記事
Toshihiko Minamoto · 2021年3月17日 31m read

Open Exchange

Kubernetesにおけるミラーリングを使用しない高可用性IRISデプロイ

この記事では、従来のIRISミラーリング構成の代わりに、Kubernetesの
Deploymentと分散永続ストレージを使って高可用性IRIS構成を構築します。 このデプロイでは、ノード、ストレージ、アベイラビリティーゾーンといったインフラストラクチャ関連の障害に耐えることが可能です。 以下に説明する方法を使用することで、RTOがわずかに延長されますが、デプロイの複雑さが大幅に軽減されます。

図1 - 従来のミラーリング構成と分散ストレージを使ったKubernetesの比較

この記事に記載されているすべてのソースコードは、https://github.com/antonum/ha-iris-k8s より入手できます。

要約

3ノードクラスタを実行しており、Kubernetesにいくらか詳しい方は、このまま以下のコードを使用してください。

```
kubectl apply -f https://raw.githubusercontent.com/longhorn/longhorn/master/deploy/longhorn.yaml
kubectl apply -f https://github.com/antonum/ha-iris-k8s/raw/main/tldr.yaml
```

上記の2行の意味がよくわからない方、またはこれらを実行できるシステムがない場合は、「高可用性要件」のセクションにお進みください。 説明しながら進めていきます。

最初の行は、Longhornというオープンソースの分散Kubernetesストレージをインストールしています。
2行目は、DurableSYS用にLonghornベースのボリュームを使って、InterSystems
IRISデプロイをインストールしています。

すべてのポッドが実行状態になるまで待ちます。 kubectl get pods -A

上記の手順を済ませると、http://<IRIS Service Public IP>:52773/csp/sys/%25CSP.Portal.Home.zen
にあるIRIS管理ポータル（デフォルトのパスワードは「SYS」）にアクセスできるようになります。また、次のようにしてIRISコマンドラインも使用できるようになります。

```
kubectl exec -it iris-podName-xxxx -- iris session iris
```

障害をシミュレートする

では、実際に操作してみましょう。 ただし、操作の前に、データベースにデータを追加して、IRISがオンラインになったときに存在することを確認してください。

```
kubectl exec -it iris-6d8896d584-8lzn5 -- iris session iris
USER>set ^k8stest($i(^k8stest))=$zdt($h)_" running on "_$system.INetInfo.LocalHostNam
e()
USER>zw ^k8stest
^k8stest=1
^k8stest(1)="01/14/2021 14:13:19 running on iris-6d8896d584-8lzn5"
```

ここからが「カオスエンジニアリング」の始まりです。

```
# IRIS??? - ?????????????
kubectl exec -it iris-6d8896d584-8lzn5 -- iris stop iris quietly

# ?????? - ??????????
kubectl delete pod iris-6d8896d584-8lzn5

# iris??????????????????????? - ???????????????????
kubectl drain aks-agentpool-29845772-vmss000001 --delete-local-data --ignore-
daemonsets --force

# ?????? - ???????????????????
# ????kubectl?????????? ???????????VM?????????????
?????????????????????????????????????? ??????????
```

高可用性要件

以下の障害に耐えられるシステムを構築しています。

- コンテナ/VM内のIRISインスタンス。 IRIS - レベル障害
- ポッド/コンテナの障害
- 個々のクラスタノードの一時的な使用不能。
  アベイラビリティーゾーンが一時的にオフラインになる場合などが挙げられます。
- 個々のクラスタノードまたはディスクの永久的障害。

基本的に、「障害をシミュレートする」セクションで試したシナリオです。

上記のいずれかの障害が発生すると、システムは人間が手をいれなくてもオンラインになり、データも失われません。 データの永続性が保証する範囲には一応制限がありますが、
ジャー
ナルサイクル
とアプリケーション内のト
ランザクションの使用状況に応じて、IRISで実現され
ます（
https://docs.intersystems.com/irisforhealthlatestj/csp/docbook/Doc.View.cls?KEY=GCDIjournal#GCDIjournalwritecycle）。いずれにしても、RPO（目標復旧ポイント）は2秒未満です。

システムの他のコンポーネント（Kubernetes
APIサービス、etcdデータベース、ロードバランサーサービス、DNSなど）はスコープ外であり、通常、Azure AKSやAWS EKSなどのマネージドKubernetesサービスで管理されます。

別の言い方をすれば、個別の計算コンポーネントやストレージコンポーネントの障害はユーザーが処理するものであり、その他のコンポーネントの障害はインフラストラクチャ/クラウドプロバイダーが対処するものと言えます。

アーキテクチャ

InterSystems IRISの高可用性について言えば、これまでミラーリングの使用が勧められてきました。
ミラーリングでは、データは、常時オンライン状態にある2つのIRISインスタンスが同期的に複製されます。 各ノードにはデータベースの完全なコピーが維持され、プライマリノードがオフラインになると、ユーザーはバックアップノードに接続し直すことができます。
基本的に、ミラーリング手法では、計算とストレージの両方の冗長性は、IRISが管理するものです。

ミラーをさまざまなアベイラビリティーゾーンにデプロイしたミラーリングにより、計算処理とストレージの障害に備えた必要な冗長性を実現し、わずか数秒という優れたRTO（目標復旧時間または障害後にシステムがオンラインに復旧するまでにかかる時間）を達成することができます。
AWSクラウドでミラーリングされたIRISの
デプロイテンプレートは、https://jp.community.intersystems.com/node/486206から入手できます。

一方で、ミラーリングには、設定やバックアップ/復旧手順が複雑で、セキュリティ設定とローカルのデータベース以外のファイルの複製機能が不足しているという欠点があります。

コンテナオーケストレータ–にはKubernetesなど（今や2021年。ほかにオーケストレーターってありますか?!）がありますが、これらは、障害時に、Deploymentオブジェクトが障害のあるIRISポッド/コンテナを自動的に再起動することで、計算冗長性を実現しています。
Kubernetesアーキテクチャ図に、実行中のIRISノードが1つしかないのはこのためです。
2つ目のIRISノードを常時実行し続ける代わりに、計算可用性をKubernetesにアウトソースしています。
Kubernetesは、元のIRISポッドが何らかの理由でエラーとなった場合に、IRISポッドの再作成を確保します。

図2 フェイルオーバーのシナリオ

ここまでよろしいでしょうか。 IRISノードに障害が発生すると、Kubernetesは単に新しいノードを作成します。
クラスタによって異なりますが、計算障害が発生してからIRISがオンラインになるまでには、10〜90秒かかります。 ミラーリングではわずか2秒で復旧されるわけですから、これはダウングレードとなりますが、万が一サービス停止となった場合に許容できる範囲であれば、複雑さが緩和されることは大きなメリットと言えます。
ミラーリングの構成は不要です。 セキュリティ設定やファイル複製を気にする必要もありません。

率直に言うと、KubernetesでIRISを実行し、コンテナにログインしても、高可用性環境で実行していることには気づくこともないでしょう。
単一インスタンスのIRISデプロイと全く同じように見え、何ら感触にも変化はありません。

では、ストレージはどうでしょうか。 データベースを扱っているわけですから気になります。
どのようなフェイルオーバーのシナリオであっても、システムはデータの永続性も処理する必要があります。
ミラーリングは、IRISノードのローカルである計算ノードに依存しているため、
ノードが停止するか、一時的に使用不可になってしまえば、そのノードのストレージも停止してしまいます。
ミラーリング構成では、IRISがIRISレベルでデータベースを複製するのはこれに対処するためです。

コンテナの再起動時に元の状態を維持したデータベースを使用できるだけでなく、ノードやネットワークのセグメント全体（アベイラビリティーゾーン）が停止するといったイベントに備えて、冗長性を提供できるストレージが必要です。 ほんの数年前、これを解決できる簡単な答えは存在しませんでした。
上の図から推測できるように、今ではその答えを得ています。 **分散コンテナストレージ**です。

分散ストレージは、基盤のホストボリュームを抽象化し、k8sクラスタのすべてのノードが使用できる共同の1つのクラスターとして提示します。
この記事では、インストールが非常に簡単なLonghorn（https://longhorn.io

）という無料のオープンソースのストレージを使用しますが、
OpenEBS、Porworx、StorageOSといったほかのストレージも利用できます。同じ機能が提供されています。
CNCF Incubatingプロジェクトであるrook Cephも検討する価値があるでしょう。 この種のハイエンドとしては、
NetAppやPureStorageといったエンタープライズ級のストレージソリューションがあります。

手順

「要約」セクションでは、1回にすべてをインストールしました。
インストールと検証の手順の説明は、付録Bをご覧ください。

Kubernetesストレージ

まずは、コンテナとストレージ全般について、またIRISがどこに適合するのかについて説明しましょう。

デフォルトでは、コンテナ内のすべてのデータは揮発性であるため、
コンテナが停止すればデータは消失します。 Dockerでは、ボリュームの概念を使用することができるため、
基本的に、ホストOSのディレクトリをコンテナに公開することができます。

```
docker run --detach
  --publish 52773:52773
  --volume /data/dur:/dur
  --env ISC_DATA_DIRECTORY=/dur/iconfig
  --name iris21 --init intersystems/iris:2020.3.0.221.0
```

上記の例では、IRISコンテナを起動し、host-localの「/data/dur」ディレクトリを、「/dur」マウントポイントのコンテナからアクセスできるようにしています。 つまり、コンテナがこのディレクトリに何かを格納している場合、それは保持され、コンテナの次回起動時に使用できるようになります。

IRIS側では、ISCDATADIRECTORYを指定することで、IRISに、コンテナの再起動時に損失してはいけないすべてのデータを特定のディレクトリに格納するように指示することができます。 ドキュメントの「Durable SYS」というIRISの機能をご覧ください（https://docs.intersystems.com/irisforhealthlatestj/csp/docbook/Doc.View.cls?KEY=ADOCK#ADOCKirisdurablerunning）。

Kubernetesでの構文は異なりますが、概念は同じです。

IRISの基本的なKubernetes Deploymentは以下のようになります。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iris
spec:
  selector:
    matchLabels:
      app: iris
  strategy:
    type: Recreate
  replicas: 1
  template:
    metadata:
      labels:
        app: iris
    spec:
```

```
      containers:
      - image: store/intersystems/iris-community:2020.4.0.524.0
        name: iris
        env:
        - name: ISC_DATA_DIRECTORY
          value: /external/iris
        ports:
        - containerPort: 52773
          name: smp-http
        volumeMounts:
        - name: iris-external-sys
          mountPath: /external
      volumes:
      - name: iris-external-sys
        persistentVolumeClaim:
          claimName: iris-pvc
```

上記のデプロイ仕様では、「volumes」の部分にストレージボリュームがリストされています。 このボリュームには、「iris-pvc」などのpersistentVolumeClaimを介して、コンテナの外部からアクセスできます。
volumeMountsは、このボリュームをコンテナ内に公開します。 「iris-external-sys」は、ボリュームマウントを特定のボリュームに関連付ける識別子です。 実際には複数のボリュームが存在する可能性があり、ほかのボリュームと区別するためにこの名前が使用されるわけですから、「steve」と名付けることも可能です。

すでにお馴染みのISCDATADIRECTORYは、IRISが特定のマウントポイントを使用して、コンテナの再起動後も存続する必要のあるすべてのデータを格納するように指示する環境変数です。

では、persistentVolumeClaimのiris-pvcを見てみましょう。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: iris-pvc
spec:
  storageClassName: longhorn
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

かなりわかりやすいと思います。 「longhorn」ストレージクラスを使って、1つのノードでのみ読み取り/書き込みとしてマウント可能な、10ギガバイトを要求しています。

ここで重要なのは、storageClassName: longhornです。

AKSクラスタで利用できるストレージクラスを確認してみましょう。

```
kubectl get StorageClass
NAME                            PROVISIONER                     RECLAIMPOLICY   VOLU
MEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
azurefile                       kubernetes.io/azure-
file        Delete          Immediate           true                   10d
```

```
azurefile-premium               kubernetes.io/azure-
file        Delete          Immediate           true                   10d
default (default)               kubernetes.io/azure-
disk        Delete          Immediate           true                   10d
longhorn                        driver.longhorn.io              Delete          Imme
diate           true                    10d
managed-premium                 kubernetes.io/azure-
disk        Delete          Immediate           true                   10d
```

Azureのストレージクラスはほとんどありませんが、これらは、デフォルトでインストールされたクラスと、以下の一番最初のコマンドの一部でインストールしたLonghornのクラスです。

```
kubectl apply -f https://raw.githubusercontent.com/longhorn/longhorn/master/deploy/lo
nghorn.yaml
```

永続ボリュームクレームの定義に含まれる#storageClassName:
longhorn
をコメントアウトすると、現在「default」としてマークされているストレージクラスが使用されます。これは、通常のAzureディスクです。

分散ストレージが必要な理由を説明するために、この記事の始めに説明した、longhornストレージを使用しない「カオスエンジニアリング」実験をもう一度行ってみましょう。
最初の2つのシナリオ（IRISの停止とポッドの削除）は正常に完了し、システムは稼働状態に回復します。
ノードをドレインまたは強制終了しようとすると、システムは障害状態になります。

```
#forcefully drain the node
kubectl drain aks-agentpool-71521505-vmss000001 --delete-local-data --ignore-
daemonsets

kubectl describe pods
...
  Type     Reason            Age                  From               Message
  ----     ------            ----                 ----               -------
  Warning  FailedScheduling  57s (x9 over 2m41s)  default-scheduler  0/3 nodes are av
ailable: 1 node(s) were unschedulable, 2 node(s) had volume node affinity conflict.
```

基本的に、KubernetesはクラスタのIRISポッドを再起動しようとしますが、最初に起動されていたノードは利用できないため、ほかの2つのノードに「ボリュームノードアフィニティの競合」が発生しています。 このストレージタイプでは、基本的にボリュームはノードホストで使用可能なディスクに関連付けられているため、最初に作成されたノードでしか使用できないのです。

ストレージクラスにlonghornを使用すると、「強制ドレイン」と「ノードの強制終了」の2つの実験は正常に完了し、間もなくしてIRISポッドの動作が再開します。 これを実現するために、Longhornは3ノードクラスタをで使用可能なストレージを制御し、3つのすべてのノードにデータを複製しています。
ノードの1つが永久的に使用不可になると、Longhornは迅速にクラスタストレージを修復します。 「ノードの強制終了」シナリオでは、IRISポッドはほかの2つのボリュームレプリカを使ってすぐに別のノードで再起動されます。 するとAKSは失われたノードに置き換わる新しいノードをプロビジョニングし、準備ができたら、Longhornがそのノードに必要なデータを再構築します。
すべては自動的に行われるため、ユーザーが手を入れる必要はありません。

図3 置換されたノードにボリュームレプリカを再構築するLonghorn

k8sデプロイについて

デプロイの他の側面をいくつか見てみましょう。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iris
spec:
  selector:
    matchLabels:
      app: iris
  strategy:
    type: Recreate
  replicas: 1
  template:
    metadata:
      labels:
        app: iris
    spec:
      containers:
      - image: store/intersystems/iris-community:2020.4.0.524.0
        name: iris
        env:
        - name: ISC_DATA_DIRECTORY
          value: /external/iris
        - name: ISC_CPF_MERGE_FILE
          value: /external/merge/merge.cpf
        ports:
        - containerPort: 52773
          name: smp-http
        volumeMounts:
        - name: iris-external-sys
          mountPath: /external
        - name: cpf-merge
          mountPath: /external/merge
        livenessProbe:
          initialDelaySeconds: 25
          periodSeconds: 10
          exec:
            command:
            - /bin/sh
            - -c
            - "iris qlist iris | grep running"
      volumes:
      - name: iris-external-sys
        persistentVolumeClaim:
          claimName: iris-pvc
      - name: cpf-merge
        configMap:
          name: iris-cpf-merge
```

strategy: Recreateとreplicas:
1では、Kubernetesに、常にIRISポッドの1つのインスタンスのみを実行し続けることを指示しています。
これが「ポッドの削除」シナリオを処理する部分です。

livenessProbe
のセクションでは、IRISが常時、コンテナ内で稼働し、「IRIS停止」シナリオを処理できるようにしています。
initialDelaySecondsは、IRISが起動するまでの猶予期間です。

IRISがデプロイを起動するのにかなりの時間が掛かっている場合は、これを増やすと良いでしょう。

IRISの**CPF**
MERGE機能を使用すると、コンテナの起動時に、iris.cpf構成ファイルのコンテンツを変更することができます。
関連するドキュメントについては、
https://docs.intersystems.com/irisforhealthlatestj/csp/docbook/DocBook.UI.Page.cls?KEY=RACScpf#RACScpfeditmerge をご覧ください。
この例では、Kubernetesの構成図を使って、マージファイルのコンテンツを管理しています（
https://github.com/antonum/ha-iris-k8s/blob/main/iris-cpf-merge.yaml
）。ここでは、IRISインスタンスが使用するグローバルバッファとgmheapの値を調整していますが、iris.cpfファイルにあるものはすべて調整できます。
デフォルトのIRISパスワードでさえも、CPFマージファイルの「PasswordHash」フィールドで変更可能です。
詳細については、
https://docs.intersystems.com/irisforhealthlatestj/csp/docbook/Doc.View.cls?KEY=ADOCK#ADOCKirisimagespasswordauth をご覧ください。

永続ボリュームクレーム（https://github.com/antonum/ha-iris-k8s/blob/main/iris-pvc.yaml
）デプロイ（https://github.com/antonum/ha-iris-k8s/blob/main/iris-deployment.yaml
）とCPFマージコンテンツを使った構成図（https://github.com/antonum/ha-iris-k8s/blob/main/iris-cpf-merge.yaml
）のほか
に、デプロイには
、IRISデプロイをパブリックインタ
ーネットに公開するサービスが必要です（https://github.com/antonum/ha-iris-k8s/blob/main/iris-svc.yaml）。

```
kubectl get svc
NAME         TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)           AGE
iris-svc     LoadBalancer   10.0.18.169   40.88.123.45   52773:31589/TCP   3d1h
kubernetes   ClusterIP      10.0.0.1      <none>          443/TCP           10d
```

iris-svcの外部IPは、http://40.88.123.45:52773/csp/sys/%25CSP.Portal.Home.zen
を介してIRIS管理ポータルにアクセスするために使用できます。 デフォルトのパスワードは「SYS」です。

ストレージのバックアップ/復元とスケーリング

Longhornには、ボリュームの構成と管理を行えるウェブベースのUIがあります。

kubectlを使用して、ポッドや実行中のlonghorn-uiコンポーネントを特定し、ポート転送を確立できます。

```
kubectl -n longhorn-system get pods
# longhorn-ui????ID??????????

kubectl port-forward longhorn-ui-df95bdf85-gpnjv 9000:8000 -n longhorn-system
```

Longhorn UIは、http://localhost:9000で利用できるようになります。

図4 LonghornのUI

Kubernetesコンテナストレージのほとんどのソリューションでは、高可用性のほか、バックアップ、スナップショット、および復元のための便利なオプションが用意されています。
詳細は実装によって異なりますが、一般的には、バックアップをVolumeSnapshotに関連付ける方法です。
この方法はLonghornでも利用できます。
使用しているKubernetesのバージョンとプロバイダーによっては、ボリュームスナップショット機能（
https://github.com/kubernetes-csi/external-snapshotter）もインストールする必要があります。

そのようなボリュームショットの例には、「iris-volume-snapshot.yaml」が挙げられます。
使用する前に、バックアップを、LonghornのS3バケットまたはNFSボリュームに構成する必要があります。
https://longhorn.io/docs/1.0.1/snapshots-and-backups/backup-and-restore/...

```
# IRIS??????????????????????????????
kubectl apply -f iris-volume-snapshot.yaml
```

IRISでは、バックアップ/スナップショットを取得する前にExternal Freezeを実行し、後でThawを実行することをお勧めします。
詳細については、
https://docs.intersystems.com/irisforhealthlatestj/csp/documatic/%25CSP.Documatic.cls?LIBRARY=%25SYS&CLASSNAME=Backup.General#ExternalFreezeをご覧ください。

IRISボリュームのサイズを増やすには、IRISが使用する、永続ボリュームクレーム（iris-pvc.yamlファイル）のストレージリクエストを調整します。

```
...
  resources:
    requests:
      storage: 10Gi #change this value to required
```

そして、pvcの仕様を再適用します。 Longhornは、実行中のポッドにボリュームが接続されている場合、この変更を実際に適用することはできません。
そのため、ボリュームサイズが増えるように、デプロイでレプリカ数を一時的にゼロに変更します。

高可用性 - 概要

この記事の冒頭で、高可用性の基準を設定しました。 このアーキテクチャでは、次のようにそれを実現します。

障害の領域	**自動的に緩和処置を行う機能**
コンテナ/VM内のIRISインスタンス。 IRIS - レベル障害	IRISが機能停止となった場合、デプロイのLiveness Probeによってコンテナが再起動されます。
ポッド/コンテナの障害	デプロイによってポッドが再作成されます。
個々のクラスタノードの一時的な使用不能。 アベイラビリティーゾーンがオフラインになる場合などが挙げられます。	デプロイによって別のノードにポッドが再作成されます。Longhornによって、別のノードでデータが使用できるよ
個々のクラスタノードまたはディスクの永久的障害。	上記と同様かつ、k8sクラスタオートスケーラーによってードが新しいノードに置き換えられます。Longhornによって、新しいノードにデータが再構築され

ゾンビやその他の考慮事項

Dockerコンテナでの IRISの実行を理解している方であれば、「--init」フラグを使用したことがあるでしょう。

```
docker run --rm -p 52773:52773 --init store/intersystems/iris-community:2020.4.0.524.0
```

このフラグは、「ゾンビプロセス」の形成を防止することを目的としています。

Kubernetesでは、「shareProcessNamespace:
true」を使用する（セキュリティの考慮事項が適用されます）か、コンテナで「tini」を使用することができます。
以下に、tiniを使用したDockerfileの例を示します。

```
FROM iris-community:2020.4.0.524.0
...
# Tini??????
USER root
ENV TINI_VERSION v0.19.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini /tini
RUN chmod +x /tini
USER irisowner
ENTRYPOINT ["/tini", "--", "/iris-main"]
```

2021年以降、すべてのInterSystemsが提供するコンテナイメージには、デフォルトでtiniが含まれています。

いくつかのパラメーターを調整することで、「ノードの強制ドレイン/ノードの強制終了」シナリオのフェイルオーバー時間をさらに短縮することができます。

Longhornのポッド削除ポリシー（https://longhorn.io/docs/1.1.0/references/settings/#pod-deletion-policy-when-node-is-down）およびkubernetes
taintベースのエビクション機能（https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/#taint-based-evictions）をご覧ください。

免責事項

InterSystemsに勤務する者として、
この内容を記載しておく必要があります。
この記事では、分散型Kubernetesブロックストレージの一例としてLonghornを使用しています。 InterSystemsは、個々のストレージソリューションや製品を検証ないしは公式なサポート声明を発行していません。 特定のストレージソリューションがニーズに適合しているかどうかは、ユーザー自身がテストして検証する必要があります。

分散ストレージには、ノードローカルストレージとは大きく異なるパフォーマンス特性も備わっています。 特に書き込み操作の場合、データが永続化された状態とみなされるには、データを複数の場所に書き込む必要があります。
必ずワークロードをテストし、CSIドライバが提供する特定の動作とオプションを理解するようにしてください。

InterSystemsでは基本的に、Longhornなどの特定のストレージjソリューションを検証あるいは承認していません。これは、個々のHDDブランドやサーバーハードウェアメーカーを検証しないのと同じです。
個人的には、Longhornは扱
いやすいと感じています。プロジェクトのGitHubペー
ジ（https://github.com/longhorn/longhorn
）では、その開発チームは積極的に応答し、よく助けていただいています。

まとめ

Kubernetesエコシステムは、過去数年間で大きな進化を遂げました。今日では、分散ブロックストレージソリューションを使用することで、IRISインスタンスやクラスタノード、さらにはアベイラビリティーゾーンの障害を維持できる高可用性構成を構築できるようにもなっています。

計算とストレージの高可用性をKubernetesコンポーネントにアウトソースできるため、従来のIRISミラーリングに比べ、システムの構成と管理が大幅に単純化しています。 ただしこの構成では、ミラーリング構成ほどのRTOとストレージレベルのパフォーマンスは得られないことがあります。

この記事では、Azure AKSをマネージドKubernetesとLonghorn分散ストレージシステムとして使用し、可用性の

高いIRIS構成を構築しました。 ほかにも、AWS EKS、マネージドK8s向けのGoogle Kubernetes Engine、StorageOS、Portworx、OpenEBSなどの様々な分散コンテナストレージを探ってみると良いでしょう。エンタープライズ級のストレージソリューションには、NetApp、PureStorage、Dell EMCなどもあります。

付録A: クラウドでのKubernetesクラスタの作成

パブリッククラウドプロバイダーが提供するマネージドKubernetesサービスを使うと、このセットアップに必要なk8sクラスタを簡単に作成できます。
この記事で説明したデプロイには、AzureのAKSデフォルト構成をそのまますぐに使用することができます。

3ノードで新しいAKSクラスタを作成します。 それ以外はすべてデフォルトのままにします。

図5 AKSクラスタの作成

ローカルコンピュータにkubectlをインストールします。https://kubernetes.io/docs/tasks/tools/install-kubectl/

ローカルkubectlにASKクラスタを登録します。

図6 kubectlにAKSクラスタを登録

登録が済んだら、この記事の最初に戻って、longhornとIRISデプロイをインストールします。

AWS EKSでのインストールは、もう少し複雑です。 ノードグループのすべてのインスタンスにopen-iscsiがインストトールされていることを確認する必要があります。

```
sudo yum install iscsi-initiator-utils -y
```

GKEでのLonghornのインストールには追加の手順
があります。https://longhorn.io/docs/1.0.1/advanced-resources/os-distro-specific/csi-on-gke/をご覧ください。

付録B: インストール手順

手順1 - Kubernetesクラスタとkubectl

3ノードk8sクラスタが必要です。 Azureでのクラスタの構成方法は付録Aをご覧ください。

```
$ kubectl get nodes
NAME                                STATUS   ROLES   AGE   VERSION
aks-agentpool-29845772-vmss000000   Ready    agent   10d   v1.18.10
aks-agentpool-29845772-vmss000001   Ready    agent   10d   v1.18.10
aks-agentpool-29845772-vmss000002   Ready    agent   10d   v1.18.10
```

手順2 - Longhornをインストールする

```
kubectl apply -f https://raw.githubusercontent.com/longhorn/longhorn/master/deploy/longhorn.yaml
```

「longhorn-system」ネームスペースのすべてのポッドが実行状態であることを確認してください。
これには数分かかる場合があります。

```
$ kubectl get pods -n longhorn-system
```

```
NAME                                  READY   STATUS    RESTARTS   AGE
csi-attacher-74db7cf6d9-jgdxq         1/1     Running   0          10d
csi-attacher-74db7cf6d9-l99fs         1/1     Running   1          11d
...
longhorn-manager-flljf                1/1     Running   2          11d
longhorn-manager-x76n2                1/1     Running   1          11d
longhorn-ui-df95bdf85-gpnjv           1/1     Running   0          11d
```

詳細とトラブルシューティングについては、Longhornインストールガイド（https://longhorn.io/docs/1.1.0/deploy/install/install-with-kubectl）をご覧ください。

手順3 - GitHubリポジトリのクローンを作成する

```
$ git clone https://github.com/antonum/ha-iris-k8s.git
$ cd ha-iris-k8s
$ ls
LICENSE                 iris-deployment.yaml     iris-volume-snapshot.yaml
README.md               iris-pvc.yaml            longhorn-aws-secret.yaml
iris-cpf-merge.yaml     iris-svc.yaml            tldr.yaml
```

手順4 - コンポーネントを1つずつデプロイして検証する

tldr.yamlファイルには、デプロイに必要なすべてのコンポーネントが1つのバンドルとして含まれています。ここでは、コンポーネントを1つずつインストールし、それぞれのセットアップを個別に検証します。

```
# ???tldr.yaml??????????????????
$ kubectl delete -f https://github.com/antonum/ha-iris-k8s/raw/main/tldr.yaml

# ???????????????
$ kubectl apply -f iris-pvc.yaml
persistentvolumeclaim/iris-pvc created

$ kubectl get pvc
NAME       STATUS   VOLUME                                     CAPACITY   ACCESS MODE
S   STORAGECLASS   AGE
iris-pvc   Bound    pvc-
fbfaf5cf-7a75-4073-862e-09f8fd190e49   10Gi       RWO            longhorn       10s

# ????????
$ kubectl apply -f iris-cpf-merge.yaml

$ kubectl describe cm iris-cpf-merge
Name:         iris-cpf-merge
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
merge.cpf:
----
[config]
globals=0,0,800,0,0,0
gmheap=256000
Events:  <none>
```

```
# iris??????????
$  kubectl apply -f iris-deployment.yaml
deployment.apps/iris created

$ kubectl get pods
NAME                    READY   STATUS              RESTARTS   AGE
iris-65dcfd9f97-v2rwn   0/1     ContainerCreating   0          11s

# ??????????? ????????????????????????????

$ kubectl exec -it iris-65dcfd9f97-v2rwn   -- bash

irisowner@iris-65dcfd9f97-v2rwn:~$ iris session iris
Node: iris-65dcfd9f97-v2rwn, Instance: IRIS

USER>w $zv
IRIS for UNIX (Ubuntu Server LTS for x86-64 Containers) 2020.4 (Build 524U) Thu Oct 2
2 2020 13:04:25 EDT
# h<enter>?IRIS??????
# exit<enter>???????

# IRIS???????????????
$ kubectl logs iris-65dcfd9f97-v2rwn
...
[INFO] ...started InterSystems IRIS instance IRIS
01/18/21-23:09:11:312 (1173) 0 [Utility.Event] Private webserver started on 52773
01/18/21-23:09:11:312 (1173) 0 [Utility.Event] Processing Shadows section (this syste
m as shadow)
01/18/21-23:09:11:321 (1173) 0 [Utility.Event] Processing Monitor section
01/18/21-23:09:11:381 (1323) 0 [Utility.Event] Starting TASKMGR
01/18/21-23:09:11:392 (1324) 0 [Utility.Event] [SYSTEM MONITOR] System Monitor starte
d in %SYS
01/18/21-23:09:11:399 (1173) 0 [Utility.Event] Shard license: 0
01/18/21-23:09:11:778 (1162) 0 [Database.SparseDBExpansion] Expanding capacity of spa
rse database /external/iris/mgr/iristemp/ by 10 MB.

# iris??????????
$ kubectl apply -f iris-svc.yaml
service/iris-svc created

$ kubectl get svc
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)           AGE
iris-svc     LoadBalancer   10.0.214.236   20.62.241.89   52773:30128/TCP   15s
```

手順5 - 管理ポータルにアクセスする

最後に、サービスの外部IP（http://20.62.241.89:52773/csp/sys/%25CSP.Portal.Home.zen
）を使って、IRISの管理ポータルに接続します。ユーザー名は「SYSTEM」、パスワードは「SYS」です。
初回ログイン時にパスワードの変更が求められます。

#AWS #Azure #Google Cloud Platform (GCP) #Kubernetes #Mirroring #クラウド #バックアップ
#フェイルオーバー #高可用性 #InterSystems IRIS
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/kubernetes%E3%81%AB%E3%81%8A%E3%81%91%E3%82%8B%E3%83%9F%E3%83%A9%E3%83%BC%E3%83%AA%E3%83%B3%E3%82%B0%E3%82%92%E4%BD%BF%E7%94%A8%E3%81%97%E3%81%AA%E3%81%84%E9%AB%98%E5%8F%AF%E7%94%A8%E6%80%A7iris%E3%83%87%E3%83%97%E3%83%AD%E3%82%A4