Artigo
Angelo Bruno Braga · Fev. 25, 2022 27min de leitura

Open Exchange

Implantações IRIS com Alta disponibilidade no Kubernetes sem utilizar espelhamento

Neste artigo iremos construir uma configuração IRIS de alta disponibilidade utilizando implantações Kubernetes com armazenamento persistente distribuído substituindo o "tradicional" espelhamento IRIS. Esta implantação será capaz de tolerar falhas relacionadas a infraestrutura como falhas em nós, armazenamento e de Zonas de Disponibilidade. A abordagem descrita reduz muito a complexidade da implantação em detrimento um objetivo de tempo de recuperação (RTO) ligeiramente estendido.

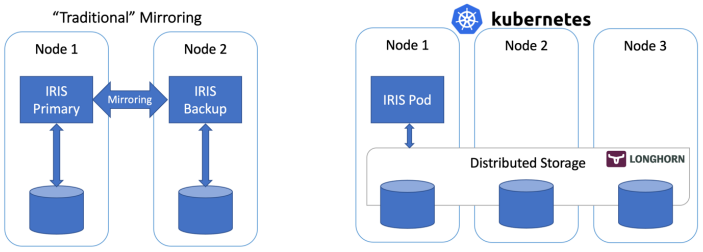

Figura 1 - Espelhamento Tradicional x Kubernetes com Armazenamento Distribuído

Todos os códigos fonte deste artigo estão disponíveis em https://github.com/antonum/ha-iris-k8s

TL;DR

Assumindo que você possui um cluster com 3 nós e também uma certa familiaridade com o Kubernetes – vá em frente:

```
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
```

Se você não tem certeza sobre o que as duas linhas acima fazem ou não possui o sistema onde executá-las - pule para a seção " Requisitos de Alta Disponibilidade". Iremos explicar tudo detalhadamente conforme evoluirmos no artigo.

Esta primeira linha instala o Longhorn - armazenamento distribuído Kubernetes de código aberto. A segunda instala o InterSystems IRIS, utilizando um volume baseado no Longhorn para o Durable SYS.

Aguarde até que todos os pods atinjam o estado de "em execução". kubectl get pods -A

Agora você deve ser capaz de acessar o portal de administração do IRIS em http://<IP Público do IRIS>:52773/csp/sys/%25CSP.Portal.Home.zen (a senha padrão é 'SYS') e a linha de comando do IRIS através de:

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

Simule a Falha

Agora pode começar a bagunçar as coisas. Mas, antes de fazê-lo, tente adicionar alguns dados na base de dados para se certificar que estarão lá quando o IRIS voltar a ficar disponível.

```
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"
```

Nossa "engenharia do caos" inicia aqui:

```
# Parar o IRIS - Contêiner será reiniciado automaticamente
kubectl exec -it iris-6d8896d584-8lzn5 -- iris stop iris quietly

# Deletar o  pod - O Pod será recriado
kubectl delete pod iris-6d8896d584-8lzn5

# "Drenar à força" o nó que está servindo o pod IRIS - O Pod seria recriado em outro
nó
kubectl drain aks-agentpool-29845772-vmss000001 --delete-local-data --ignore-
daemonsets --force

# Deletar o nó - O Pod seria recriado em outro nó
# bem... você não pode realmente fazê-
lo com o kubectl. Localize a instância ou a VM e a destrua.
# se você possuir acesso à maquina - desligue a força ou desconecte o cabo de rede. E
stou falando sério!
```

Requisitos de Alta Disponibilidade

Estamos construindo um sistema que possa tolerar uma falha das seguintes:

- Instância IRIS dentro de um contêiner/VM. Falha a nível do IRIS.
- Falha no Pod/Contêiner.
- Indisponibilidade temporária do nó de cluster individual. Um bom exemplo seria quando uma Zona de Disponibilidade fica temporariamente fora do ar.
- Falha permanente do nó de cluster individual ou disco.

Basicamente os cenários que executamos na seção ˜ Simule a falha".

Se alguma destas falhas ocorrer, o sistema deverá se recuperar sem que necessite de nenhum envolvimento humano e sem que ocorra nenhuma perda de dados.. Tecnicamente existem limites do que a persistência de dados garante. O IRIS por si só provê com base no Ciclo do Journal e no uso de transações em uma aplicação: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.cls?KEY=GCDIjournal#GCDIjournalwritecycle De qualquer forma, estamos falando de menos de dois segundos de objetivo de ponto de recuperação (RPO).

Outros componentes do sistema (Serviço de APIs Kubernetes, base de dados etcd, serviço de Balanceamento de Carga, DNS e outros) estão fora do escopo e são gerenciados tipicamente pelo Serviço Gerenciado Kubernetes como o Azure AKS ou AWS EKS, então assumimos que eles já estão em alta disponibilidade.

Outra forma de se ver – somos responsáveis por lidar com falhas individuais de componentes e armazenamento, assumindo que o resto é tratado pelo provedor de infraestrutura/nuvem.

Arquitetura

Quando se trata de alta disponibilidade para o InterSystems IRIS, a recomendação tradicional é a utilização de Espelhamento. Utilizando o Espelhamento você terá duas instâncias ligadas do IRIS replicando de forma síncrona os dados. Cada nó mantém uma cópia completa da base de dados e, se o nó principal falhar, os usuários reconectam no nó Backup. Essencialmente, com a abordagem de uso do Espelhamento, o IRIS é responsável pela redundância tanto de computação quanto de armazenamento.

Com os servidores de espelhamento implantados em zonas de disponibilidade distintas o espelhamento provê a redundância necessária tanto para falhas de computação quanto para falhas de armazenamento e permite o excelente objetivo de tempo de recuperação (RTO - tempo necessário para que um sistema se recupere após uma falha) de apenas poucos segundos. Você pode encontrar o modelo de implantação para o IRIS Espelhado na Nuvem AWS aqui:
https://community.intersystems.com/post/intersystems-iris-deployment%C2%A0guide-aws%C2%A0using-cloudformation-template

O lado menos bonito do espelhamento é a complexidade de configurá-lo, realizando procedimentos de backups e restore e lidando com a falta de replicação para configurações de segurança e arquivos locais que não os de bases de dados.

Orquestradores de contêineres como o Kubernetes (espere, estamos em 2021… exitem outros?!) proveem uma redundância computacional através da implantação de objetos, automaticamente reiniciando o Pod/Contêiner IRIS no caso de falha. É por isso que você vê apenas um nó IRIS executando no diagrama de arquitetura Kubernetes. Ao invés de manter um segundo nó de IRIS executando nós terceirizamos a disponibilidade de computação para o Kubernetes. O Kubernetes se certificará que o pod IRIS pode ser recriado no caso de falha do pod original por qualquer motivo.

Figura 2 Cenario de Failover

Tudo bem até agora… Se o nó do IRIS falhar, o Kubernetes apenas cria um novo. Dependendo do seu cluster, ele leva algo entre 10 a 90 segundos para trazer o IRIS de volta ao ar após uma falha de computação. É um passo atrás comparado com apenas alguns segundos no espelhamento mas, se é algo que você pode tolerar no caso de um evento indesejado, a recompensa é a grande redução da complexidade. Sem configuração de espelhamento e nem configurações de segurança e replicações de arquivos com que se preocupar.

Sinceramente, se você se logar em um contêiner, executando o IRIS no Kubernetes, você nem mesmo perceberá que está executando em um ambiente de alta disponibilidade. Tudo parece e se comporta como uma implantação de uma instância individual de IRIS.

Espere, e quanto ao armazenamento? Estamos lidando com um banco de dados … Seja qual for o cenário de falha que possamos imaginar, nosso sistema deverá cuidar da persistência dos dados também. O Espelhamento depende da computação, local no nó IRIS. Se o nó morre ou fica temporariamente indisponível – o armazenamento para o nó também o fica. É por isso que na configuração de espelhamento o IRIS cuida da replicação das bases de dados no nível do IRIS.

Precisamos de um armazenamento que possa não só preservar o estado da base de dados na reinicialização do contêiner mas também possa prover redundância em casos como a queda do nó ou de um segmento inteiro da rede (Zona de Disponibilidade). A apenas alguns anos atrás não existia uma resposta fácil para isso.Como você pode supor a partir do diagrama acima – temos uma baita resposta agora. É chamada de **armazenamento distribuído de contêineres**.

O armazenamento distribuído abstrai os volumes de hospedeiros subjacentes e os apresenta como um armazenamento conjunto disponível para todos os nós do cluster k8s. Nós utilizamos o Longhorn https://longhorn.io neste artigo; é grátis, de código aberto e bem fácil de instalar. Mas, você também pode verificar outros como o OpenEBS, Portworx e StorageOS que devem disponibilizar as mesmas funcionalidades. Rook Ceph é outro projeto de incubação CNCF a se considerar. No outro lado do espectro existem soluções de armazenamento de nível empresarial como o NetApp, PureStorage e outros.

Guia Passo a Passo

Na seção TL;DR nós instalamos tudo de uma vez. O Apêndice B lhe guiará através da instalação passo a passo e dos procedimentos de validação.

Armazenamento Kubernetes

Vamos voltar um pouco por um segundo e falar sobre contêineres e armazenamento em geral e como o IRIS se encaixa no cenário.

Por padrão todos os dados dentro do contêiner são efêmeros. Quando o contêiner morre, o dado desaparece. No Docker, você pode utilizar o conceito de volumes. Essencialmente isto permite que você exponha o diretório de seu SO hospedeiro para o contêiner.

```
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
```

No exemplo acima estamos iniciando o contêiner IRIS e fazendo com que o diretório local do hospedeiro ‘ /data/dur ’ fique acessível no ponto de montagem ‘ /dur ’ do contêiner. Desta forma, se o contêiner estiver armazenando qualquer coisa neste diretório, ela será preservada e disponível para utilização quando o próximo contêiner iniciar.

Do lado IRIS das coisas, podemos instruir o IRIS para armazenar todos os dados que devem sobreviver ao reinicio do contêiner em um diretório determinado especificando ISCDATADIRECTORY. Durable SYS é o nome da funcionalidade do IRIS que você pode precisar dar uma olhada na documentação https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.cls?KEY=ADOCK#ADOCKirisdurablerunning

No Kubernetes a sintaxe é diferente mas os conceitos são os mesmos.

Aqui está a Implantação Kubernetes básica para IRIS.

```
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
```

Na especificação de implantação acima, a parte ‘ volumes ’ lista os volumes de armazenamento. Eles podem estar disponíveis fora do contêiner através do Requisição de Volume Persistente (PersistentVolumeClaim) como ‘ iris-pvc ’ . O 'volumeMounts' expõe este volume dentro do contêiner. ‘ iris-external-sys ’ é o identificador que amarra a montagem do volume ao volume específico. Na verdade, podemos ter múltiplos volumes e este nome é utilizado apenas para distinguir um de outro. Você pode chamá-lo de ‘ steve ’ se quiser.

A variável de ambiente ISCDATADIRECTORY, já familiar, instrui o IRIS a utilizar um ponto de montagem específico para armazenar todos os dados que precisam sobreviver ao reinício do contêiner.

Agora vamos dar uma olhada na Requisição de Volume Persistente iris-pvc.

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

Bastante direto. Requisitando 10 gigabytes, montado como Read/Write em apenas um nó, utilizando a classe de armazenamento de ‘ longhorn ’ .

Aquela storageClassName: longhorn é de fato crítica aqui.

Vamos olhar quais classes de armazenamento estão disponíveis no meu cluster AKS:

```
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
```

Existem poucas classes de armazenamento do Azure, instaladas por padrão e uma do Longhorn que instalamos como parte de nosso primeiro comando:

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

Se você comentar #storageClassName: longhorn na definição da Requisição de Volume Persistente, será utilizada a classe de armazenamento que estiver marcada como " default " que é o disco regular do Azure.

Para ilustrar porquê precisamos de armazenamento distribuído vamos repetir o experimento da " engenharia do caos " que descrevemos no início deste artigo sem o armazenamento longhorn. Os dois primeiros cenários (parar o IRIS e deletar o Pod) deveriam completar com sucesso e os sistemas deveriam se recuperar ao estado operacional. Ao tentar tanto drenar ou destruir o nó deveria deixar o sistema em estado de falha.

```
#drenar o nó a força
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.
```

Essencialmente, o Kubernetes tentará reiniciar o pod IRIS pod no cluster mas, o nó onde ele iniciou originalmente não está disponível e os outros dois nós apresentam " conflito de afinidade de nó de volume " . Com este tipo de armazenamento o volume só está disponível no nó onde ele foi originalmente criado, visto que ele é basicamente amarrado ao disco disponível no nó hospedeiro.

Com o longhorn como classe de armazenamento, tanto o experimento " drenar a força " quanto o " destruir nó " funcionam com sucesso e o pod IRIS retorna a operação brevemente. Para conseguí-lo o Longhorn assume controle dos armazenamentos disponíveis dos 3 nós do cluster e replica os dados através deles. O Longhorn prontamente repara o armazenamento do cluster se um dos nós fica permanentemente indisponível. No nosso cenário " Destruir nó " o pod IRIS é reiniciado em outro nó rapidamente utilizando as replicas nos dois outros volumes remanescentes. Então, o AKS provisiona um novo nó para substituir o nó perdido e assim que ele está pronto, o Longhorn entra em ação e reconstrói os dados necessários no novo nó. Tudo é automático, sem seu envolvimento.

Figura 3 Longhorn reconstruindo a réplica do volume no nó substituído

Mais sobre implantação k8s

Vamos dar uma olhada em alguns outros aspectos de nossa implantação:

```
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, replicas: 1 informa ao Kubernetes que em algum momento ele deve manter uma e exatamente uma instância de do pod IRIS executando. Isto é o que dá conta do nosso cenário " deletar pod " .

A seção **livenessProbe** garante que o IRIS está sempre ativo no contêiner e trata o cenário " IRIS está fora " . initialDelaySeconds garante algum tempo para que o IRIS inicie. Você pode querer aumentá-lo se o IRIS estiver levando um tempo considerável para iniciar em sua implementação.

CPF MERGE funcionalidade do IRIS que lhe permite modificar o conteúdo do arquivo de configuração iris.cpf durante a inicialização do contêiner. Veja https://docs.intersystems.com/irisforhealthlatest/csp/docbook/DocBook.UI.Page.cls?KEY=RACScpf#RACScpfeditmerge para a documentação referente a ela. Neste exemplo estou utilizando o Kubernetes Config Map para gerenciar o conteúdo do arquivo mesclado: https://github.com/antonum/ha-iris-k8s/blob/main/iris-cpf-merge.yaml Aqui ajustamos os valores para os global buffers e gmheap, utilizados pela instância do IRIS, mais tudo que você pode encontrar no arquivo iris.cpf. Você pode até mesmo alterar a senha padrão do IRIS utilizando o campo ÞasswordHashhò arquivo CPF Merge. Leia mais em: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.cls?KEY=ADOCK#ADOCKirisimagespasswordauth

Além da Requisição de Volume Persistente https://github.com/antonum/ha-iris-k8s/blob/main/iris-pvc.yaml da implantação https://github.com/antonum/ha-iris-k8s/blob/main/iris-deployment.yaml e ConfigMap com o conteúdo do CPF Merge https://github.com/antonum/ha-iris-k8s/blob/main/iris-cpf-merge.yaml nossa implantação precisa de um serviço que exponha a implantação IRIS para a Internet pública: 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
```

O IP Externo do iris-svc pode ser utilizado para acessar o portal de administração do IRIS através de http://40.88.123.45:52773/csp/sys/%25CSP.Portal.Home.zen. A senha padrão é 'SYS'.

Backup/Restore e Dimensionamento do Armazenamento

Longhorn disponibiliza uma interface web para usuários para configurar e gerenciar volumes.

Identificar o pod, executar o componente de interface para usuários (longhorn-ui) e estabelecer um encaminhamento de portas com kubectl:

```
kubectl -n longhorn-system get pods
# note the longhorn-ui pod id.

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

A interface para usuários Longhorn ficará disponível em http://localhost:9000

Figura 4 Longhorn UI

Além da alta disponibilidade, a maioria das soluções de armazenamento para contêineres disponibilizam opções convenientes para backup, snapshots e restore. Os detalhes são específicos para cada implantação mas a convenção comum é de que o backup é associado ao VolumeSnapshot. Também é assim para o Longhorn. Dependendo da sua versão do Kubernetes e do seu provedor você também pode precisar instalar o volume snapshotter https://github.com/kubernetes-csi/external-snapshotter

`îris-volume-snapshot.yaml`é um exemplo de um snapshot de volume. Antes de utilizá-lo você deve configurar backups ou para um bucket S3 ou para um volume NFS no Longhorn. https://longhorn.io/docs/1.0.1/snapshots-and-backups/backup-and-restore/...

```
# Realizar um backup consistente do volume IRIS
kubectl apply -f iris-volume-snapshot.yaml
```

Para o IRIS é recomendado que você execute o External Freeze antes de realizar o backup/snapshot e então o Thaw depois. Veja os detalhes aqui: https://docs.intersystems.com/irisforhealthlatest/csp/documatic/%25CSP.Documatic.cls?LIBRARY=%25SYS&CLASSNAME=Backup.General#ExternalFreeze

Para aumentar o tamanho do volume IRIS - ajuste a requisição de armazenamento na requisição de volume persistente (arquivo `îris-pvc.yaml`), utilizado pelo IRIS.

```
...
  resources:
    requests:
      storage: 10Gi #altere este valor para o necessário
```

Então, aplique novamente a especificação pvc. O Longhorn não consegue aplicar esta alteração enquanto o volume está conectado ao Pod em execução. Temporariamente altere o contador de réplicas para zero na implantação para que o tamanho do volume possa ser aumentado.

Alta Disponibilidade – Visão Geral

No início deste artigo nós definimos alguns critérios para alta disponibilidade. Aqui está como conseguimos alcançá-los com esta arquitetura:

Domínio de Falha	**Mitigado automaticamente por**
Instância IRIS no contêiner/VM. Falha no nível do IRIS.	Sonda Deployment Liveness reinicia o contêiner no caso fora
Falha de Pod/Contêiner.	Implantação recria o Pod
Indisponibilidade temporária de um nó do cluster individual. Um bom exemplo seria uma Zona de Disponibilidade fora.	Implantação recria o pod em outro nó. O Longhorn torna disponíveis em outro nó.
Falha permanente de um nó de cluster individual ou disco.	Mesmo do anteiror + k8s cluster autoscaler substituindo u danificado por um novo. O Longhorn reconstrói os dados

Zumbis e outras coisas a considerar

Se você estiver familiarizado em executar o IRIS em contêineres Docker, você já deve ter utilizado a flag `--init.`

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

O objetivo desta flag é previnir a formação de processos "zumbis". No Kubernetes, você pode tanto utilizar ‘ shareProcessNamespace: true ’ (considerações de segurança se aplicam) ou em seus próprios contêineres utilizar

tini.` Exemplo de Dockerfile com tini:

```
FROM iris-community:2020.4.0.524.0
...
# Add 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"]
```

Desde 2021, todas as imagens de contêineres disponibilizadas pela InterSystems incluiria tini por padrão.

Você pode posteriormente diminuir o tempo de recuperação para os cenários " Drenar a força o nó/destruir nó " ajustando poucos parâmetros:

Política de Deleção de Pods do Longhorn https://longhorn.io/docs/1.1.0/references/settings/#pod-deletion-policy-when-node-is-down e despejo baseado em taint do kubernetes: https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/#taint-based-evictions

Disclaimer

Como funcionário InterSystems, Eu tenho que colocar isto aqui: *O Longhorn é utilizado neste artigo como um exemplo de Armazenamento em Blocos Distribuído para Kubernetes. A InterSystems não valida e nem emite uma declaração de suporte oficial para soluções ou produtos de armazenamento individual. Você precisa testar e validar se qualquer solução específica de armazenamento atende a suas necessidades.*

Solucões para armazenamento distribuído podem possuir características substancialmente distintas de performance., quando comparadas a armazenamento em nó local. Especialmente para operações de escrita, onde os dados devem ser escritos em múltiplas localidades antes de ser considerado em estado persistente. Certifique-se de testar suas cargas de trabalho e entender o comportamento específico, bem como as opções que seu driver para Interface de Armazenamento de Contêiner (CSI) oferece.

Basicamente, a InterSystems não valida e/ou endossa soluções específicas de armazenamento como o Longhorn da mesma forma que não valida marcas de HDs ou fabricantes de hardware para servidores. Eu pessoalmente achei o Longhorn fácil de se utilizar e seu time de desenvolvimento extremamente responsivo e útil na página do projeto no GitHub. https://github.com/longhorn/longhorn

Conclusão

O ecossistema Kubernetes evoluiu de forma significante nos últimos anos e, com a utilização de soluções de armazenamento em blocos distribuído, você pode agora construir uma configuração de Alta Disponibilidade que pode manter uma instância IRIS, nó de cluster e até mesmo falhas de Zonas de Disponibilidade.

Você pode terceirizar a alta disponibilidade de computação e armazenamento para componentes do Kubernetes, resultando em um sistema significativamente mais simples de se configurar e manter, comparando-se ao espelhamento tradicional do IRIS. Da mesma forma, esta configuração pode não lhe prover o mesmo RTO e nível de performance de armazenamento que uma configuração de Espelhamento.

Neste artigo criamos uma configuração IRIS de alta disponibilidade utilizando o Azure AKS como Kubernetes gerenciado e o sistema de armazenamento distribuído Longhorn. Você pode explorar múltiplas alternativas como AWS EKS, Google Kubernetes Engine para K8s gerenciados, StorageOS, Portworx e OpenEBS para armazenamento distribuído para contêiner ou mesmo soluções de armazenamento de nível empresarial como NetApp, PureStorage, Dell EMC e outras.

Apêndice A. Criando um Cluster Kubernetes na nuvem

Serviço Gerenciado Kubernetes de um dos provedores públicos de nuvem é uma forma fácil de criar um cluster k8s necessário para esta configuração.A configuração padrão do AKS da Azure é pronto para ser utilizado para a implantação descrita neste artigo.

Criar um novo cluster AKS com 3 nós. Deixe todo o resto padrão.

Figura 5 Criar um cluster AKS

Instale o kubectl em seu computador localmente: https://kubernetes.io/docs/tasks/tools/install-kubectl/

Registre seu cluster AKS com o kubectl local

Figura 6 Registre o cluster AKS com kubectl

Depois disto, você pode voltar para o início do artigoe instalar o longhorn e a implantação IRIS.

A instalação no AWS EKS é um pouco mais complicada. Você precisa se certificar que cada instância em seu grupo de nós tem o open-iscsi instalado.

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

Instalar o Longhorn no GKE necessita de um passo extra, descrito aqui: https://longhorn.io/docs/1.0.1/advanced-resources/os-distro-specific/csi-on-gke/

Apêndice B. Instalação Passo a Passo

Passo 1 – Cluster Kubernetes e kubectl

Você precisa um cluster k8s com 3 nós. Apêndice A descreve como conseguir um na Azure.

```
$ 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
```

Passo 2 – Instalar o Longhorn

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

Certifique-se de que todos os pods no namespace ‘ longhorn-system ’ estão no estado de em execução. Isso pode levar alguns minutos.

```
$ 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
```

Consulteo guia de instalação do Longhornpara detalhes e solução de problemas
https://longhorn.io/docs/1.1.0/deploy/install/install-with-kubectl

Passo 3 – Faça um clone do repositório GitHub

```
$ 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
```

Passo 4 – implemente e valide os componentes um a um

o arquivo tldr.yaml contém todos os componentes necessários para a implantação em um pacote. Aqui iremos instalá-los um a um e validar a configuração de cada um deles individualmente.

```
# Se você aplicou o tldr.yaml previamente, apague-o.
$ kubectl delete -f https://github.com/antonum/ha-iris-k8s/raw/main/tldr.yaml

# Criar a Requisição de Volume Persistente
$ 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

# Criar o Mapa de Configuração
$ 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>

# criar a implantação 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

# Anote o nome do pod. Você irá utilizá-lo para conectar ao pod no próximo comando

$ 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> to exit IRIS shell
# exit<enter> to exit pod

# acesse os logs do contêiner 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.

# crie o serviço 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
```

Passo 5 – Acesse o portal de administração

Finalmente – conecte-se ao portal de administração do IRIS, utilizando o IP externo do serviço: http://20.62.241.89:52773/csp/sys/%25CSP.Portal.Home.zen usuário SYSTEM, senha SYS. Será solicitado que você altere no seu primeiro login.

#Alta Disponibilidade #AWS #Azure #Backup #Espelhamento #GCP #Kubernetes #Melhores Práticas #Tolerância a falhas #InterSystems IRIS
Confira o aplicativo relacionado no InterSystems Open Exchange

URL de origem:https://pt.community.intersystems.com/post/implanta%C3%A7%C3%B5es-iris-com-alta-disponibilidade-no-kubernetes-sem-utilizar-espelhamento