアプリケーションを GKE にデプロイするには、Helm を利用します。
これについては、「An Introduction to Helm, the Package Manager for Kubernetes」と公式ドキュメント
が役立つでしょう。

パイプラインファイルの「kubernetes-
deploy」ステージの最後のジョブとして、次の行を追加してください。
また、「env」セクションには、新しい変数もいくつか追加します。
$ cat <rootrepodir>/.github/workflows/workflow.yaml

...

env:

...

```
  DNSZONE: example.com

  HELMVERSION: 3.1.1

  EXTERNALDNSCHARTVERSION: 2.20.6
```

...


```
jobs:
```

```
...
  kubernetes-deploy:

...

  steps:

...
```



```
    - name: Install External DNS
```

```
      run: |
        wget -q https://get.helm.sh/helm-v${HELMVERSION}-linux-amd64.tar.gz

        tar -zxvf helm-v${HELMVERSION}-linux-amd64.tar.gz
cd linux-amd64
        ./helm version

        ./helm repo add bitnami https://charts.bitnami.com/bitnami



        gcloud container clusters get-credentials ${GKECLUSTER} --zone
    ${GKEZONE} --project ${PROJECTID}
        echo ${GOOGLECREDENTIALS} > ./credentials.json

        kubectl create secret generic external-dns --from-file=./credentials.json --dry-
    run -o yaml | kubectl apply -f -
        ./helm upgrade external-dns bitnami/external-dns \

          --install \



          --atomic \









          --version=${EXTERNALDNSCHARTVERSION} \
```

```
--set google.project=${PROJECTID} \
--set google.serviceAccountSecret=external-dns \

--set registry=txt \




--set txtOwnerId=k8s \









--set policy=sync \
```



```
--set domainFilters={${DNSZONE}} \
```

```
    --set rbac.create=true
```

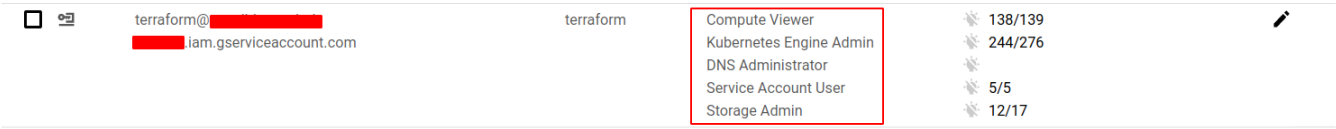
--set が設定するパラメータについては、External DNS
チャートのドキュメントを参照してください。
とりあえず上記の行を追加して、次に進みましょう。

クラウド DNS の作成

まず、<rootrepodir>　/terraform/ ディレクトリに新しいファイルを追加します。
あなたのドメインゾーンを使用してください。

```
$ cat <rootrepodir>    /terraform/clouddns.tf
resource "googlednsmanagedzone" "my-zone" {
  name = "zpm-zone"
  dnsname = "example.com."
  description = "My DNS zone"
}
```

また、Terraform に代わって機能するユーザーには、少なくとも次のロールが与えられていることを確認してください（DNS 管理者および Kubernetes Engine
管理者ロールに注意してください）。



GitHub Actions パイプラインを実行している場合は、これらが作成されます。
External DNS とCloud DNS の準備ができたら（実質的に準備できた時点で）、zpm-service をロードバランサーサービスタイプとは異なる方法で公開する方法を検討できるようになります。

Kubernetes

zpm-service は、通常の Kubernetes サービスロードバランサーを使って公開されています。ファイル <rootrepodir>/k8s/service.yaml を参照してください。 Kubernetes Service についての詳細は、「Service を使用したアプリケーションの公開」をお読みください。 重要なのは、ロードバランサーサービスにはドメイン名を設定する機能がないため、Kubernetes Services リソースが作成した実際の Google ロードバランサーが TCP/UDP レベルの OSI を操作するということです。 このレベルは、HTTP と証明書について何も関知していないため、 Google ネットワークロードバランサーを HTTP ロードバランサーに置き換える必要があります。
このようなロードバランサーを作るには、Kubernetes Service の代わりに Kubernetes Ingress リソースを使用できます。

ここで、「Kubernetes NodePort vs LoadBalancer vs Ingress? When should I use what?」を読んでおくと良いでしょう。

??????????? ????? Ingress

?????????????? <root_repo_dir>/k8s/ ディレクトリに移動し、次の変更を行います。 Service のタイプは NodePort です。

```
$ cat <rootrepodir>    /k8s/service.yaml

apiVersion: v1


kind: Service




metadata:




  name: zpm-registry
```

```
spec:
  selector:

    app: zpm-registry



  ports:

  - protocol: TCP
```



```
    port: 52773
```

```
    targetPort: 52773
  type: NodePort
```

次に、Ingress マニフェストを追加する必要があります。

```
$ cat <rootrepodir>    /k8s/ingress.yaml
apiVersion: extensions/v1beta1


kind: Ingress




metadata:




  annotations:
```

```
    networking.gke.io/managed-certificates: zpm-registry-certificate
    external-dns.alpha.kubernetes.io/hostname: zpm.example.com

  name: zpm-registry



  namespace: iris



spec:
```



```
  rules:
```

```
- host: zpm.example.com
  http:
    paths:
    - backend:
        serviceName: zpm-registry
        servicePort: 52773
```

```
        path: /*
```

また、太字で示されている行をワークフローファイルに追加します。

```
$ cat <rootrepodir>    /.github/workflows/workflow.yaml

...

- name: Apply Kubernetes manifests



  working-directory: ./k8s/

...
```

```
kubectl apply -f ingress.yaml
```

このマニフェストをデプロイすると、Google Cloud コントローラによって、外部 HTTP ロードバランサーが作成されます。これは、ホストである zpm.example.com へのトラフィックをリスンし、このトラフィックを、Kubernetes NodePort Service デプロイ中に開かれたポート経由で、すべての Kubernetes ノードに送信します。
このポートは任意ですが、完全に自動化されているため、気にする必要はありません。

アノテーションを使用すると、Ingress のより詳細な構成を定義することができます。

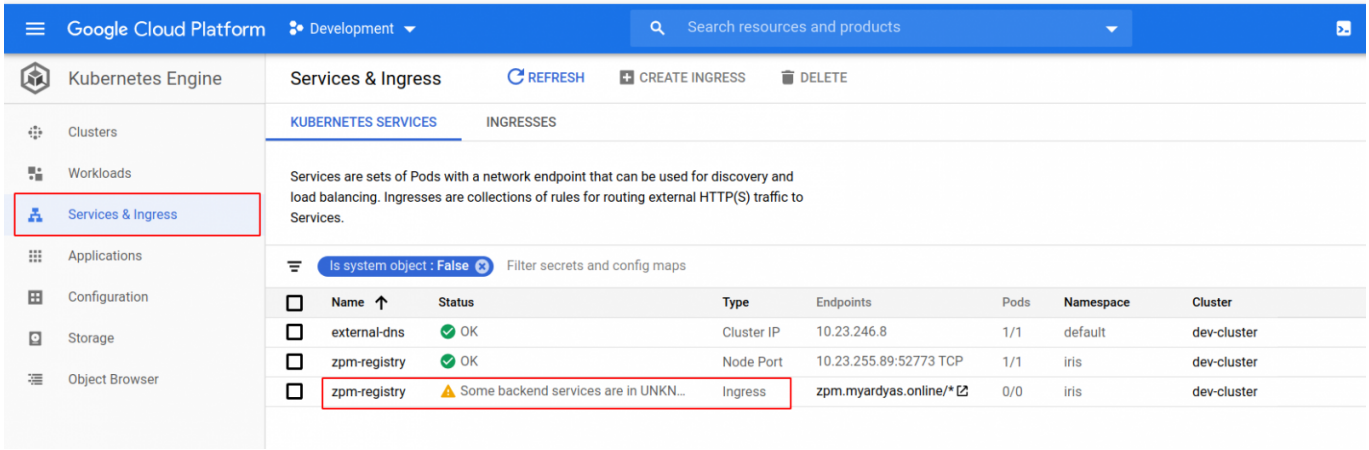
```
annotations:

  kubernetes.io/ingress.class: gce


  networking.gke.io/managed-certificates: zpm-registry-certificate


  external-dns.alpha.kubernetes.io/hostname: zpm.example.com
```

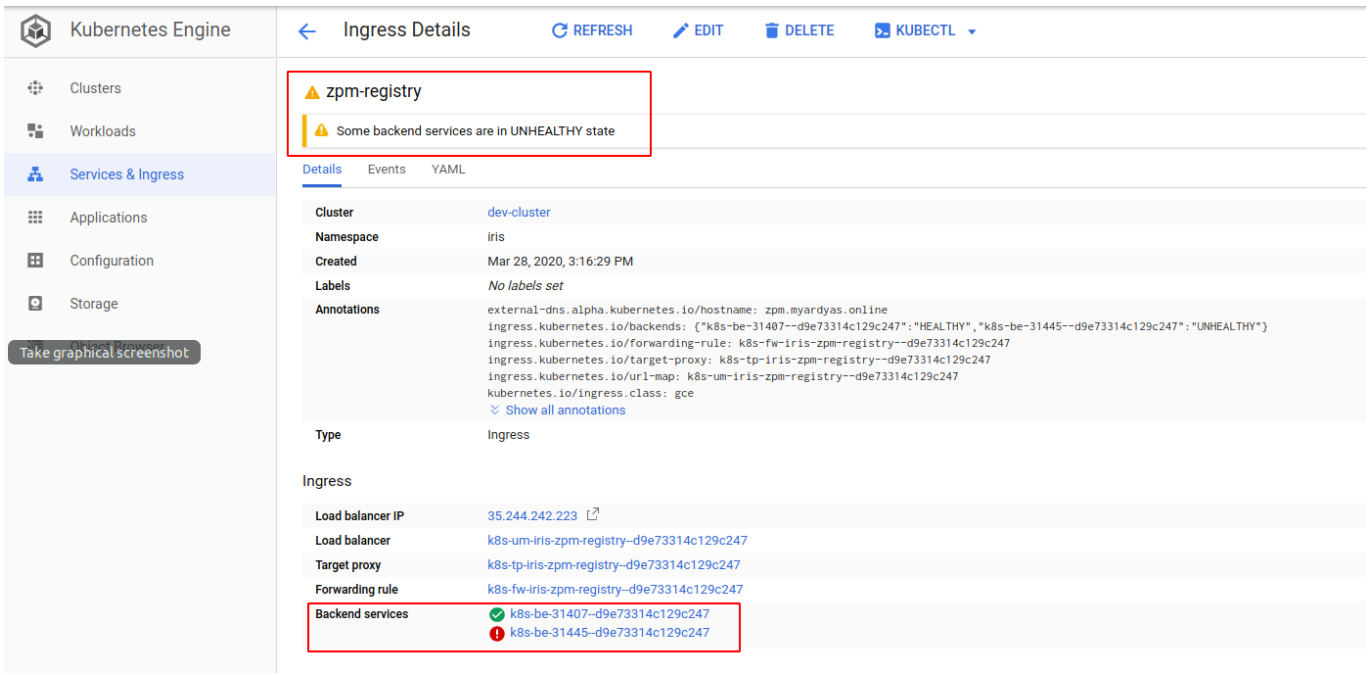
最初の行は、"gce" Ingress コントローラを使用することを示します。
このエンティティは、Ingress リソースに従って HTTP ロードバランサーを作成します。

2 行目は、証明書に関連する行です。 この設定については、後で説明します。

3 行目は、指定されたホスト名（zpm.example.com）を HTTP ロードバランサーの IP アドレスにバインドする外部 DNS を設定します。

このマニフェストを実装した場合、（約 10 分後に）Ingress が作成されてはいても、ほかのすべてが機能しているわけではないことがわかります。





"Backend services"とは何でしょうか。このうちの 1 つはうまく機能していないようです。

"k8s-be-31407-..." や "k8s-be-31445-..."などの名前がありますか？ これらは、1 つの Kubernetes ノードで開かれているポートです。 ポートの番号はそれぞれ異なります。

31407
は、ヘルスチェック用に開かれているポートで、ノードが連続してアライブであるように、HTTP ロードバランサーが送信するポートです。

Kubernetes に接続して、ノードポート 31407
をプロキシすると、ヘルスチェックの結果を確認できます。

```
$ gcloud container clusters get-credentials <CLUSTERNAME>  --zone <LOCATION> --project <PROJECTID>
$ kubectl proxy &

$ curl localhost:8001/healthz
```

ok


```
$ fg
```

^Ctrl+C

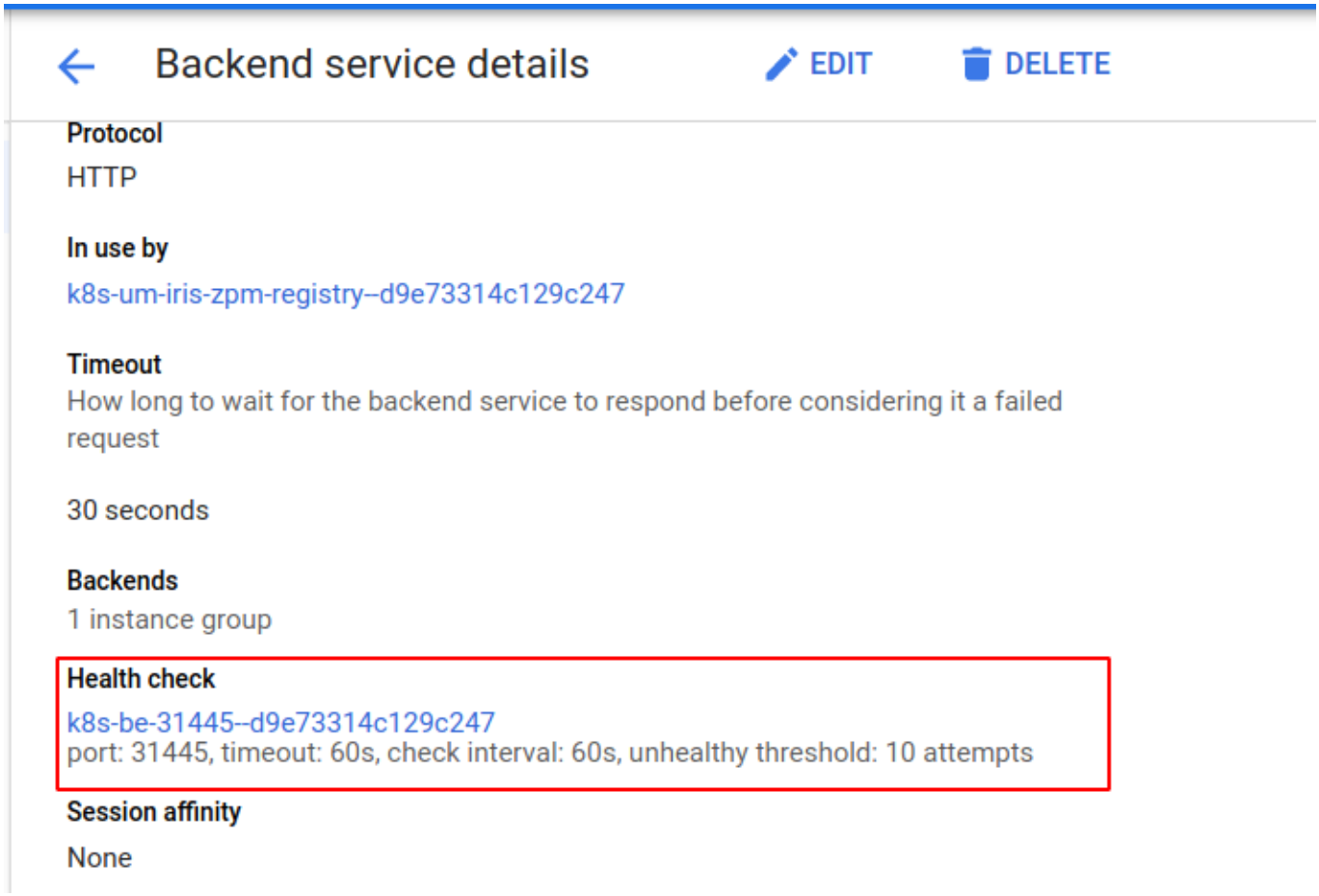
もう 1 つの 31445 ポートは、zpm-registry サービス用に開かれている NodePort です。
$ kubectl -n iris get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

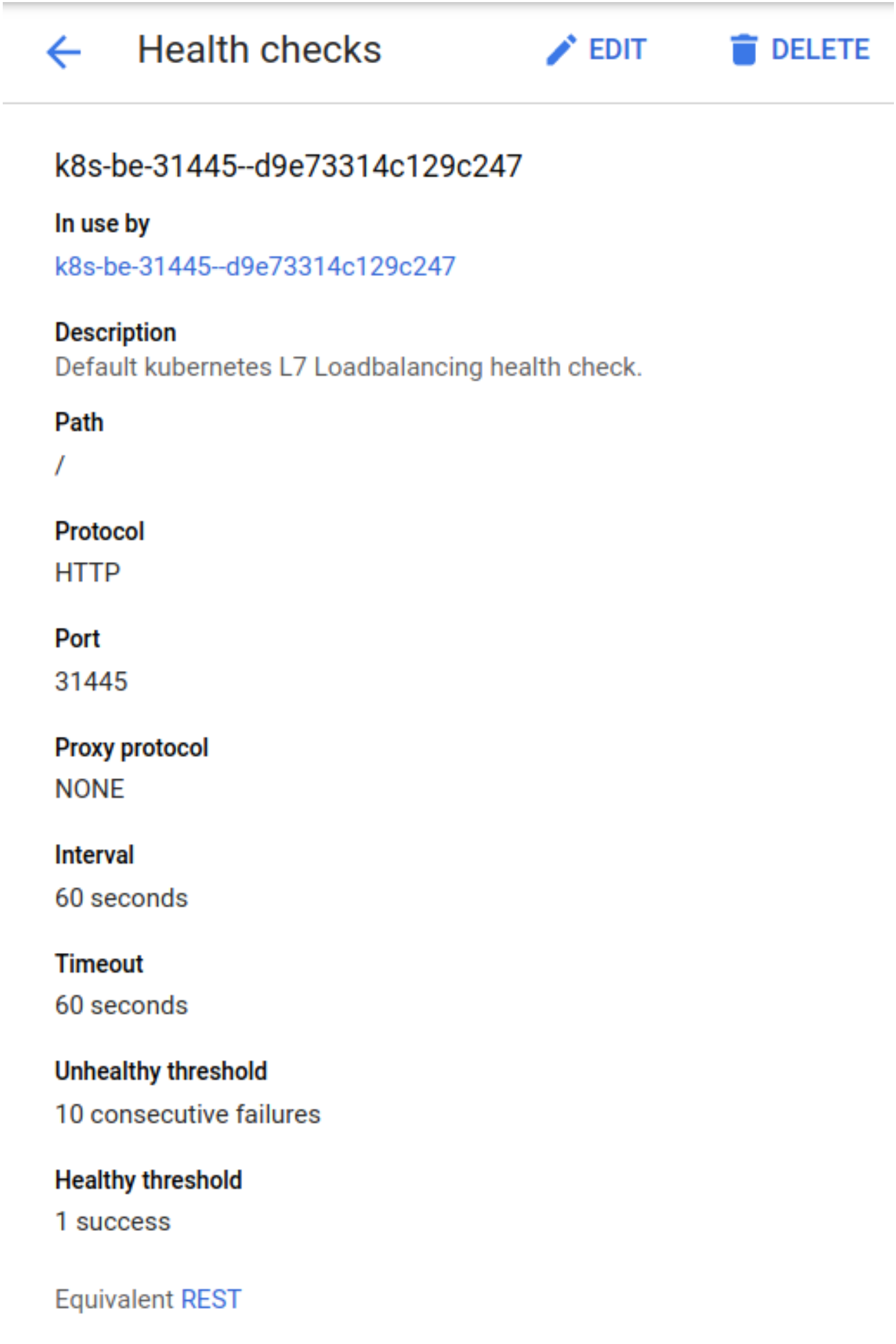
zpm-registry NodePort 10.23.255.89 <NONE> 52773:31445/TCP 24m

また、デフォルトのヘルスチェックでは、HTTPロードバランサーはこのサービスが停止しているというレポートを送信します。 本当にそうなのでしょうか？

それらのヘルスチェックの詳細を確認する必要があります。 Backend service 名をクリックし、 下に少しスクロールして、ヘルスチェック名を確認してください。



詳細を確認するには、ヘルスチェック名をクリックします。


Health checks
EDIT
DELETE
k8s-be-31445--d9e73314c129c247
In use by
k8s-be-31445--d9e73314c129c247
Description
Default kubernetes L7 Loadbalancing health check.
Path
/
Protocol
HTTP
Port
31445
Proxy protocol
NONE
Interval
60 seconds
Timeout
60 seconds
Unhealthy threshold
10 consecutive failures
Healthy threshold
1 success
Equivalent REST

ヘルスチェックの "/" パスを "/csp/sys/UtilHome.csp" のように、IRIS
が理解できるパスに置き換える必要があります。

```
$ curl -I localhost:52773/csp/sys/UtilHome.csp

Handling connection for 52773
HTTP/1.1 200 OK
```

では、ヘルスチェックに新しいパスを設定するには、どうすればよいのでしょうか。
デフォルトの "/" パスが存在する場合は、その代わりに、Readiness/Liveness probes
を使用してください。

Kubernetes StatefulSet マニフェストにプローブを追加してみましょう。

```
$ cat <rootrepodir>/k8s/statefulset.tpl

...


containers:



- image: DOCKERREPONAME:DOCKERIMAGETAG



...
```

```
- containerPort: 52773
  name: web

readinessProbe:


  httpGet:


    path: /csp/sys/UtilHome.csp
```



```
    port: 52773
```

```
    initialDelaySeconds: 10
    periodSeconds: 10

  livenessProbe:


    httpGet:


      path: /csp/sys/UtilHome.csp


      port: 52773
```

```
    periodSeconds: 10
  volumeMounts:

  - mountPath: /opt/zpm/REGISTRY-DATA



    name: zpm-registry-volume



  - mountPath: /mount-helper
```



```
    name: mount-helper
```

ここまでで、いくつかの変更を加えて、メインイベントである証明書への扉を開くことができました。 では、ラストスパートに取り掛かるとしましょう。

証明書の取得

Kubernetes の SSL/TLS
証明書を取得するさまざまな方法を説明した次の動画をぜひご覧ください。

- Create a Kubernetes TLS Ingress from scratch in Minikube
- Automatically Provision TLS Certificates in K8s with cert-manager
- Use cert-manager with Let's Encrypt® Certificates Tutorial: Automatic Browser-Trusted HTTPS
- Super easy new way to add HTTPS to Kubernetes apps with ManagedCertificates on GKE

要約すると、証明書を取得するにはいくつかの方法があり、openssl 自己証明書、Let ' s Encrypt の certbot（手動）、Let's Encrypt に接続した cert-manager（自動）、そしてネイティブの Google アプローチである Managed Certificate を使用できます。ここでは、単純にするために、最後の Managed Certificate を使用します。

では追加しましょう。
$ cat <rootrepodir>/k8s/managed-certificate.yaml

apiVersion: networking.gke.io/v1beta1

kind: ManagedCertificate

metadata:

name: zpm-registry-certificate

spec:

  domains:

  - zpm.example.com

太字で示された行をデプロイパイプラインに追加します。

 $ cat <rootrepodir>/.github/workflows/workflow.yaml

...

- name: Apply Kubernetes manifests

  working-directory: ./k8s/

  run: |

${GKEZONE} --project ${PROJECTID}
kubectl apply -f namespace.yaml

kubectl apply -f managed-certificate.yaml

kubectl apply -f service.yaml

...

Ingress での有効化は、Ingress アノテーションとして先に行いましたが、覚えていますか？
$ cat <rootrepodir>/k8s/ingress.yaml

...

annotations:

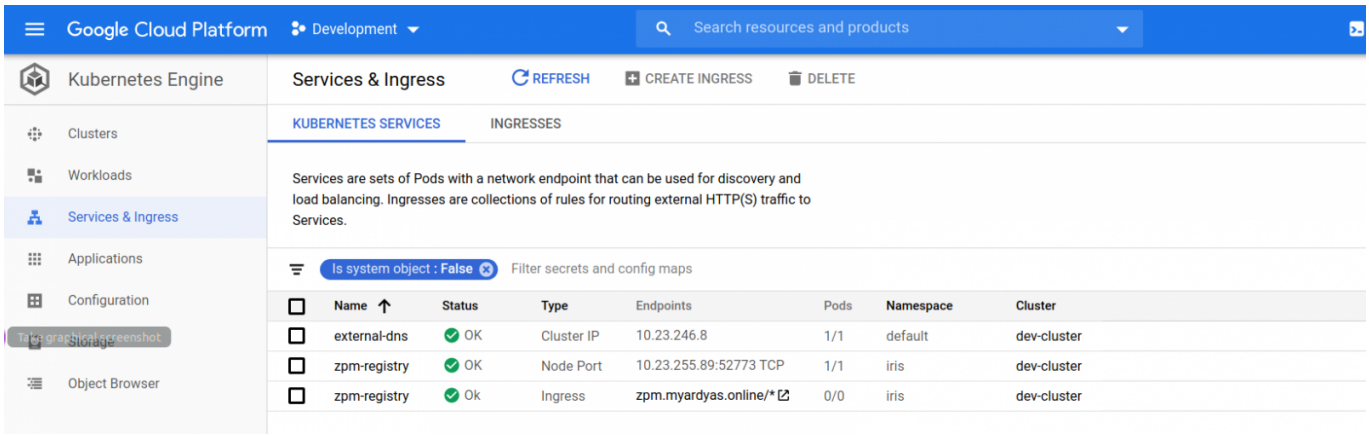
kubernetes.io/ingress.class: gce

networking.gke.io/managed-certificates: zpm-registry-certificate

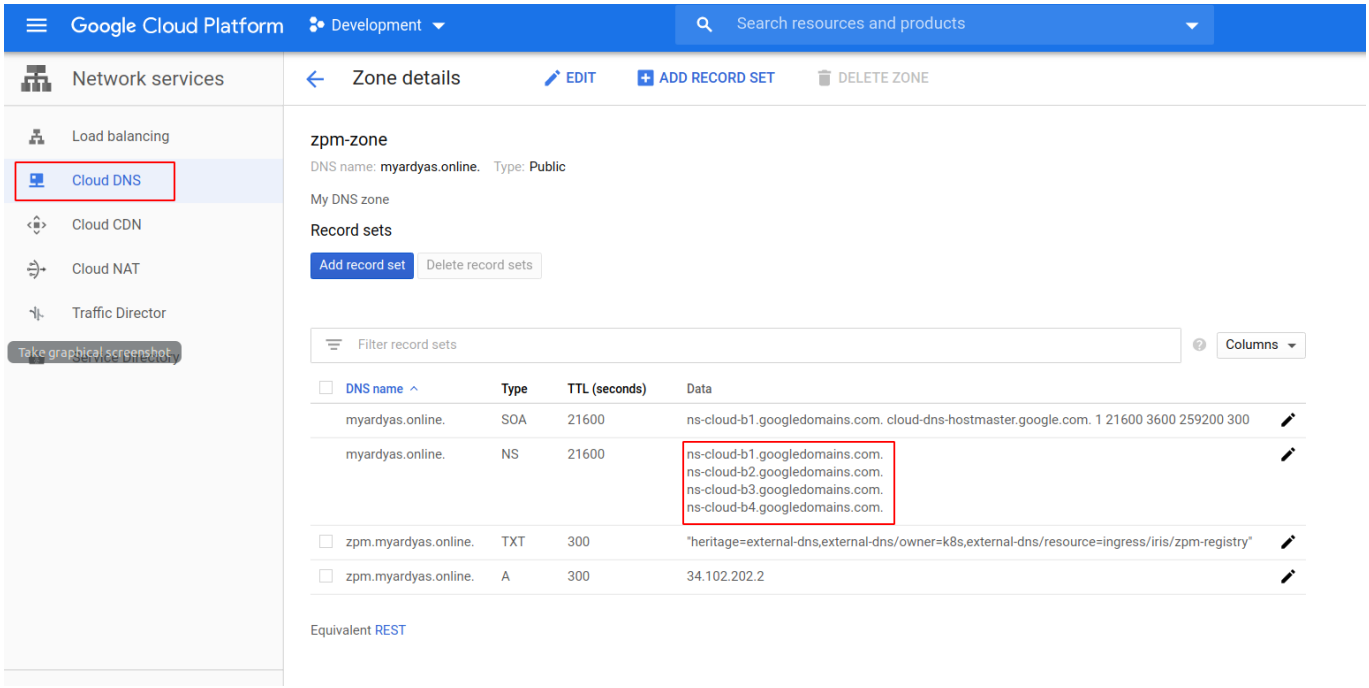
...

これで、すべての変更をリポジトリにプッシュする準備が整いました。
$ git add .github/ terraform/ k8s/
$ git commit -m "Add TLS to GKE deploy"
$ git push

15 分ほどすると（クラスタプロビジョニングを実行する必要があります）、「緑信号」の Ingress を確認できるでしょう。



次に、Google が提供した少なくとも 2 つのネームサーバーをドメイン名レジストラコンソールで設定する必要があります（Google Domains 以外のレジストラを使用している場合）。



このプロセスはレジストラによって異なるため、ここでは説明しません。通常、レジストラのドキュメントで詳しく説明されています。

Google は、External DNS

が自動的に作成した新しいリソースレコードをすでに認識しています。

```
$ dig +short @ns-cloud-b1.googledomains.com. zpm.example.com
34.102.202.2
```

このレコードが世界中に伝搬されるまでにはしばらく時間がかかりますが、
とはいえ、最終的には次のようになります。

```
$ dig +short zpm.example.com
34.102.202.2
```

証明書のステータスを確認することをお勧めします。

```
 $ gcloud container clusters get-credentials <CLUSTERNAME> --zone
<LOCATION> --project <PROJECTID>
$ kubectl -n iris get managedcertificate zpm-registry-certificate -ojson ¦ jq '.status'

{


  "certificateName": "mcrt-158f20bb-cdd3-451d-8cb1-4a172244c14f",




  "certificateStatus": "Provisioning",
```

{
"domain": "zpm.myardyas.online",

"status": "Provisioning"

}

]

}

さまざまなステータスの意味については、「Google マネージド SSL 証明書の使用」のページをご覧ください。 証明書を初めてプロビジョニングする場合は、1時間ほどかかることがあります。

domainStatus "FailedNotVisible" が発生することがありますが、 この場合は、Google ネームサーバーを DNS
レジストラコンソールで実際に追加していることを確認してください。

certificateStatus と domainStatus の両方が利用可能になるまで待つことをお勧めします。「Google マネージド SSL 証明書の使用」で説明されているとおり、しばらく時間がかかることがありますが、
最終的には、次のようにして zpm-registry を呼び出せるようになるでしょう。

```
curl -XGET -u system:SYS  https://zpm.example.com/registry/packages/-/all
```

まとめ

Google は、Kubernetes リソースに基づいて、必要なすべての Google リソースを作成することに長けています。

また、Google マネージド証明書は、証明書の取得を大幅に単純化できる優れた機能です。

Google リソース（GKE、CloudDNS）の維持には費用が掛かります。そのため、いつものように不要になったら忘れずに削除するようにしてください。

#DevOps #GitHub #Kubernetes #クラウド #コンテナ化 #ベストプラクティス #InterSystems IRIS #Open Exchange
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:https://jp.community.intersystems.com/post/google-kubernetes-engine-%E3%81%AB%E3%83%87%E3%83%97%E3%83%AD%E3%82%A4%E3%81%95%E3%82%8C%E3%81%9F-iris-%E3%83%99%E3%83%BC%E3%82%B9%E3%81%AE%E3%82%B5%E3%83%BC%E3%83%93%E3%82%B9%E3%81%AB-tls-%E3%81%A8-dns-%E3%82%92%E8%BF%BD%E5%8A%A0%E3%81%99%E3%82%8B