Artículo
Ricardo Paiva · 28 ene, 2022 Lectura de 28 min

Open Exchange

Implementación de IRIS con alta disponibilidad en Kubernetes sin mirroring

En este artículo, crearemos una configuración de IRIS con alta disponibilidad utilizando implementaciones en Kubernetes con almacenamiento persistente distribuido en vez del "tradicional" par de *mirror* de IRIS. Esta implementación sería capaz de tolerar fallos relacionados con la infraestructura, por ejemplo, fallos en los nodos, en el almacenamiento y en la Zona de Disponibilidad. El enfoque descrito reduce en gran medida la complejidad de la implementación, a costa de un Tiempo Objetivo de Recuperación (RTO, *Recovery Time Objective*) ligeramente mayor.

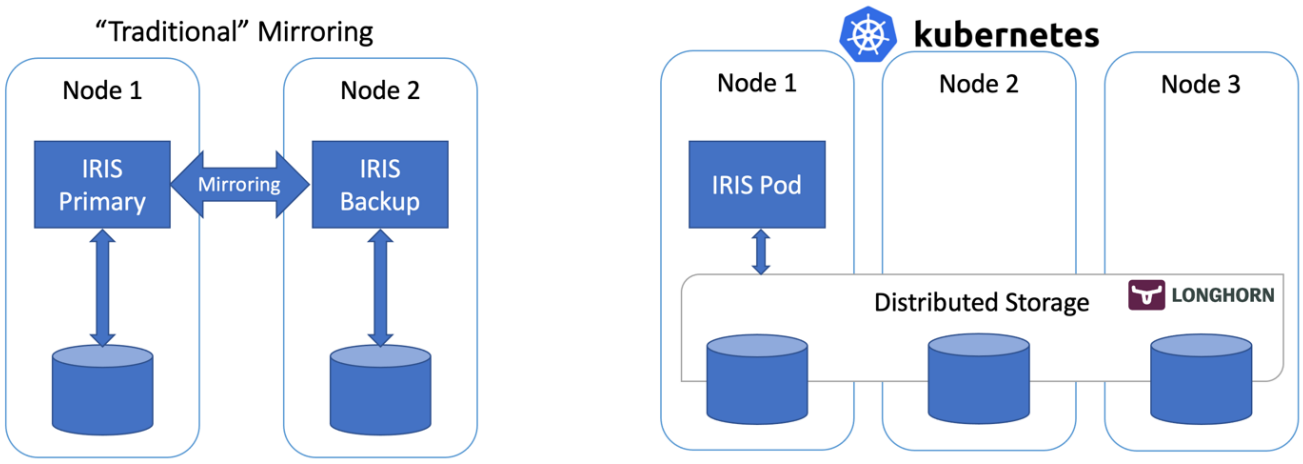


Figura 1: Mirroring tradicional vs Kubernetes con Almacenamiento Distribuido

Todos los códigos fuente utilizados en este artículo están disponibles en https://github.com/antonum/ha-iris-k8s

TL;DR

Suponiendo que tienes un clúster de 3 nodos funcionando y que estás un poco familiarizado con Kubernetes, sigue este script:

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

Si no estás seguro de lo que significan las dos líneas anteriores o no tienes el sistema adecuado para ejecutarlas, ve al apartado "Requisitos de alta disponibilidad". Explicaremos las cosas con más detalle conforme avancemos.

La primera línea instala Longhorn, un almacenamiento distribuido de código abierto de Kubernetes. La segunda instala una implementación de InterSystems IRIS, usando un volumen basado en Longhorn para SYS duradero.

Espera hasta que todos los *pods* estén en funcionamiento. kubectl get pods -A

Ahora deberías tener acceso al Portal de Administración de IRIS en http://<IRIS Service Public IP>:52773/csp/sys/%25CSP.Portal.Home.zen (la contraseña predeterminada es "SYS") y a la línea de comandos de IRIS mediante:

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

Simulación del error

Ahora comenzaremos a jugar un poco. Pero, antes de hacerlo, intenta añadir algunos datos en la base de datos y asegúrate de que siguen allí cuando IRIS esté de nuevo *online*.

```
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"
```

Nuestra "ingeniería del caos" empieza aquí:

```
# Stop IRIS - Container will be restarted automatically
kubectl exec -it iris-6d8896d584-8lzn5 -- iris stop iris quietly

# Delete the pod - Pod will be recreated
kubectl delete pod iris-6d8896d584-8lzn5

# "Force drain" the node, serving the iris pod - Pod would be recreated on another no
de
kubectl drain aks-agentpool-29845772-vmss000001 --delete-local-data --ignore-
daemonsets --force

# Delete the node - Pod would be recreated on another node
# well... you can't really do it with kubectl. Find that instance or VM and KILL it.
# if you have access to the machine - turn off the power or disconnect the network ca
ble. Seriously!
```

Requisitos de alta disponibilidad

Estamos creando un sistema que pueda tolerar un error en las siguientes estructuras:

- Instancia de IRIS dentro del contenedor/máquina virtual IRIS - Nivel del error.
- Error en el Pod/Contenedor.

- Falta de disponibilidad temporal en el nodo del clúster individual. Un buen ejemplo sería la Zona de Disponibilidad temporalmente *off-line*.
- Error permanente en el nodo o en el disco del clúster individual.

Básicamente, los escenarios que hemos probado en la sección "Simulación del error".

Si se produce alguno de estos errores, el sistema debe ponerse *online* sin intervención humana y sin pérdida de datos. Técnicamente hay límites sobre qué persistencia de datos garantiza. El propio IRIS puede proporcionarlos basándose en el Ciclo del *journal* y en el uso de transacciones dentro de una aplicación: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.cls?KEY=GCDIjournal#GCDIjournalwritecycle . De cualquier forma, hablamos de menos de dos segundos para el RPO (Recovery Point Objective, "Punto objetivo de recuperación ").

Otros componentes del sistema (el servicio API de Kubernetes, la base de datos del ETCD, el servicio de LoadBalancer, DNS y otros) están fuera de nuestros objetivos y suelen ser gestionados por el Servicio Administrado de Kubernetes, como Azure AKS o AWS EKS, por lo que suponemos que ya tienen una alta disponibilidad.

Otra forma de verlo es que nosotros somos responsables de solucionar los fallos individuales de cálculo y de los componentes de almacenamiento, y asumimos que el resto está a cargo del proveedor de la infraestructura o de la nube.

Arquitectura

Para la alta disponibilidad en InterSystems IRIS, la recomendación habitual es utilizar *mirroring*. Con *mirroring,* se tienen dos instancias de IRIS siempre activas, replicando datos de forma sincronizada. Cada nodo mantiene una copia completa de la base de datos y si el nodo primario se cae, los usuarios pueden reconectarse al nodo de *backup*. Esencialmente, con el enfoque *mirroring*, IRIS es responsable de la redundancia tanto de cálculo como de almacenamiento.

Con los *mirrors* en diferentes zonas de disponibilidad, *mirroring* ofrece la redundancia necesaria tanto para los errores de cálculo como de almacenamiento, y permite un excelente RTO (Tiempo Objetivo de Recuperación o el tiempo que tarda un sistema en volver a estar online después de un fallo) de solo unos cuantos segundos. Aquí podéis acceder a la plantilla de implementación para Mirrored IRIS en la nube de AWS: https://community.intersystems.com/post/intersystems-iris-deployment%C2%A0guide-aws%C2%A0using-cloudformation-template

El lado menos bonito de *mirroring* es la complejidad de su configuración, la realización de procesos de *backup* o restauración, y la falta de replicación para las configuraciones de seguridad y los archivos locales que no tienen relación con la base de datos.

Los orquestadores de contenedores como Kubernetes (espera, es 2021... ¡¿queda algún otro?!) proporcionan redundancia de cálculo a través de la implementación de objetos, reiniciando automáticamente el Pod/Contenedor de IRIS en caso de fallo. Por eso solo se ve un nodo de IRIS ejecutándose en el diagrama de arquitectura de Kubernetes. En vez de mantener un segundo nodo de IRIS siempre funcionando, subcontratamos la disponibilidad de cálculo a Kubernetes. Kubernetes se asegurará de que el *pod* de IRIS sea recreado en caso de

que el *pod* original falle por cualquier razón.

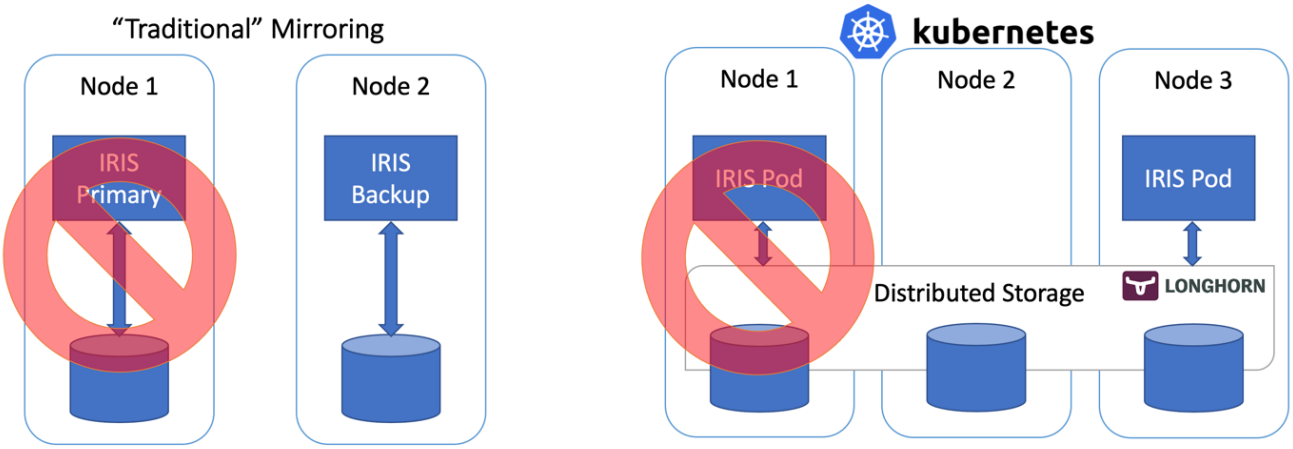


Figura 2. Escenario de la tolerancia a fallos

Por ahora todo va bien··· Si el nodo de IRIS falla, Kubernetes simplemente crea uno nuevo. Dependiendo del clúster, se necesitan entre 10 y 90 segundos para que IRIS se vuelva a conectar después del fallo de cálculo. Esto es un paso atrás comparado con el par de segundos del *mirroring*, pero es algo que se puede tolerar en el improbable caso de que haya una interrupción; y la recompensa es la gran reducción de la complejidad. No hay *mirroring* para configurar. No hay que preocuparse por la configuración de seguridad ni por la replicación de los archivos.

Francamente, si inicias sesión dentro del contenedor, ejecutando IRIS en Kubernetes, ni siquiera notarás que se está ejecutando dentro del entorno de alta disponibilidad. Todo se ve y parece como una sencilla instancia de implementación en IRIS.

Espera... ¿qué pasa con el almacenamiento? Al final, estamos lidiando con una base de datos··· Sea cual sea el escenario de respuesta ante fallos que podamos imaginar, nuestro sistema debería encargarse también de la persistencia de los datos. *Mirroring* depende del cálculo, a nivel local en el nodo de IRIS. Si el nodo se vuelve no operativo o simplemente deja de estar disponible temporalmente, también lo hace el almacenamiento de ese nodo. Por eso en la configuración de *mirroring*, IRIS se encarga de replicar bases de datos a nivel de IRIS.

Necesitamos un almacenamiento que no solo pueda preservar el estado de la base de datos al reiniciar el contenedor, sino que también pueda proporcionar redundancia en caso de que un nodo o un segmento entero de la red (Zona de Disponibilidad) se caiga. Hasta hace unos años, no había una respuesta fácil para esto. Como puedes adivinar por la imagen anterior, ya tenemos esa respuesta. Se llama **almacenamiento distribuido en contenedores**.

El almacenamiento distribuido extrae los volúmenes del *host* y los presenta como un almacenamiento conjunto disponible para cada nodo del clúster K8s. En este artículo utilizamos Longhorn https://longhorn.io - es gratuito, de código abierto y bastante fácil de instalar. Pero también puedes echar un vistazo a otros programas, como OpenEBS, Portworx y StorageOS, que ofrecen la misma funcionalidad. Rook Ceph es otro proyecto de incubación del CNCF a tener en cuenta. La gama alta de las opciones serían las soluciones de almacenamiento

empresarial, como NetApp, PureStorage y otras.

Guía paso a paso

En la sección TL;DR acabamos de instalar todo de una sola vez. El Anexo B te dirigirá paso a paso en los procesos de instalación y validación.

Almacenamiento de Kubernetes

Vamos a ir hacia atrás un momento y vamos a hablar de los contenedores y el almacenamiento en general y de cómo IRIS se integra en todo esto.

De forma predeterminada, todos los datos dentro del contenedor son efímeros. Cuando el contenedor se borra, los datos desaparecen. Para tener almacenamiento persistente en Docker y que no se elimine la información al borrar el contenedor, es necesario utilizar "volúmenes de datos". Básicamente, permite mostrar al contenedor el directorio que se encuentra en el sistema operativo del host.

```
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
```

En el ejemplo anterior, iniciamos el contenedor IRIS y hacemos que el directorio local del host "/data/dur" sea accesible al contenedor en el punto de montaje "/dur". Por lo tanto, si el contenedor está almacenando algo dentro de este directorio, se conservará y estará disponible para su uso cuando inicie de nuevo el contenedor.

Desde el punto de vista de IRIS, podemos dar indicaciones a IRIS para que almacene en el directorio específico todos los datos que necesita conservar cuando reinicie el contenedor, especificando ISCDATADIRECTORY. Durable SYS es el nombre de la función de IRIS que podrías necesitar buscar en la documentación: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.cls?KEY=ADOCK#ADOCKirisdurablerunning

En Kubernetes la sintaxis es diferente, pero los conceptos son los mismos.

Esta es la implementación básica de Kubernetes 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
```

En la implementación de la especificación anterior, la parte "volúmenes" lista los volúmenes de almacenamiento. Pueden estar disponibles fuera del contenedor, por medio de persistentVolumeClaim como "iris-pvc". volumeMounts muestra este volumen dentro del contenedor. "iris-external-sys" es el identificador que vincula el montaje del volumen con el volumen específico. En realidad, podemos tener varios volúmenes y este nombre se utiliza solo para distinguir uno de otro. Si quieres, puedes llamarlo "Steve".

La ya conocida variable de entorno ISCDATADIRECTORY dirige a IRIS para que utilice un punto de montaje específico para almacenar todos los datos que necesiten conservarse tras el reinicio del contenedor.

Ahora, echemos un vistazo a la reclamación de volúmenes persistente iris-pvc.

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

Es bastante sencillo. Se solicitan 10 gigabytes, montables como lectura/escritura en un solo nodo, utilizando la clase de almacenamiento de "longhorn".

La clase storageClassName: longhorn es realmente crítica aquí.

Veamos qué clases de almacenamiento están disponibles en mi clúster 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
```

Hay algunas clases de almacenamiento en Azure, instaladas de forma predeterminada, y una en Longhorn que instalamos como parte del primer comando:

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

Si omites #storageClassName: longhorn en la definición de la reclamación de volúmenes persistente, utilizará la clase de almacenamiento, actualmente marcada como "predeterminada", la cual es un disco normal de Azure.

Para ilustrar por qué necesitamos el almacenamiento distribuido, vamos a repetir los experimentos de "ingeniería del caos" que describimos al principio del artículo sin el almacenamiento de Longhorn. Los dos primeros escenarios (parar IRIS y eliminar el Pod) se completarían correctamente y los sistemas volverían al estado operativo. Intentar vaciar o eliminar el nodo llevaría al sistema a un estado de error.

```
#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.
```

Esencialmente, Kubernetes intentaría reiniciar el *pod* de IRIS en el clúster, pero el nodo donde se inició originalmente no está disponible y los otros dos nodos tienen un "conflicto de afinidad con el nodo del volumen". Con este tipo de almacenamiento, el volumen está disponible solo en el nodo que se creó originalmente, ya que básicamente está vinculado al disco disponible en el host del nodo.

Con Longhorn como clase de almacenamiento, tanto los experimentos "forzar el vaciado" como "eliminar el nodo" tuvieron éxito, y el *pod* de IRIS vuelve a funcionar en poco tiempo. Para lograrlo, Longhorn toma el control sobre el almacenamiento disponible en los 3 nodos del

clúster y replica los datos a través de los tres nodos. Longhorn repara rápidamente el almacenamiento del clúster si uno de los nodos deja de estar disponible de forma permanente. En nuestro escenario " eliminar el nodo " , el pod de IRIS se reinicia inmediatamente en otro nodo usando dos réplicas restantes del volumen. A continuación, AKS proporciona un nuevo nodo para sustituir el que perdió y tan pronto como esté listo Longhorn entra en acción y reconstruye los datos necesarios en el nuevo nodo. Todo es automático, no necesita tu participación.

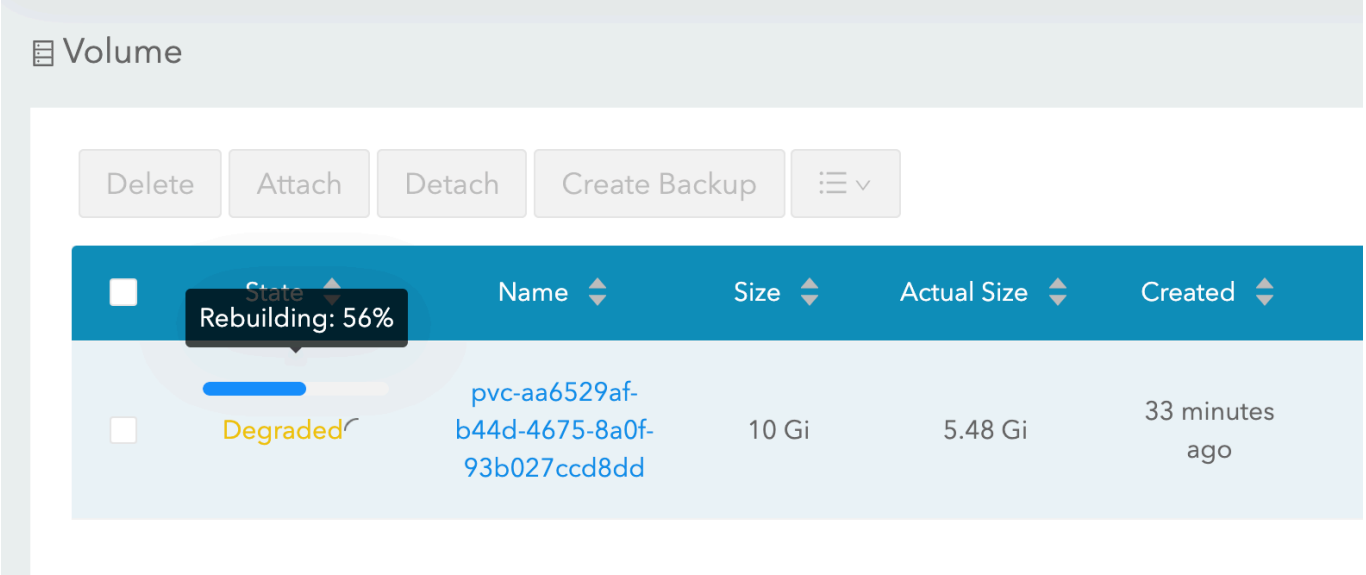


Figura 3. Longhorn reconstruyendo el volumen de la replicación en el nodo reemplazado

Más información sobre la implementación de K8s

Echemos un vistazo a algunos otros aspectos de nuestra implementación:

```
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
```

Estrategia: Recreate, replicas: 1 le dice a Kubernetes que en cualquier momento debe mantener una y exactamente una instancia del pod de IRIS en ejecución. Esto es de lo que se encarga nuestro escenario "eliminar el pod".

livenessProbe. Esta sección se asegura de que IRIS siempre esté dentro del contenedor y gestione el escenario "IRIS se ha caído". initialDelaySeconds permite un cierto periodo de gracia para que IRIS comience. Es posible que quieras aumentarlo si IRIS está tardando mucho tiempo en iniciar tu implementación.

CPF MERGE es una función de IRIS que te permite modificar el contenido del archivo de configuración iris.cpf al iniciar el contenedor. Consulta https://docs.intersystems.com/irisforhealthlatest/csp/docbook/DocBook.UI.Page.cls?KEY=RACScpf#RACScpfeditmerge para más información. En este ejemplo estoy usando Kubernetes Config Map para administrar el contenido del archivo merge: https://github.com/antonum/ha-iris-k8s/blob/main/iris-cpf-merge.yaml Aquí ajustamos los buffers globales y los valores de gmheap, utilizados por la instancia de IRIS, pero todo lo que puedes encontrar en el archivo iris.cpf es posible. Incluso puedes cambiar la contraseña predeterminada de IRIS usando el campo "PasswordHash" en el archivo CPF Merge. Más información en: https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.cls?KEY=ADOCK#ADOCKirisimagespasswordauth

Además de la reclamación de volúmenes persistente https://github.com/antonum/ha-iris-k8s/blob/main/iris-pvc.yaml, implementación https://github.com/antonum/ha-iris-k8s/blob/main/iris-deployment.yaml y ConfigMap con contenido de CPF Merge https://github.com/antonum/ha-iris-k8s/blob/main/iris-cpf-merge.yaml, nuestra puesta en marcha necesita un servicio que exponga la implementación de IRIS en el

internet público: 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
```

La IP externa del iris-svc se puede utilizar para acceder al Portal de administración de IRIS a través de: http://40.88.123.45:52773/csp/sys/%25CSP.Portal.Home.zen. La contraseña predeterminada es "SYS".

Copia de seguridad/restauración y escalado del almacenamiento

Longhorn ofrece una interfaz de usuario basada en la web para configurar y administrar los volúmenes.

Identifica el pod, ejecutando el componente longhorn-ui y establece el reenvío de puertos con 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
```

La interfaz de usuario de Longhorn estará disponible en: http://localhost:9000

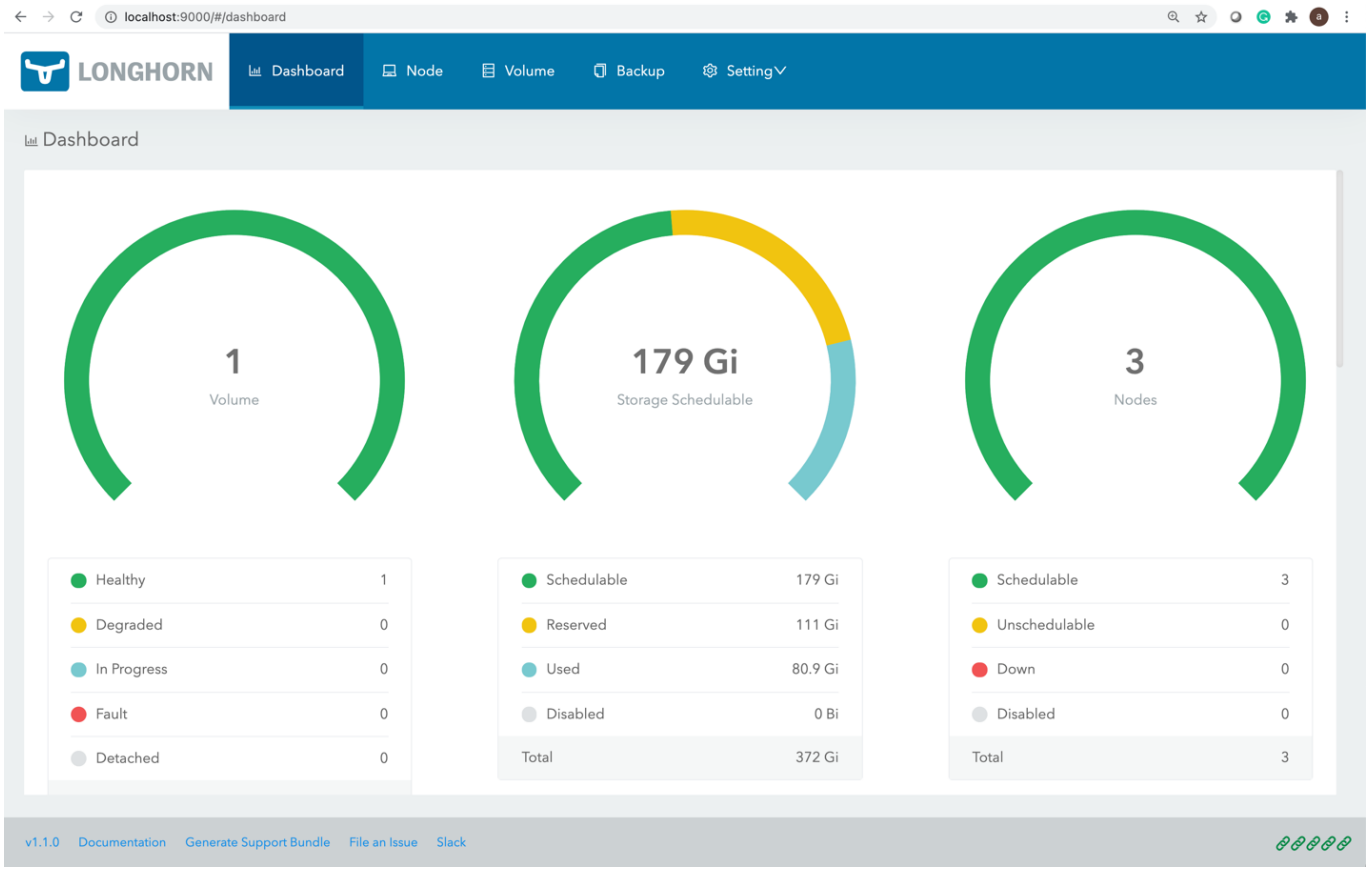


Figura 4. Interfaz de usuario de Longhorn

Además de la alta disponibilidad, la mayoría de las soluciones de almacenamiento para contenedores y Kubernetes ofrecen opciones útiles para copias de seguridad, *snapshots* y recuperación. Los detalles son específicos de la implementación, pero lo habitual es que la copia de seguridad esté asociada con VolumeSnapshot. Así es para Longhorn. Dependiendo de tu versión de Kubernetes y de tu proveedor, es posible que también necesites instalar el volumen de snapshotter https://github.com/kubernetes-csi/external-snapshotter

"iris-volume-snapshot.yaml" es un ejemplo de ese *snapshot* de volumen. Antes de utilizarlo, es necesario configurar las copias de seguridad en el bucket S3 o en el volumen del sistema de archivos de red (NFS) en Longhorn. https://longhorn.io/docs/1.0.1/snapshots-and-backups/backup-and-restore/set-backup-target/

```
# Take crash-consistent backup of the iris volume
kubectl apply -f iris-volume-snapshot.yaml
```

En el caso de IRIS, se recomienda ejecutar *Freeze* externo antes de tomar la copia de seguridad/*snapshot* y después Thaw. Consulta los detalles aquí: https://docs.intersystems.com/irisforhealthlatest/csp/documatic/%25CSP.Documatic.cls?LIBRARY=%25SYS&CLASSNAME=Backup.General#ExternalFreeze

Para aumentar el tamaño del volumen de IRIS - Ajusta la solicitud de almacenamiento en la reclamación de volúmenes persistente (archivo "iris-pvc.yaml"), utilizada por IRIS.

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

A continuación, vuelve a aplicar la especificación de la reclamación de volúmenes persistente (PVC). Longhorn no puede aplicar este cambio mientras el volumen esté conectado al pod que está en ejecución. Cambia temporalmente el recuento de réplicas hasta cero en la implementación, para poder aumentar el tamaño del volumen.

Alta disponibilidad: Resumen

Al principio del artículo, establecimos algunos criterios para la alta disponibilidad. Esto es lo que logramos con esta arquitectura:

Dominio de fallos	**Mitigados automáticamente por**
Instancia de IRIS dentro del contenedor/máquina virtual. IRIS: Nivel del fallo	La implementación Liveness probe reinicia el co caso de que IRIS no funcione
Fallo en el Pod/Contenedor.	La implementación recrea el Pod
Falta de disponibilidad temporal en el nodo del clúster individual. Un buen ejemplo sería la Zona de disponibilidad que está off-line.	La implementación recrea el pod en otro nodo. L hace que los datos estén disponibles en otro no
Fallo permanente en el nodo o en el disco del clúster	Igual que lo expuesto anteriormente + el autoes

individual.	clúster K8s sustituye un nodo dañado por uno n Longhorn reconstruye los datos en el nuevo nod

Zombis y otras cosas a tener en cuenta

Si estás familiarizado con la ejecución de IRIS en los contenedores Docker, es posible que hayas utilizado la marca "--init".

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

El objetivo de esta marca es evitar la formación de los "procesos zombis". En Kubernetes, puedes utilizar o bien "shareProcessNamespace: true" (se aplican consideraciones de seguridad) o utilizar "tini" en tus propios contenedores. Ejemplo de Dockerfile con 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"]
```

A partir de 2021, todas las imágenes de contenedores proporcionadas por InterSystems incluirán tini de forma predeterminada.

Puedes reducir aún más el tiempo de tolerancia a fallos para los escenarios de "forzar el vaciado del nodo/eliminar el nodo" ajustando algunos parámetros:

Política para eliminar Pods de Longhorn https://longhorn.io/docs/1.1.0/references/settings/#pod-deletion-policy-when-node-is-down y desalojo basado en imperfecciones de Kubernetes: https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/#taint-based-evictions

Descargo de responsabilidad

Como empleado de InterSystems, tengo que poner esto aquí: *Longhorn se utiliza en este artículo como un ejemplo de almacenamiento en bloque distribuido para Kubernetes. InterSystems no valida ni emite ninguna afirmación de soporte oficial para soluciones o productos de almacenamiento individual. Necesitas probar y validar si alguna solución de almacenamiento específica se ajusta a tus necesidades.*

El almacenamiento distribuido también puede tener características de rendimiento sustancialmente diferentes, en comparación con el almacenamiento local de los nodos. Especialmente para las operaciones de escritura, en las que los datos deben escribirse en

varias ubicaciones antes de que se considere que están en el estado persistente. Asegúrate de probar tus cargas de trabajo y entender el comportamiento específico y las opciones que ofrece tu controlador CSI.

Básicamente, InterSystems no valida ni respalda soluciones de almacenamiento específicas como Longhorn, de la misma manera que no validamos marcas individuales de discos duros o fabricantes de hardware para servidores. Personalmente, Longhorn me pareció fácil de utilizar y su equipo de desarrollo, extremadamente atentos y amables en la página de proyectos de GitHub. https://github.com/longhorn/longhorn

Conclusiones

El ecosistema de Kubernetes ha evolucionado significativamente en los últimos años y con el uso de soluciones de almacenamiento en bloque distribuido, ahora se puede construir una configuración de alta disponibilidad que puede sostener la instancia de IRIS, el nodo del clúster e incluso los errores en la zona de disponibilidad.

Puedes subcontratar el cálculo y la alta disponibilidad del almacenamiento para los componentes de Kubernetes, lo que resulta en un sistema significativamente más sencillo de configurar y mantener, comparado con el *mirroring* tradicional de IRIS. Al mismo tiempo, esta configuración podría no proporcionar el mismo RTO y rendimiento de almacenamiento que la configuración *mirrored*.

En este artículo, creamos una configuración IRIS de alta disponibilidad utilizando Azure AKS como sistema de almacenamiento distribuido administrado por Kubernetes y Longhorn. Puedes explorar varias alternativas como AWS EKS, Google Kubernetes Engine, StorageOS, Portworx y OpenEBS como almacenamiento distribuido en contenedores o incluso soluciones de almacenamiento a nivel empresarial como NetApp, PureStorage, Dell EMC y otros.

Anexo A. Cómo crear un clúster de Kubernetes en la nube

El servicio de Kubernetes administrado de uno de los proveedores de nube pública es una forma sencilla de crear el clúster K8s necesario para esta configuración. La configuración predeterminada en AKS de Azure está lista para ser utilizada en la implementación descrita en este artículo.

Crea un nuevo clúster AKS con 3 nodos. Deja todo lo demás de forma predeterminada.

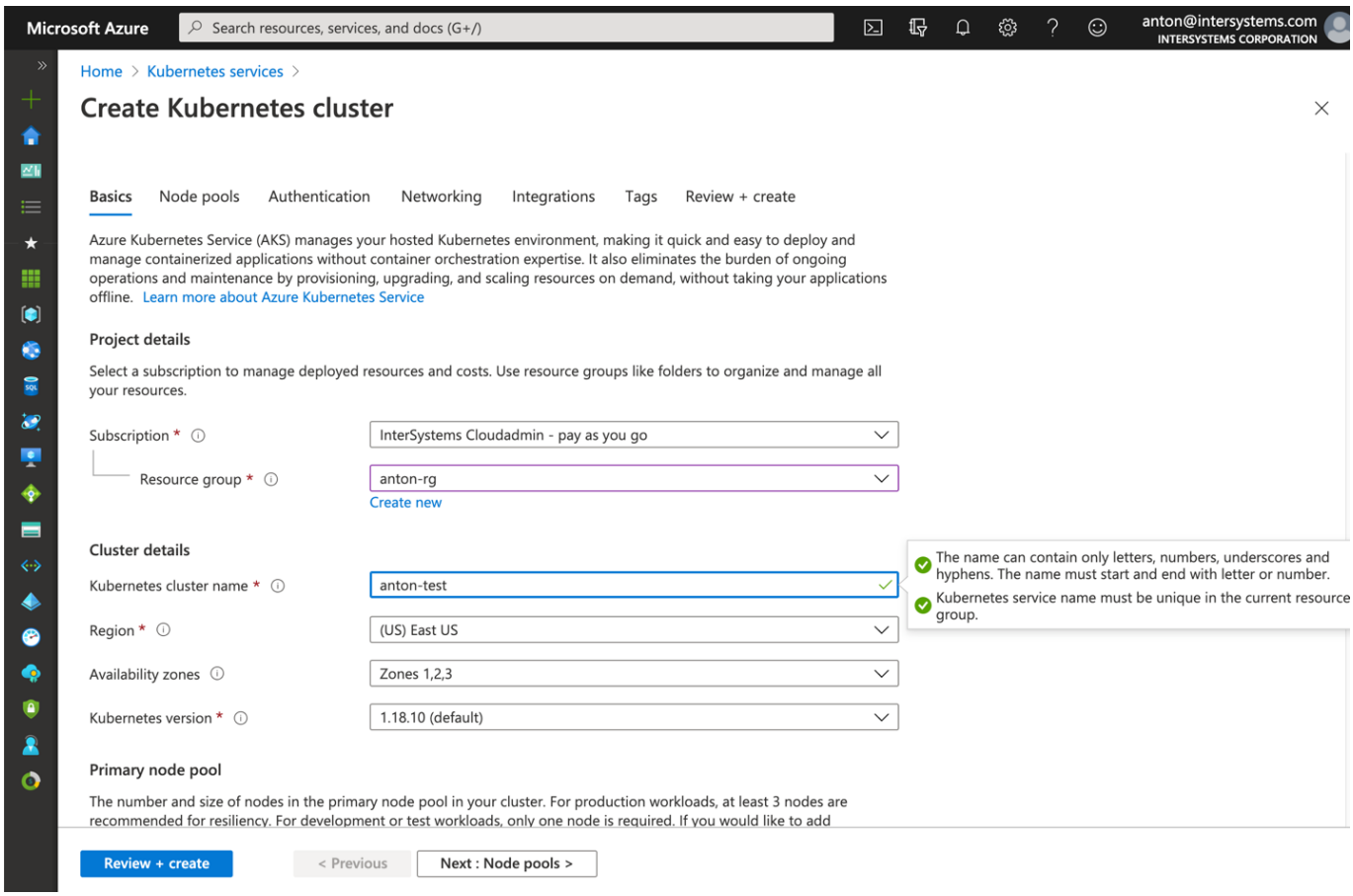


Figura 5. Crear un clúster AKS

Instala kubectl en tu equipo de forma local: https://kubernetes.io/docs/tasks/tools/install-kubectl/

Registra tu clúster de AKS con kubectl local

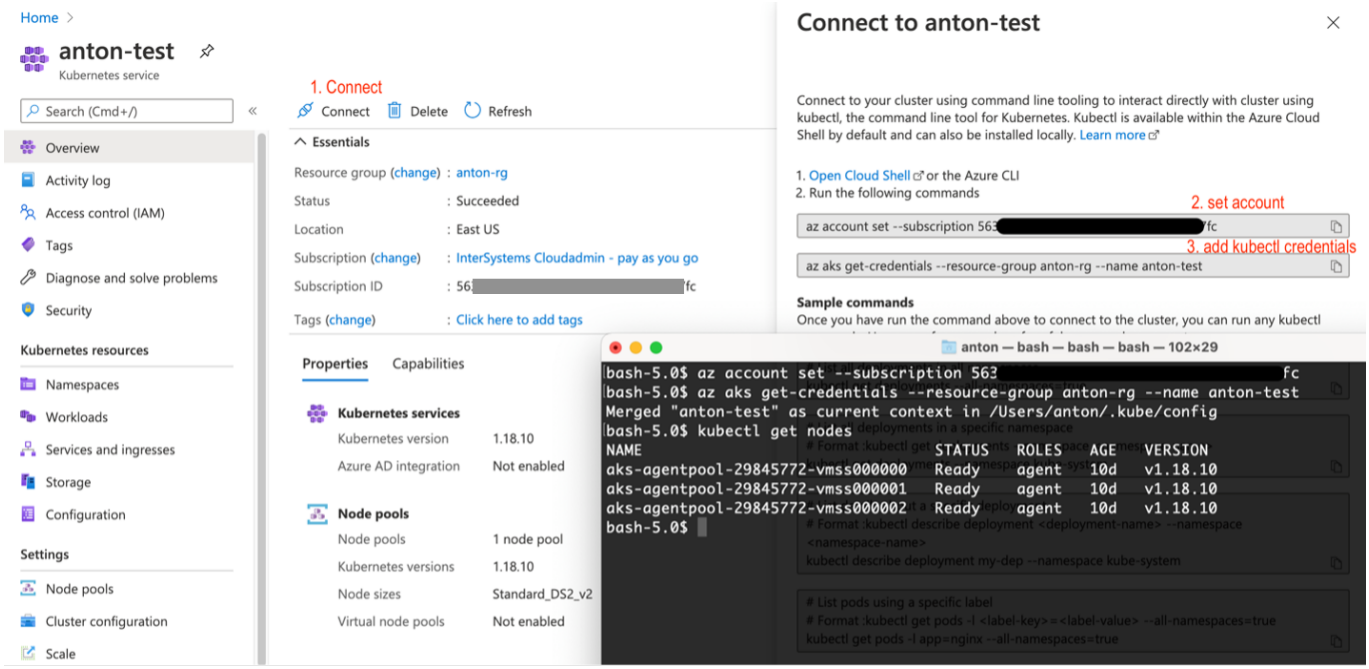


Figura 6. Registrar el clúster de AKS con kubectl

Después, puedes volver al principio del artículo e instalar Longhorn y la implementación de IRIS.

La instalación en AWS EKS es un poco más complicada. Debes asegurarte de que todas las instancias de tu grupo de nodos tengan instalado open-iscsi.

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

Instalar Longhorn en GKE requiere de un paso adicional, descrito aquí: https://longhorn.io/docs/1.0.1/advanced-resources/os-distro-specific/csi-on-gke/

Anexo B. Instalación paso a paso

Paso 1: Clúster de Kubernetes y kubectl

Necesitas 3 nodos del clúster K8s. En el Anexo A se describe cómo obtener uno en 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
```

Paso 2: Instalar Longhorn

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

Asegúrate de que todos los *pods* del *namespace* ‘ longhorn-system ’ estén funcionando. Podría tardar unos 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
```

Consulta la guía de instalación de Longhorn para más detalles y resolución de problemas https://longhorn.io/docs/1.1.0/deploy/install/install-with-kubectl

Paso 3: Clonar el repositorio de 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
```

Paso 4: Implementar y validar componentes uno por uno

El archivo tldr.yaml contiene todos los componentes necesarios para la implementación en un solo paquete. Aquí los instalaremos uno por uno y validaremos la configuración de cada uno de ellos de forma individual.

```
# If you have previously applied tldr.yaml - delete it.
$ kubectl delete -f https://github.com/antonum/ha-iris-k8s/raw/main/tldr.yaml

# Create Persistent Volume Claim
$ 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

# Create Config Map
$ 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>

# create iris deployment
$  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

# note the pod name. You'll use it to connect to the pod in the next command

$ 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

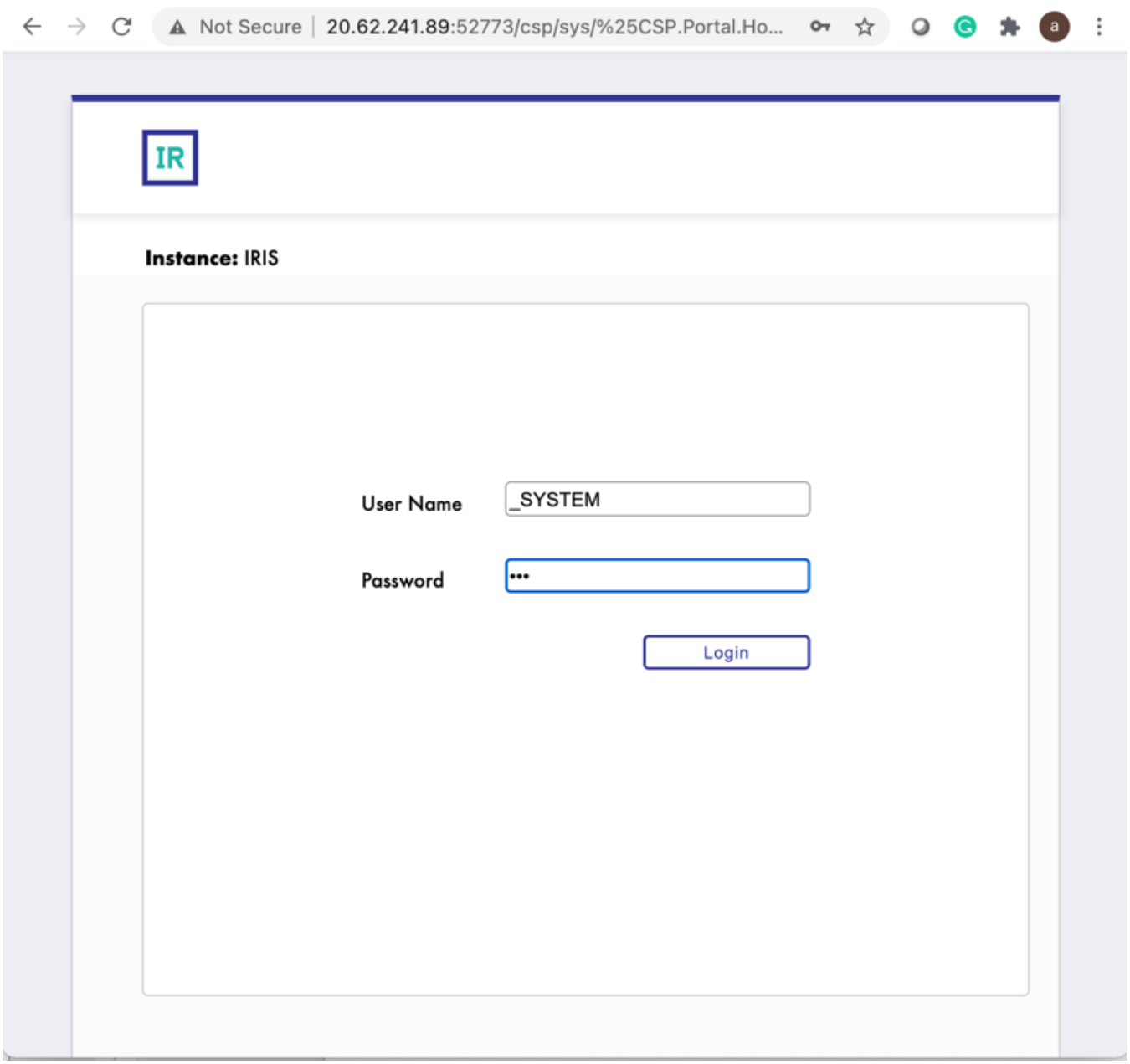
# access the logs of the IRIS container
$ kubectl logs iris-65dcfd9f97-v2rwn
...
[INFO] ...started InterSystems IRIS instance IRIS
01/18/21-23:09:11:312 (1173) 0 [Utility.Event] Private webserver started on 52773
01/18/21-23:09:11:312 (1173) 0 [Utility.Event] Processing Shadows section (this syste
m 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.

# create iris service
$ 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
```

Paso 5: Acceso al Portal de administración

Por último: Conéctate al Portal de administración de IRIS, utilizando la IP externa del servicio http://20.62.241.89:52773/csp/sys/%25CSP.Portal.Home.zen username SYSTEM, Password SYS. Se te pedirá que la cambies cuando inicies sesión por primera vez.



#Alta disponibilidad #AWS #Azure #Backup #Failover #GCP #Kubernetes #Mejores prácticas #Mirroring #InterSystems IRIS
Ir a la aplicación en InterSystems Open Exchange

URL de fuente:https://es.community.intersystems.com/post/implementaci%C3%B3n-de-iris-con-alta-disponibilidad-en-kubernetes-sin-mirroring