

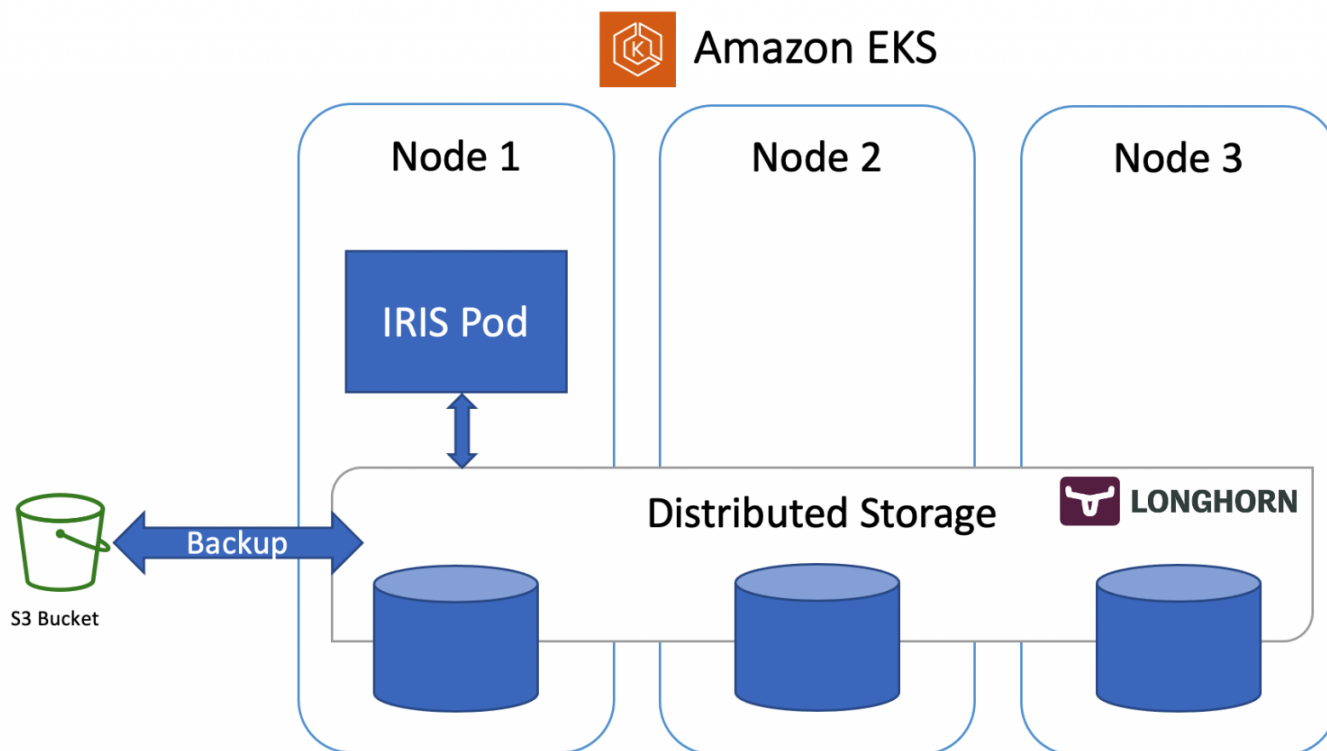
Artículo

[Ricardo Paiva](#) · 3 mayo, 2022 · Lectura de 7 min

[Open Exchange](#)

Amazon EKS e IRIS. Alta disponibilidad y copias de seguridad

Todo el código fuente del artículo está disponible en: <https://github.com/antonum/ha-iris-k8s>



En el [artículo anterior](#), comentamos cómo configurar IRIS en el clúster k8s con alta disponibilidad, basado en el almacenamiento distribuido, en vez del mirroring tradicional. Como ejemplo, ese artículo utilizaba el clúster Azure AKS. En esta ocasión, seguiremos explorando las configuraciones de alta disponibilidad en los k8s. Esta vez, basado en Amazon EKS (servicio administrado para ejecutar Kubernetes en AWS) e incluiría una opción para hacer copias de seguridad y restauraciones de la base de datos, basado en Kubernetes Snapshot.

Instalación

Vamos a lo más importante. Primero: necesitas una cuenta de AWS, y las herramientas [AWS CLI](#), [kubectl](#) y [eksctl](#) instaladas. Para crear el nuevo clúster, ejecuta el siguiente comando:

```
eksctl create cluster \
--name my-cluster \
--node-type m5.2xlarge \
--nodes 3 \
--node-volume-size 500 \
--region us-east-1
```

Este comando tarda unos 15 minutos, implementa el clúster EKS y lo convierte en un clúster predeterminado para

tu herramienta kubectl. Puedes verificar la implementación ejecutando:

```
kubectl get nodes
```

NAME			STATUS	ROLES	AGE	VERSION
ip-192-168-19-7.ca-central-1.compute.internal	Ready	<none>	18d			v1.18.9-eks-d1db3c
ip-192-168-37-96.ca-central-1.compute.internal	Ready	<none>	18d			v1.18.9-eks-d1db3c
ip-192-168-76-18.ca-central-1.compute.internal	Ready	<none>	18d			v1.18.9-eks-d1db3c

El siguiente paso es instalar el motor de almacenamiento distribuido Longhorn.

```
kubectl create namespace longhorn-system
kubectl apply -f https://raw.githubusercontent.com/longhorn/longhorn/v1.1.0/deploy/iscsi/longhorn-iscsi-installation.yaml --namespace longhorn-system
kubectl apply -f https://raw.githubusercontent.com/longhorn/longhorn/master/deploy/longhorn.yaml --namespace longhorn-system
```

Y, por último, el propio IRIS:

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

En este momento, tendrás un clúster EKS completamente funcional con el almacenamiento distribuido de Longhorn y la implementación de IRIS instaladas. Puedes volver al artículo anterior e intentar hacer todo tipo de daño en el clúster y la implementación de IRIS, solo para ver cómo el sistema se repara a sí mismo. Echa un vistazo a la sección ["Simulación del error"](#).

Bonus #1 IRIS en ARM

IRIS EKS y Longhorn son compatibles con la arquitectura ARM, por lo que podemos implementar la misma configuración utilizando instancias de AWS Graviton2, basadas en la arquitectura ARM.

Todo lo que necesitas es cambiar el tipo de instancia para los nodos EKS a la familia "m6g" y la imagen IRIS a la basada en ARM.

```
eksctl create cluster /
--name my-cluster-arm /
--node-type m6g.2xlarge /
--nodes 3 /
--node-volume-size 500 /
--region us-east-1
```

tldr.yaml

```
containers:
#- image: store/intersystems/iris-community:2020.4.0.524.0
- image: store/intersystems/irishealth-community-arm64:2020.4.0.524.0
name: iris
```

O simplemente utiliza:

```
kubectl apply -f https://github.com/antonum/ha-iris-k8s/raw/main/tldr-iris4h-arm.yaml
```

¡Eso es todo! Ya tienes el clúster IRIS Kubernetes, ejecutándose en la plataforma ARM.

Bonus #2 Copia de seguridad y restauración

Una parte de la arquitectura para nivel de producción que se suele pasar por alto es la capacidad de crear copias de seguridad de la base de datos y que las restaure y/o clone rápidamente cuando sea necesario.

En Kubernetes, la manera más habitual de hacerlo es utilizando Persistent Volumen Snapshots (Snapshots de volumen persistente).

En primer lugar, debes instalar todos los componentes de k8s requeridos:

```
#Install CSI Snapshotter and CRDs
```

```
kubectl apply -n kube-system -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/master/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -n kube-system -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/master/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
kubectl apply -n kube-system -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/master/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
kubectl apply -n kube-system -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/master/manifests/deploys/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
kubectl apply -n kube-system -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/master/manifests/deploys/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
```

A continuación, configura las credenciales del bucket S3 para Longhorn (consulta [estas instrucciones](#)):

```
#Longhorn backup target s3 bucket and credentials longhorn would use to access that bucket
#See https://longhorn.io/docs/1.1.0/snapshots-and-backups/backup-and-restore/set-backup-target/ for manual setup instructions
longhorn_s3_bucket=longhorn-backup-123xx #bucket name should be globally unique, unless you want to reuse existing backups and credentials
longhorn_s3_region=us-east-1
longhorn_aws_key=AKIAVHCUNTEXAMPLE
longhorn_aws_secret=g2q2+5DVXk5p3AHIB5m/Tk6U6dXrEXAMPLE
```

El siguiente comando tomará las variables de entorno del paso anterior y las utilizará para configurar la copia de seguridad de Longhorn

```
#configure Longhorn backup target and credentials
```

```
cat <&&EOF | kubectl apply -f -
apiVersion: longhorn.io/v1beta1
kind: Setting
metadata:
  name: backup-target
  namespace: longhorn-system
value: "s3://$longhorn_s3_bucket@$longhorn_s3_region/" # backup target here
---
apiVersion: v1
```

```
kind: Secret
metadata:
  name: "aws-secret"
  namespace: "longhorn-system"
  labels:
data:
  # echo -n '<secret>' | base64
  AWS_ACCESS_KEY_ID: $(echo -n $longhorn_aws_key | base64)
  AWS_SECRET_ACCESS_KEY: $(echo -n $longhorn_aws_secret | base64)
  ---
apiVersion: longhorn.io/v1beta1
kind: Setting
metadata:
  name: backup-target-credential-secret
  namespace: longhorn-system
value: "aws-secret" # backup secret name here
EOF
```

Puede parecer mucho, pero esencialmente le indica a Longhorn que utilice un bucket S3 específico con las credenciales precisas para almacenar el contenido de las copias de seguridad.

¡Eso es todo! Si ahora vas a la interfaz de usuario de Longhorn, podrás crear copias de seguridad, restaurarlas, etc.

Un recordatorio rápido sobre cómo conectarse a la interfaz de usuario de Longhorn:

```
kubectl get pods -n longhorn-system
# note the full pod name for 'longhorn-ui-...' pod
kubectl port-forward longhorn-ui-df95bdf85-469sz 9000:8000 -n longhorn-system
```

Esto reenviaría el tráfico desde la interfaz de usuario de Longhorn a tu <http://localhost:9000>

Programación de la copia de seguridad/restauración

Hacer copias de seguridad y restauraciones a través de la interfaz de usuario de Longhorn podría ser un primer paso suficientemente bueno, pero vamos a ir un paso más allá y realizar copias de seguridad y restauraciones programáticamente, utilizando APIs de Snapshots k8s.

En primer lugar, el propio snapshot. iris-volume-snapshot.yaml

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
  name: iris-longhorn-snapshot
spec:
  volumeSnapshotClassName: longhorn
  source:
    persistentVolumeClaimName: iris-pvc
```

Este snapshot de volumen se refiere al volumen de la fuente "iris-pvc" que usamos para nuestra implementación de IRIS. Así que con solo aplicar esto, el proceso para crear una copia de seguridad se iniciaría inmediatamente .

Es una buena idea ejecutar el Freeze/Thaw de IRIS Write Daemon antes o después del snapshot.

```
#Freeze Write Daemon
```

```
echo "Freezing IRIS Write Daemon"
kubectl exec -it -n $namespace $pod_name -- iris session iris -U%SYS "##Class(Backup.General).ExternalFreeze()"
status=$?
if [[ $status -eq 5 ]]; then
    echo "IRIS WD IS FROZEN, Performing backup"
    kubectl apply -f backup/iris-volume-snapshot.yaml -n $namespace
elif [[ $status -eq 3 ]]; then
    echo "IRIS WD FREEZE FAILED"
fi
#Thaw Write Daemon
kubectl exec -it -n $namespace $pod_name -- iris session iris -U%SYS "##Class(Backup.General).ExternalThaw()"
```

El proceso de restauración es bastante sencillo. Esencialmente se crea un nuevo PVC y se especifican los snapshots como la fuente.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: iris-pvc-restored
spec:
  storageClassName: longhorn
  dataSource:
    name: iris-longhorn-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

A continuación, solo hay que crear una nueva implementación, basada en este PVC. Comprueba este [script de prueba en un repositorio de github](#) que podría realizarse con la siguiente secuencia:

- Crear una nueva implementación de IRIS
- Añadir algunos datos a IRIS
- Freeze Write Daemon, tome un snapshot, thaw Write Daemon
- Crear un clon de implementación de IRIS, basado en el snapshot
- Verificar que todos los datos todavía están allí

En este momento tendrá dos implementaciones idénticas de IRIS, una de las cuales es un clon hecho por medio de una copia de seguridad de la otra.

¡Espero que os resulte útil!

[#Alta disponibilidad](#) [#AWS](#) [#Backup](#) [#Contenedorización](#) [#Despliegue](#) [#DevOps](#) [#Nube](#) [#InterSystems IRIS](#)
[#InterSystems IRIS for Health](#) [#Open Exchange](#)
[Ir a la aplicación en InterSystems Open Exchange](#)

URL de
fuente: <https://es.community.intersystems.com/post/amazon-eks-e-iris-alta-disponibilidad-y-copias-de-seguridad>
