Article
Lorenzo Scalese · Feb 8, 2022 11m read

Open Exchange

How to set up a mirror programmatically

History

Version	Date	Changes
V1	2022-02-08	Initial release
V1.1	2022-04-06	Certificates generation with sh file instead of pki-script Using environment variables in configuration files

Hi Community,

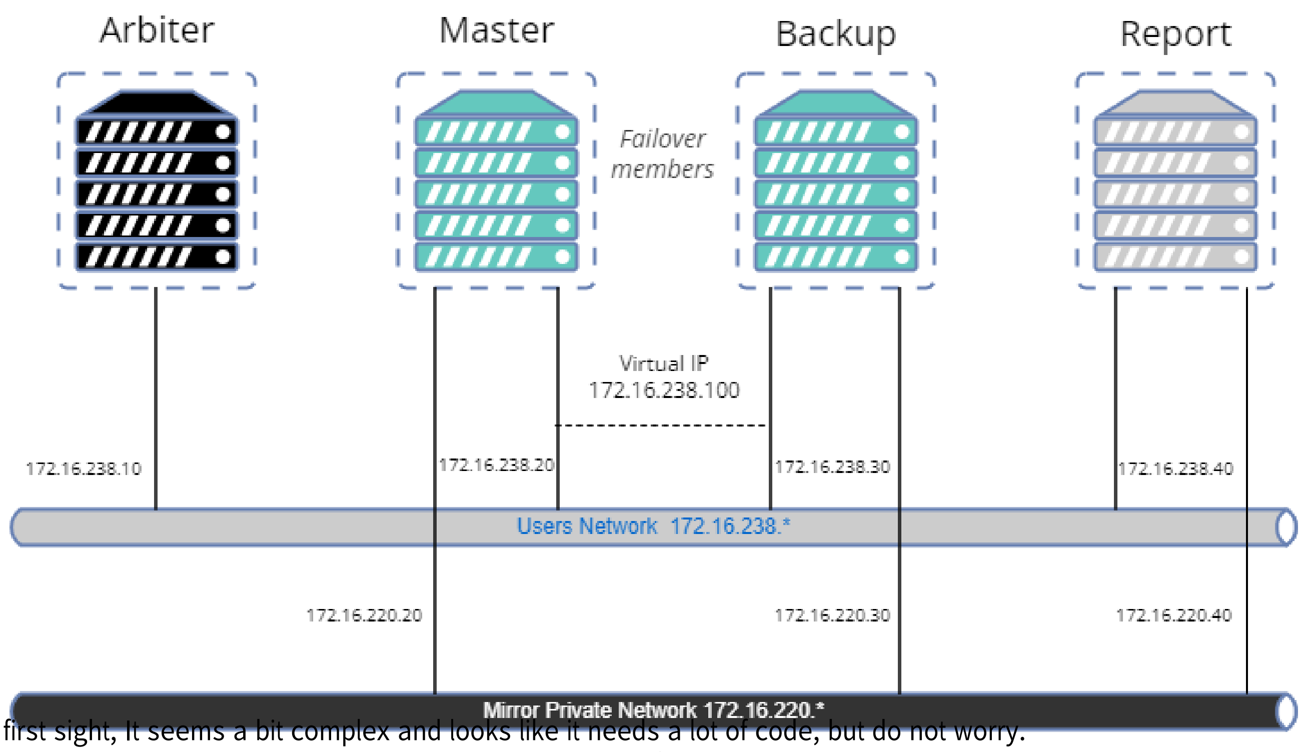
Have you already set up a mirrored environment? Does it have a private network, virtual IP address, and SSL configuration?
After doing this a couple of times, I realized that it is long, and there are a lot of manual actions required to generate certificates and configure each IRIS instance.
It is a pain in the neck for people who often have to do this.

For example, a Quality Assurance team might need to create a new environment for each new application version to test. The support team might require to create an environment to reproduce a complex issue.

We definitely need tools to create them fast.

In this article we will create a sample to set up a mirror with :

- Arbiter.
- Primary.
- Backup failover member.
- Read-write report async member.
- SSL configuration for journal transfers between nodes.
- Private Network for the mirror.
- Virtual IP Address.
- A mirrored database.



At first sight, It seems a bit complex and looks like it needs a lot of code, but do not worry.
There are libraries hosted on OpenExchange to easily perform most operations.

The purpose of this article is to provide an example of how to adapt the process to your needs, but it's not a best practice guide in terms of security matters.
So, let's create our sample.

Tools and libraries

- config-api: This library will be used to configure IRIS. It supports mirroring configuration since version 1.1.0. We will not give a detailed description of how to use this library. A set of articles already exists. here. In short, config-api will be used to create IRIS template configuration files (JSON format) and load them easily.
- ZPM.
- Docker.
- OpenSSL.

Github page

You can find all necessary resource files on iris-mirroring-samples repository.

Prepare your system

Clone the existing repository:

```
git clone https://github.com/lscalese/iris-mirroring-samples
cd iris-mirroring-samples
```

If you prefer to create a sample from scratch, instead of cloning the repository, just create a new directory with subdirectories: backup, and config-files. Download irissession.sh :

```
mkdir -p iris-mirroring-samples/backup iris-mirroring-samples/config-files
cd  iris-mirroring-samples
wget -O session.sh https://raw.githubusercontent.com/lscalese/iris-mirroring-
samples/master/session.sh
```

To avoid the issue "permission denied ” later, we need to create irisowner group, irisowner user, and change the group of backup directory to irisowner

```
sudo useradd --uid 51773 --user-group irisowner
sudo groupmod --gid 51773 irisowner
sudo chgrp irisowner ./backup
```

This directory will be used as a volume to share a database backup after setting up the first mirror member with the other nodes.

Get an IRIS License

Mirroring is not available with the IRIS community edition.
If you do not have a valid IRIS container License yet, connect to Worldwide Response Center (WRC) with your credentials.
Click "Actions" --> "Online distribtion", then "Evaluations" button and select "Evaluation License"; fill the form.
Copy your license file iris.key to this directory.

Login to Intersystems Containers Registry

For convenience reasons, we use Intersystems Containers Registry (ICR) to pull docker images. If you don't know your docker login\password, just connect to SSO.UI.User.ApplicationTokens.cls with your WRC credentials, and you can retrieve your ICR Token.

```
docker login -u="YourWRCLogin" -p="YourICRToken" containers.intersystems.com
```

Create the myappdata database and a global mapping

We do not really create myappdata database now but prepare a configuration to create it at docker build time.
For that, we just create a simple file using JSON format;
config-api library will be used to load it in IRIS instances.

Create the file config-files/simple-config.json

```
{
   "Defaults":{
       "DBDATADIR" : "${MGRDIR}myappdata/",
       "DBDATANAME" : "MYAPPDATA"

   },
   "SYS.Databases":{
       "${DBDATADIR}" : {}
   },
   "Databases":{
       "${DBDATANAME}" : {
           "Directory" : "${DBDATADIR}"
       }
   },
   "MapGlobals":{
       "USER": [{
           "Name" : "demo.*",
           "Database" : "${DBDATANAME}"
       }]
   },
   "Security.Services" : {
       "%Service_Mirror" : {                      /* Enable the mirror service on thi
s instance */
           "Enabled" : true
       }
   }
}
```

This configuration file allows you to create a new database with default settings and make global mapping demo.* in the USER namespace.

For more information about config-api configuration file capabilities refer to the related article or the github page

The Docker file

The docker file is based on the existing docker template, but we need to make some changes to create a working directory, install tools for using virtual IP, install ZPM,etc…

Our IRIS image is the same for each mirror member. The mirroring will be set up on the container starting with the correct configuration depending on its role (first member, failover backup, or read-write report). See the comments on the Dockerfile below:

```
ARG IMAGE=containers.intersystems.com/intersystems/iris:2021.1.0.215.0
# Don't need to download the image from WRC. It will be pulled from ICR at build time
.

FROM $IMAGE

USER root

COPY session.sh /
COPY iris.key /usr/irissys/mgr/iris.key

# /opt/demo will be our working directory used to store our configuration files and o
ther installation files.
# Install iputils-
arping to have an arping command.  It's required to configure Virtual IP.
# Download the latest ZPM version (ZPM is included only with community edition).
RUN mkdir /opt/demo && \
    chown ${ISC_PACKAGE_MGRUSER}:${ISC_PACKAGE_IRISGROUP} /opt/demo && \
    chmod 666 /usr/irissys/mgr/iris.key && \
    apt-get update && apt-get install iputils-arping gettext-base && \
    wget -O /opt/demo/zpm.xml https://pm.community.intersystems.com/packages/zpm/late
st/installer

USER ${ISC_PACKAGE_MGRUSER}

WORKDIR /opt/demo
```

```
# Set Default Mirror role to master.
# It will be overridden on docker-
compose file at runtime (master for the first instance, backup, and report)
ARG IRIS_MIRROR_ROLE=master

# Copy the content of the config-files directory into /opt/demo.
# Currently we have only created a simple-
config to setup our database and global mapping.
# Later in this article we will add other configuration files to set up the mirror.
ADD config-files .

SHELL [ "/session.sh" ]

# Install ZPM
# Use ZPM to install config-api
# Load simple-config.json file with config-api to:
#  - create "myappdata" database,
#  - add a global mapping in namespace "USER" for global "demo.*" on "myappdata" data
base.
# Basically, the entry point to install your ObjectScript application is here.
# For this sample, we will load simple-
config.json to create a simple database and a global mapping.
RUN \
Do $SYSTEM.OBJ.Load("/opt/demo/zpm.xml", "ck") \
zpm "install config-api" \
Set sc = ##class(Api.Config.Services.Loader).Load("/opt/demo/simple-config.json")

# Copy the mirror initialization script.
COPY init_mirror.sh /
```

Build the IRIS image

The Dockerfile is ready; we can build the image:

```
docker build --no-cache --tag mirror-demo:latest .
```

This image will be used to run primary, backup, and report nodes.

The .env file

JSON Configuration files and docker-compose use environment variables.
Their values are stored in a file named .env, for this sample our env file is:

```
APP_NET_SUBNET=172.16.238.0/24
MIRROR_NET_SUBNET=172.16.220.0/24

IRIS_HOST=172.16.238.100
IRIS_PORT=1972
IRIS_VIRTUAL_IP=172.16.238.100

ARBITER_IP=172.16.238.10

MASTER_APP_NET_IP=172.16.238.20
MASTER_MIRROR_NET_IP=172.16.220.20
```

```
BACKUP_APP_NET_IP=172.16.238.30
BACKUP_MIRROR_NET_IP=172.16.220.30

REPORT_APP_NET_IP=172.16.238.40
REPORT_MIRROR_NET_IP=172.16.220.40
```

Prepare the first mirror member configuration file

config-api library allows configuring a mirror, so we have to create a configuration file dedicated to the first mirror member config-files/mirror-master.json

For convenience, comments are located directly in the JSON. You can download the mirror-master.json without comment here.

```
{
    "Security.Services" : {
        "%Service_Mirror" : {
            "Enabled" : true
        }
    },
    "SYS.MirrorMaster" : {
        "Demo" : {
            "Config" : {
                "Name" : "Demo",                                /* The name of our mi
rror */
                "SystemName" : "master",                        /* This instance name
 in the mirror */
                "UseSSL" : true,
                "ArbiterNode" : "${ARBITER_IP}|2188",           /* IP Address and por
t of the arbiter node */
                "VirtualAddress" : "${IRIS_VIRTUAL_IP}/24",     /* Virtual IP Address
 */
                "VirtualAddressInterface" : "eth0",             /* Network interface
used for the Virtual IP Address. */
                "MirrorAddress": "${MASTER_MIRROR_NET_IP}",     /* IP Address of this
 node in the private mirror network */
                "AgentAddress": "${MASTER_APP_NET_IP}"          /* IP Address of this
 node (Agent is installed on the same machine) */
            },
            "Databases" : [{                                    /* List of databases
to add to the mirror */
                "Directory" : "/usr/irissys/mgr/myappdata/",
                "MirrorDBName" : "MYAPPDATA"
            }],
            "SSLInfo" : {                                       /* SSL Configuration
*/
                "CAFile" : "/certificates/CA_Server.cer",
                "CertificateFile" : "/certificates/master_server.cer",
                "PrivateKeyFile" : "/certificates/master_server.key",
                "PrivateKeyPassword" : "",
                "PrivateKeyType" : "2"
            }
        }
    }
}
```

Prepare the failover member configuration file

Create a configuration file the failover backup member config-files/mirror-backup.json.

It looks like the first member:

```
{
    "Security.Services" : {
        "%Service_Mirror" : {
            "Enabled" : true
        }
    },
    "SYS.MirrorFailOver" : {
        "Demo" : {                                          /* Mirror to join */
            "Config": {
                "Name" : "Demo",
                "SystemName" : "backup",                    /* This instance name in
the mirror */
                "InstanceName" : "IRIS",                    /* IRIS Instance name of
the first mirror member */
                "AgentAddress" : "${MASTER_APP_NET_IP}",    /* Agent IP Address of th
e first mirror member */
                "AgentPort" : "2188",
                "AsyncMember" : false,
                "AsyncMemberType" : ""
            },
            "Databases" : [{                                /* DB in mirror */
                 "Directory" : "/usr/irissys/mgr/myappdata/"
            }],
            "LocalInfo" : {
                "VirtualAddressInterface" : "eth0",         /* Network interface used
 for the Virtual IP Address. */
                "MirrorAddress": "${BACKUP_MIRROR_NET_IP}"  /* IP Address of this nod
e in the private mirror network */
            },
            "SSLInfo" : {
                "CAFile" : "/certificates/CA_Server.cer",
                "CertificateFile" : "/certificates/backup_server.cer",
                "PrivateKeyFile" : "/certificates/backup_server.key",
                "PrivateKeyPassword" : "",
                "PrivateKeyType" : "2"
            }
        }
    }
}
```

Prepare the read-write async member configuration file

It is pretty similar to the failover configuration file. The differences are the values of AsyncMember, AsyncMemberType, and MirrorAddress.
Create the file ./config-files/mirror-report.json:

```
{
    "Security.Services" : {
        "%Service_Mirror" : {
            "Enabled" : true
        }
    },
    "SYS.MirrorFailOver" : {
        "Demo" : {
            "Config": {
                "Name" : "Demo",
                "SystemName" : "report",
                "InstanceName" : "IRIS",
                "AgentAddress" : "${MASTER_APP_NET_IP}",
                "AgentPort" : "2188",
                "AsyncMember" : true,
                "AsyncMemberType" : "rw"
            },
            "Databases" : [{
                 "Directory" : "/usr/irissys/mgr/myappdata/"
            }],
            "LocalInfo" : {
                "VirtualAddressInterface" : "eth0",
                "MirrorAddress": "${REPORT_MIRROR_NET_IP}"
            },
            "SSLInfo" : {
                "CAFile" : "/certificates/CA_Server.cer",
                "CertificateFile" : "/certificates/report_server.cer",
                "PrivateKeyFile" : "/certificates/report_server.key",
                "PrivateKeyPassword" : "",
                "PrivateKeyType" : "2"
            }
        }
    }
}
```

Generate certificates and configure IRIS nodes and

All configuration files are ready!

Now we have to add script to generate certificates to secure communication between each nodes. A script ready to use is available on the repository gen-certificates.sh

```
# sudo is required due to chown, chgrp chmod usage.
sudo ./gen-certificates.sh
```

To configure each node init_mirror.sh will be performed on containers start. It will be configured later in docker-compose.yml in the command section command: ["-a", "/init_mirror.sh"] :

```
#!/bin/bash

# Database used to test the mirror.
DATABASE=/usr/irissys/mgr/myappdata

# Directory contain myappdata backuped by the master to restore on other nodes and ma
king mirror.
BACKUP_FOLDER=/opt/backup

# Mirror configuration file in json config-api format for the master node.
MASTER_CONFIG=/opt/demo/mirror-master.json
```

```
# Mirror configuration file in json config-api format for the backup node.
BACKUP_CONFIG=/opt/demo/mirror-backup.json

# Mirror configuration file in json config-api format for the report async node.
REPORT_CONFIG=/opt/demo/mirror-report.json

# The mirror name...
MIRROR_NAME=DEMO

# Mirror Member list.
MIRROR_MEMBERS=BACKUP,REPORT

# Performed on the master.
# Load the mirror configuration using config-api with /opt/demo/simple-
config.json file.
# Start a Job to auto-accept other members named "backup" and "report" to join the mi
rror (avoid manuel validation in portal management).
master() {
rm -rf $BACKUP_FOLDER/IRIS.DAT
envsubst < ${MASTER_CONFIG} > ${MASTER_CONFIG}.resolved
iris session $ISC_PACKAGE_INSTANCENAME -U %SYS <<- END
Set sc = ##class(Api.Config.Services.Loader).Load("${MASTER_CONFIG}.resolved")
Set ^log.mirrorconfig(\$i(^log.mirrorconfig)) = \$SYSTEM.Status.GetOneErrorText(sc)
Job ##class(Api.Config.Services.SYS.MirrorMaster).AuthorizeNewMembers("${MIRROR_MEMBE
RS}","${MIRROR_NAME}",600)
Hang 2
Halt
END
}

# Performed by the master, make a backup of /usr/irissy
make_backup() {
iris session $ISC_PACKAGE_INSTANCENAME -U %SYS "##class(SYS.Database).DismountDatabas
e(\"${DATABASE}\")"
md5sum ${DATABASE}/IRIS.DAT
cp ${DATABASE}/IRIS.DAT ${BACKUP_FOLDER}/IRIS.TMP
mv ${BACKUP_FOLDER}/IRIS.TMP ${BACKUP_FOLDER}/IRIS.DAT
chmod 777 ${BACKUP_FOLDER}/IRIS.DAT
iris session $ISC_PACKAGE_INSTANCENAME -U %SYS "##class(SYS.Database).MountDatabase(\
"${DATABASE}\")"
}

# Restore the mirrored database "myappdata".  This restore is performed on "backup" a
nd "report" node.
restore_backup() {
sleep 5
while [ ! -f $BACKUP_FOLDER/IRIS.DAT ]; do sleep 1; done
sleep 2
iris session $ISC_PACKAGE_INSTANCENAME -U %SYS "##class(SYS.Database).DismountDatabas
e(\"${DATABASE}\")"
cp $BACKUP_FOLDER/IRIS.DAT $DATABASE/IRIS.DAT
md5sum $DATABASE/IRIS.DAT
iris session $ISC_PACKAGE_INSTANCENAME -U %SYS "##class(SYS.Database).MountDatabase(\
"${DATABASE}\")"
}

# Configure the "backup" member
#  - Load configuration file /opt/demo/mirror-
```

```
backup.json if this instance is the backup or
#    /opt/demo/mirror-
report.json if this instance the report (async R\W mirror node).
other_node() {
sleep 5
envsubst < $1 > $1.resolved
iris session $ISC_PACKAGE_INSTANCENAME -U %SYS <<- END
Set sc = ##class(Api.Config.Services.Loader).Load("$1.resolved")
Halt
END
}

if [ "$IRIS_MIRROR_ROLE" == "master" ]
then
  master
  make_backup
elif [ "$IRIS_MIRROR_ROLE" == "backup" ]
then
  restore_backup
  other_node $BACKUP_CONFIG
else
  restore_backup
  other_node $REPORT_CONFIG
fi

exit 0
```

Docker-compose file

We have four containers to start. A Docker-compose file is a perfect one to orchestrate our sample.

```
version: '3.7'

services:
  arbiter:
    image: containers.intersystems.com/intersystems/arbiter:2021.1.0.215.0
    init: true
    container_name: mirror-demo-arbiter
    command:
      - /usr/local/etc/irissys/startISCAgent.sh 2188
    networks:
      app_net:
        ipv4_address: ${ARBITER_IP}
    extra_hosts:
      - "master:${MASTER_APP_NET_IP}"
      - "backup:${BACKUP_APP_NET_IP}"
      - "report:${REPORT_APP_NET_IP}"
    cap_add:
      - NET_ADMIN

  master:
    build: .
    image: mirror-demo
    container_name: mirror-demo-master
    networks:
      app_net:
        ipv4_address: ${MASTER_APP_NET_IP}
      mirror_net:
        ipv4_address: ${MASTER_MIRROR_NET_IP}
    environment:
      - IRIS_MIRROR_ROLE=master
      - WEBGATEWAY_IP=${WEBGATEWAY_IP}
      - MASTER_APP_NET_IP=${MASTER_APP_NET_IP}
      - MASTER_MIRROR_NET_IP=${MASTER_MIRROR_NET_IP}
      - ARBITER_IP=${ARBITER_IP}
      - IRIS_VIRTUAL_IP=${IRIS_VIRTUAL_IP}
    ports:
      - 81:52773
    volumes:
      - ./backup:/opt/backup
      - ./init_mirror.sh:/init_mirror.sh
      # Mount certificates
      - ./certificates/master_server.cer:/certificates/master_server.cer
      - ./certificates/master_server.key:/certificates/master_server.key
      - ./certificates/CA_Server.cer:/certificates/CA_Server.cer
      #- ~/iris.key:/usr/irissys/mgr/iris.key
    hostname: master
    extra_hosts:
      - "backup:${BACKUP_APP_NET_IP}"
      - "report:${REPORT_APP_NET_IP}"
    cap_add:
      - NET_ADMIN
    command: ["-a", "/init_mirror.sh"]

  backup:
    image: mirror-demo
    container_name: mirror-demo-backup
    networks:
      app_net:
        ipv4_address: ${BACKUP_APP_NET_IP}
      mirror_net:
        ipv4_address: ${BACKUP_MIRROR_NET_IP}
    ports:
      - 82:52773
    environment:
      - IRIS_MIRROR_ROLE=backup
      - WEBGATEWAY_IP=${WEBGATEWAY_IP}
      - BACKUP_MIRROR_NET_IP=${BACKUP_MIRROR_NET_IP}
      - MASTER_APP_NET_IP=${MASTER_APP_NET_IP}
      - BACKUP_APP_NET_IP=${BACKUP_APP_NET_IP}
    volumes:
      - ./backup:/opt/backup
      - ./init_mirror.sh:/init_mirror.sh
      # Mount certificates
      - ./certificates/backup_server.cer:/certificates/backup_server.cer
      - ./certificates/backup_server.key:/certificates/backup_server.key
      - ./certificates/CA_Server.cer:/certificates/CA_Server.cer
      #- ~/iris.key:/usr/irissys/mgr/iris.key
    hostname: backup
    extra_hosts:
      - "master:${MASTER_APP_NET_IP}"
      - "report:${REPORT_APP_NET_IP}"
    cap_add:
      - NET_ADMIN
    command: ["-a", "/init_mirror.sh"]
  report:
    image: mirror-demo
    container_name: mirror-demo-report
    networks:
      app_net:
        ipv4_address: ${REPORT_APP_NET_IP}
      mirror_net:
        ipv4_address: ${REPORT_MIRROR_NET_IP}
    ports:
      - 83:52773
    environment:
      - IRIS_MIRROR_ROLE=report
      - WEBGATEWAY_IP=${WEBGATEWAY_IP}
      - MASTER_APP_NET_IP=${MASTER_APP_NET_IP}
      - REPORT_MIRROR_NET_IP=${REPORT_MIRROR_NET_IP}
      - REPORT_APP_NET_IP=${REPORT_APP_NET_IP}
    volumes:
      - ./backup:/opt/backup
      - ./init_mirror.sh:/init_mirror.sh
      # Mount certificates
      - ./certificates/report_server.cer:/certificates/report_server.cer
      - ./certificates/report_server.key:/certificates/report_server.key
      - ./certificates/CA_Server.cer:/certificates/CA_Server.cer
      #- ~/iris.key:/usr/irissys/mgr/iris.key
    hostname: report
    extra_hosts:
      - "master:${MASTER_APP_NET_IP}"
      - "backup:${BACKUP_APP_NET_IP}"
    cap_add:
      - NET_ADMIN
    command: ["-a", "/init_mirror.sh"]

networks:
  app_net:
    ipam:
      driver: default
      config:
        - subnet: "${APP_NET_SUBNET}"
  mirror_net:
    ipam:
      driver: default
      config:
        - subnet: "${MIRROR_NET_SUBNET}"
```

The docker-compose.yml contains a lot of environment variables. To see the resolved file type in terminal :

```
docker-compose config
```

Run containers

```
docker-compose up
```

Wait for each instance has a good mirror status:

- master node with status Primary.
- backup node with status Backup.
- report node with status Connected.

Finally, you should see these messages in docker logs:

```
mirror-demo-master | 01/09/22-11:02:08:227 (684) 1 [Utility.Event] Becoming primary m
irror server
...
mirror-demo-backup | 01/09/22-11:03:06:398 (801) 0 [Utility.Event] Found MASTER as pr
imary, becoming backup
...
mirror-demo-report | 01/09/22-11:03:10:745 (736) 0 [Generic.Event] MirrorClient: Conn
ected to primary: MASTER (ver 4)
```

You can also just check the mirror status with the portal http://localhost:81/csp/sys/utilhome.csp

Access to portals

In Docker-compose we map ports 81,82, and 83 to have an access to each management portal.
This is the default login \password for all instances:

- Master http://localhost:81/csp/sys/utilhome.csp
- Failover backup member http://localhost:82/csp/sys/utilhome.csp
- Read-Write report async member http://localhost:83/csp/sys/utilhome.csp

Test

Check the mirror monitor (management porta; this is the default user and password.):
http://localhost:81/csp/sys/op/%25CSP.UI.Portal.Mirror.Monitor.zen

Verify the mirror settings :
http://localhost:81/csp/sys/mgr/%25CSP.UI.Portal.Mirror.EditFailover.zen?$NAMESPACE=%25SYS

We can start a test by simply setting a global starting by demo.
Remember that we have configured a global mapping demo.* on namespace USER.

Open a terminal session on the primary server:

```
docker exec -it mirror-demo-master irissession iris
```

```
Set ^demo.test = $zdt($h,3,1)
```

Check if the data is available on the backup node:

```
docker exec -it mirror-demo-backup irissession iris
```

```
Write ^demo.test
```

Check if the data is available on report node :

```
docker exec -it mirror-demo-report irissession iris
```

```
Write ^demo.test
```

Good! We have a mirror environment ready, fully created programmatically.
To be a bit more complete, we should add a web gateway with https and encryption between the web gateway and IRIS, but we will leave it for the next article.

Hope this article will be useful for you if you decide to create your own script.

Source

The content of this article is inspired by:

- @Dmitry Maslennikov iris-mirror-with-docker
- @Evgeny Shvarov docker template intersystems-community/objectscript-docker-template
- @Pete Greskoff article creating-ssl-enabled-mirror-intersystems-iris-using-public-key-infrastructure-pki
- @Robert Cemper IRIS easy ECP workbench

#DevOps #Mirroring #InterSystems IRIS
Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/how-set-mirror-programmatically