Object Storage Quickstart

This section explains how to get started with the object storage module in KDB-X.

Authentication

Authenticate with cloud credentials using Kurl in order to get native access to cloud object storage.

Access to files stored on cloud storage is differentiated by the following file prefixes:

```
aws     `:s3://
gcp     `:gs://
azure   `:ms://
```

Objects can be accessed using the `[prefix]/[bucket]/[key-name]` format for file handles.

Module load

As a prerequisite, you must have the appropriate environment variables or credentials defined.

=== "Amazon Web Services"

```bash
export AWS_ACCESS_KEY_ID="… "
export AWS_SECRET_ACCESS_KEY="…"
export AWS_SESSION_TOKEN="… "
```

=== "Microsoft Azure"

```bash
export AZURE_STORAGE_SHARED_KEY=".."
export AZURE_STORAGE_ACCOUNT=".."
```

=== "Google Cloud Platform"

```bash
gcloud init
export GCP_TOKEN=$(gcloud auth print-access-token)
```


Initialize

To access the object storage module, run the following command:

```
.objstor:use`kx.objstor
.objstor.init[]
```

Sample request

For example, running `hcount` on an object with key `data/2025.02.02/tbl/b` in bucket `mybucket` within s3 would be:

```
hcount `:s3://mybucket/data/2025.02.02/tbl/b
```

Reading data

1. Start KDB-X with the object store module loaded.
2. Delve deeper into the buckets:

   === "Amazon Web Services"

   ```
    AWS_REGION must be set to eu-west-1

       # set AWS_REGION=eu-west-1 before starting `q`
       q)key`:s3://
       `s#`kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-data
       q)key`$":s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-
   data/"
       `s#`_inventory`db`par.txt`sym`symlinks
       q)key`$":s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-
   data/db/"
       `s#`2020.01.01`2020.01.02`2020.01.03`2020.01.06`2020.01.07...
       q)key`$":s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-
   data/db/2020.01.01/trade/"
       `s#`.d`cond`ex`price`size`stop`sym`time
       q)get`$":s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-
   data/db/2020.01.01/trade/.d"
       `sym`time`price`size`stop`cond`ex
   ```

   === "Microsoft Azure"

   ```
       # set AZURE_STORAGE_ACCOUNT=kxinsightsmarketplace
       # set AZURE_STORAGE_SHARED_KEY to value returned by below
       # az storage account keys list --account-name kxinsightsmarketplace --
   ```



```
resource-group kxinsightsmarketplace
     q)key`:ms://
     ,`data
     q)key`$":ms://data/"
     `s#`_inventory`db`par.txt`sym
     q)key`$":ms://data/db/"
     `s#`2020.01.01`2020.01.02`2020.01.03`2020.01.06`2020.01.07...
     q)key`$":ms://data/db/2020.01.01/trade/"
     `s#`.d`cond`ex`price`size`stop`sym`time
     q)get`$":ms://data/db/2020.01.01/trade/.d"
     `sym`time`price`size`stop`cond`ex
```

=== "Google Cloud Platform"

```
     q)key`:gs://
     `s#`kxinsights-marketplace-data
     q)key`$":gs://kxinsights-marketplace-data/"
     `s#`_inventory`db`par.txt`sym
     q)key`$":gs://kxinsights-marketplace-data/db"

`s#`2020.01.01`2020.01.02`2020.01.03`2020.01.06`2020.01.07`2020.01.08`2020.0
1..
     q)key`$":gs://kxinsights-marketplace-data/db/2020.01.01/trade/"
     `s#`.d`cond`ex`price`size`stop`sym`time
     q)get`$":gs://kxinsights-marketplace-data/db/2020.01.01/trade/.d"
     `sym`time`price`size`stop`cond`ex
```

3. Other read operations work as if the file were on block storage.

=== "Amazon Web Services"

```
     ```q
     q)hcount `$":s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-
marketplace-data/db/2020.01.01/trade/sym"
     2955832

     q)-21!`$":s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-
data/db/2020.01.01/trade/sym"
     compressedLength  | 69520
     uncompressedLength| 2955832
     algorithm         | 2i
     logicalBlockSize  | 17i
     zipLevel          | 6i

     q)read1`$":s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-
data/db/2020.01.01/trade/.d"
```


```

```
0xff010b000000000073796d0074696d650070726963650073697a650073746f7000636f6e64
0..
```

=== "Microsoft Azure"

```q
q)hcount `$":ms://data/db/2020.01.01/trade/sym"
2955832

q)-21!`$":ms://data/db/2020.01.01/trade/sym"
compressedLength  | 69520
uncompressedLength| 2955832
algorithm         | 2i
logicalBlockSize  | 17i
zipLevel          | 6i

q)read1`$:ms://data/db/2020.01.01/trade/.d"

0xff010b000000000073796d0074696d650070726963650073697a650073746f7000636f6e64
0..
```

=== "Google Cloud Platform"

```q
q)hcount `$":gs://kxinsights-marketplace-data/db/2020.01.01/trade/sym"
2955832

q)-21!`$":gs://kxinsights-marketplace-data/db/2020.01.01/trade/sym"
compressedLength  | 69520
uncompressedLength| 2955832
algorithm         | 2i
logicalBlockSize  | 17i
zipLevel          | 6i

q)read1`$":gs://kxinsights-marketplace-data/db/2020.01.01/trade/.d"

0xff010b000000000073796d0074696d650070726963650073697a650073746f7000636f6e64
0..
```

Cloud data

In this section, learn how to set up cloud access and query cloud data.

Setup

Public datasets are available for AWS, Azure, and GCP.

Choose the provider that matches your environment.

=== "Google Cloud Platform"

```
    # set AZURE_STORAGE_ACCOUNT=kxinsightsmarketplace
    # set AZURE_STORAGE_SHARED_KEY to value returned by below
    # az storage account keys list --account-name kxinsightsmarketplace --
resource-group kxinsightsmarketplace
    $ gsutil ls gs://kxinsights-marketplace-data/
    gs://kxinsights-marketplace-data/sym
    gs://kxinsights-marketplace-data/db/

    $ gsutil ls gs://kxinsights-marketplace-data/db/ | head -n 5
    gs://kxinsights-marketplace-data/db/2020.01.01/
    gs://kxinsights-marketplace-data/db/2020.01.02/
    gs://kxinsights-marketplace-data/db/2020.01.03/
    gs://kxinsights-marketplace-data/db/2020.01.06/
    gs://kxinsights-marketplace-data/db/2020.01.07/
```

=== "Amazon Web Services"

```
    # set AWS_REGION=eu-west-1 before starting `q`
    $ aws s3 ls s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-data/
    PRE db/
    2021-03-10 21:19:33      42568 sym

    $ aws s3 ls s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-
data/db/ | head -n 5
    PRE 2020.01.01/
    PRE 2020.01.02/
    PRE 2020.01.03/
    PRE 2020.01.06/
    PRE 2020.01.07/
```

=== "Microsoft Azure"

```
    $ az storage blob list --account-name kxinsightsmarketplace \
      --container-name data | jq -r '.[] | .name' | tail -n 5
    db/2020.12.30/trade/size
    db/2020.12.30/trade/stop
    db/2020.12.30/trade/sym
    db/2020.12.30/trade/time
    sym
```


Create a sample using your selected cloud provider:

=== "Google Cloud Platform"

```
    $ mkdir ~/db
    $ gsutil cp gs://kxinsights-marketplace-data/sym ~/db/
    $ tee ~/db/par.txt << EOF
    > gs://kxinsights-marketplace-data/db
    > EOF
```

=== "Amazon Web Services"

```
    $ mkdir ~/db
    $ aws s3 cp s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-
data/sym ~/db/
    $ tee ~/db/par.txt << EOF
    > s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-data/db
    > EOF
```

=== "Microsoft Azure"

```
    $ mkdir ~/db
    $ # example commands below:
    $ az storage blob download --account-name kxinsightsmarketplace \
      --container-name data --name sym --file sym
    $ tee ~/db/par.txt << EOF
    > ms://data/db
    > EOF
```

This creates a standard HDB root directory where the partition in use is object storage.

Do **not** use trailing `/` on the object store location in `par.txt`.

```
$ tree ~/db/
/home/user/db/
├── par.txt
└── sym
```

Querying data

Run `q` on this directory:


```
$ ls
par.txt  sym
$ q /home/user/db/

q).objstor:use`kx.objstor
q).objstor.init[]
```

You should now be able to run queries:

```
q)tables[]
`s#`quote`trade
q)select count i by date from quote
date      | x
----------| ---------
2018.09.04| 692639728
2018.09.05| 762152767
2018.09.06| 788482304
2018.09.07| 801891891
2018.09.10| 635192966
```

Network speed to cloud storage can limit performance, so it's helpful to create a cache using fast SSDs, local NVMe drives, or even shared memory.

If you set these environment variables before starting `q`, the cache is enabled. The example below shows how caching improves the speed of subsequent requests by using `\t`:

```
$ export KX_OBJSTR_CACHE_PATH=/dev/shm/cache/
$ ls
par.txt  sym
$ q /home/user/db
KDB+ 4.0 2021.06.12 Copyright (C) 1993-2021 Kx Systems
l64/ 20()core 64227MB XXXXXX XXXXXXXXXX 127.0.1.1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX KXCE XXXXXXXX

q).objstor:use`kx.objstor
q).objstor.init[]
q)\t select count i by date from quote
4785
q)\t select count i by date from quote
0
```

Run the `kxreaper` application to prune the cache automatically if it gets full. The following examples show how to run Kxreaper to monitor the cache directory and limit the space used to 10000MB:

```
$ kxreaper "$KX_OBJSTR_CACHE_PATH" 10000 &
```


Using an S3-compatible object store

In addition to AWS, Azure, and GCP, the object storage module works with **S3-compatible services** such as MinIO.

This is useful if you want to test locally or deploy against a private object store.

At minimum, you need to set two environment variables so that KDB-X can connect:

```
export KX_S3_ENDPOINT=http://localhost:9000
export KX_S3_USE_PATH_REQUEST_STYLE=1
```

Once set, you can reference the bucket using the `:s3://` prefix in `par.txt` or directly in queries, just like native S3.

Refer to the examples below for a full walkthrough with MinIO.

Performance considerations

Cloud storage has higher latency. Use caching, secondary threads, compression, and inventory files to improve performance.

Caching

Due to the high latency of cloud storage, KDB-X offers the ability to configure a cache to locally store the requests results on high-performance disk.

Cloud vendors charge for object storage as a combination of volume stored, per retrieval request, and volume egress. Using the built-in compression and the cache can help to reduce these costs.

Secondary threads

The way to achieve concurrency with these high-latency queries is with secondary threads, through the command line option `-s`. It is expected that the larger the number of secondary threads, irrespective of CPU core count, the better the performance of object storage. Conversely the performance of cached data appears to be better if the secondary-thread count matches the CPU core count. A balance is to be found. We expect in future to improve the thread usage for these requests.

Compression

Due to the cost of storage, possible egress costs, high-latency and low bandwidth, we recommend storing data on cloud object storage using compression.

Metadata load times

Metadata load times for a HDB process can be improved by adding an `inventory file` to the storage account.

The file must be gzipped JSON, as an array of `{Key:string,Size:int}` objects. For example:


```
[
  {
    "Key": "db/2020.12.30/trade/size",
    "Size": 563829
  },
  {
    "Key": "db/2020.12.30/trade/stop",
    "Size": 49731
  },
  {
    "Key": "db/2020.12.30/trade/sym",
    "Size": 69520
  },
  {
    "Key": "db/2020.12.30/trade/time",
    "Size": 1099583
  }
]
```


The inventory file can be created and uploaded to the storage account using the following commands.

=== "Amazon Web Services"

````
```shell-session
aws --output json s3api list-objects --bucket kxs-prd-cxt-twg-
roinsightsdemo/kxinsights-marketplace-data --prefix 'db/' --query 'Contents[].
{Key: Key, Size: Size}' | gzip -9 -c > aws.json.gz

aws s3 cp aws.json.gz s3://kxs-prd-cxt-twg-roinsightsdemo/kxinsights-marketplace-
data/_inventory/all.json.gz
```
````

=== "Microsoft Azure"

````
```shell-session
az storage blob list --account-name kxinsightsmarketplace   --container-name data
| jq  '.[] | {Key: .name , Size: .properties.contentLength }' | jq -s '.' | gzip
-9 -c > azure.json.gz

az storage blob upload --account-name kxinsightsmarketplace \
      --container-name data --name _inventory/all.json.gz --file azure.json.gz
```
````

=== "Google Cloud Platform"


```shell-session
gsutil ls -lr gs://kxinsights-marketplace-data/db/*/*/* | awk '{printf "{ \"Key\": \"%s\" , \"Size\": %s }\n", $3, $1}' | head -n -1 | jq -s '.' | sed 's/gs:\/\/kxinsights-marketplace-data\/db\///g' | gzip -9 -c > gcp.json.gz

gsutil cp gcp.json.gz gs://kxinsights-marketplace-data/_inventory/all.json.gz
```

User can control which file is used as inventory via env var

```
export KX_OBJSTR_INVENTORY_FILE=_inventory/all.json.gz
```

The reading of the inventory file bypasses the cache, and to avoid cache invalidation issues, is not readable explicitly.

Symbolic links

Symbolic links can be simulated through an optional "Path" field in an entry, which represents the realpath that the Key should resolve to. The **Path** must reside in the same bucket, and the **Size** field must represent that of the **Path**. For example:

```
[
  {
    "Key": "db/2020.12.30/trade/size",
    "Size": 563829,
    "Path": "newdb/2020.12.30/trade/size"
  },
  {
    "Key": "db/2020.12.30/trade/stop",
    "Size": 49731,
    "Path": "newdb/2020.12.30/trade/stop"
  },
  {
    "Key": "db/2020.12.30/trade/sym",
    "Size": 69520,
    "Path": "newdb/2020.12.30/trade/sym"
  },
  {
    "Key": "db/2020.12.30/trade/time",
    "Size": 1099583,
    "Path": "newdb/2020.12.30/trade/time"
  }
]
```

Object Storage and KDB-X Examples

This section provides example code snippets to help you use the object storage module on KDB-X.

As a prerequisite, you must have your cloud vendor CLI installed to run some of the commands.

=== "Amazon Web Services"

```
[AWS Command-Line Interface](https://aws.amazon.com/cli/)
```

=== "Microsoft Azure"

```
[Azure CLI](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli)
```

=== "Google Cloud Platform"

```
[Google Cloud SDK](https://cloud.google.com/sdk/)
```

Creating data on Cloud Storage

In order for data to be migrated to the cloud, it must first be staged locally on a POSIX filesystem. This is because KX Insights Core does not support writing to cloud storage using the traditional `set` and other write functions.

To migrate the below database to a cloud storage account using a sample database created in kdb+:

```
q)d:2021.09.01+til 20
q){[d;n]sv[`;.Q.par[`:test/db/;d;`trade],`]set .Q.en[`:test/;
([]sym:`$'n?.Q.A;time:.z.P+til n;price:n?100f;size:n?50)];}[;10000]each d
```

This creates the following structure:

```
test/.
├── db
│   ├── 2021.09.01
│   ├── 2021.09.02
│   ├── 2021.09.03
│   ├── 2021.09.04
│   ├── 2021.09.05
│   ├── 2021.09.06
│   ├── 2021.09.07
│   ├── 2021.09.08
│   ├── 2021.09.09
│   ├── 2021.09.10
```



```
│   ├── 2021.09.11
│   ├── 2021.09.12
│   ├── 2021.09.13
│   ├── 2021.09.14
│   ├── 2021.09.15
│   ├── 2021.09.16
│   ├── 2021.09.17
│   ├── 2021.09.18
│   ├── 2021.09.19
│   └── 2021.09.20
└── sym
```

The below functions can be used to create a storage account and copy the database to the newly created storage account

=== "Amazon Web Services"

````
Documentation provided [here]
(https://awscli.amazonaws.com/v2/documentation/api/latest/reference/s3/index.html#
directory-and-s3-prefix-operations)

For example

```shell-session
## create bucket
aws s3 mb s3://mybucket --region us-west-1

## copy database to bucket
aws s3 cp test/* s3://mybucket/ --recursive
```
````

=== "Microsoft Azure"

````
Documentation provided [here](https://docs.microsoft.com/en-
us/azure/storage/blobs/storage-quickstart-blobs-cli)

For example

```shell-session
## create bucket
az group create --name <resource-group> --location <location>
az storage account create \
    --name <storage-account> \
    --resource-group <resource-group> \
    --location <location> \
    --sku Standard_ZRS \
    --encryption-services blob
````


````

```
az ad signed-in-user show --query objectId -o tsv | az role assignment create \
    --role "Storage Blob Data Contributor" \
    --assignee @- \
    --scope "/subscriptions/<subscription>/resourceGroups/<resource-
group>/providers/Microsoft.Storage/storageAccounts/<storage-account>"

az storage container create \
    --account-name <storage-account> \
    --name <container> \
    --auth-mode login

## copy database to bucket
az storage blob upload \
    --account-name <storage-account> \
    --container-name <container> \
    --name helloworld \
    --file helloworld \
    --auth-mode login
```

=== "Google Cloud Platform"

````
Documentation provided [here](https://cloud.google.com/storage/docs/creating-
buckets#storage-create-bucket-gsutil)

For example

```shell-session
## create bucket
gsutil mb -p PROJECT_ID -c STORAGE_CLASS -l BUCKET_LOCATION -b on gs://BUCKET_NAME

## copy database to bucket
gsutil cp -r OBJECT_LOCATION gs://DESTINATION_BUCKET_NAME/
```
````

Deleting data from Cloud Storage

Deleting data on Cloud Storage should be a rare occurrence but in the event that such a change is needed, the below steps should be followed

- Offline any hdb reader processes that are currently using the storage account
- Remove any caches created by the kxreaper application
- Delete the data from the storage account using

=== "Amazon Web Services"


```
[Deleting objects]
(https://awscli.amazonaws.com/v2/documentation/api/latest/reference/s3api/delete-
object.html)
```

=== "Microsoft Azure"

```
[Deleting objects](https://docs.microsoft.com/en-us/cli/azure/storage/blob?
view=azure-cli-latest#az-storage-blob-delete)

[Batch Deleting objects](https://docs.microsoft.com/en-us/cli/azure/storage/blob?
view=azure-cli-latest#az-storage-blob-delete-batch)
```

=== "Google Cloud Platform"

```
[Deleting objects](https://cloud.google.com/storage/docs/deleting-objects#gsutil)
```

- Recreate the inventory file(if used)
- Get the reader processes online making sure they are reloaded to pick up the new inventory file and drop any metadata caches using the drop command

Changing data on Cloud Storage

Altering data e.g. changing types, adding columns etc will require the same steps as deleting data. Once the reader processes have been taken offline, the changes will be able to happen safely bearing in mind that in order to change data, it will first need to be copied from the storage account, amended and the copied back to the appropriate path using a cloud cli copy command.

Creating inventory JSON

Refer to the metadata load times in the quickstart above for instructions on how to create and use the inventory file.

Combining Cloud and Local Storage in a single HDB

The addition of the object store library allows clients to extend their tiering strategies to cloud storage. In some instances, it is necessary to query data that has some partitions on a local POSIX filesystem and other partitions on cloud storage. To give a kdb+ process access to both datasets, the par.txt can be set as follows:

=== "Amazon Web Services"

```
```shell-session
s3://mybucket/db
/path/to/local/partitions
```


```

```
```

Note: if multiple storage accounts are added they must be in the same AWS region.
```

=== "Microsoft Azure"

```
```shell-session
ms://mybucket/db
/path/to/local/partitions
```
```

=== "Google Cloud Platform"

```
```shell-session
gs://mybucket/db
/path/to/local/partitions
```
```

Note that multiple local filesystems and storage accounts can be added to par.txt.

## S3 compatible object store example

This example demonstrates how data can be stored and queried from any object store that's compatible with the S3 interface.

The example uses Docker to create a local object store using the MinIO implementation of an S3-compatible server.

**Note:** "Prerequisites"

```
The AWS CLI must be installed to work through this example.
```

1. Start the MinIO server using Docker or Podman.

   ```
   docker run -it --rm -p 9000:9000 -p 9001:9001 -e "MINIO_ROOT_USER=<insert
   minio user>" -e "MINIO_ROOT_PASSWORD=<insert minio password>" minio/minio
   server /data --console-address ":9001"
   ```

2. Set the AWS credentials by executing `aws configure`. The AWS Access Key ID should be set to the `MINIO_ROOT_USER` and AWS Secret Access Key to `MINIO_ROOT_PASSWORD` (refer to the command above).

3. Create some test data using KDB-X into the directory `./test`


   ```

```
q)d:2021.09.01+til 20
q){[d;n]sv[`;.Q.par[`:test/db/;d;`trade],`]set .Q.en[`:test/;
([]sym:`$'n?.Q.A;time:.z.P+til n;price:n?100f;size:n?50)];}[;10000]each d
```

4. Create a bucket in MinIO called `test`

```
aws --endpoint-url http://127.0.0.1:9000 s3 mb s3://test
```

5. Copy the generated test data to the `test` bucket

```
aws --endpoint-url http://127.0.0.1:9000 s3 sync test/db s3://test
aws --endpoint-url http://127.0.0.1:9000 s3 ls s3://test
```

6. Create a HDB root directory called `dbroot` that uses data from the `test` bucket

```
mkdir dbroot
cp test/sym dbroot/
echo "s3://test" > dbroot/par.txt
```

7. Start a `q` process as below, loading the `dbroot` directory that uses the `test` bucket. Environment variables are set to change the queries to use the MinIO server.

```
export KX_S3_ENDPOINT=http://127.0.0.1:9000
export KX_S3_USE_PATH_REQUEST_STYLE=1
q dbroot -q
```

8. Query the data in the bucket, which automatically retrieves the required data from the S3-compatible object store (MinIO)

```
q).objstor:use`kx.objstor
q).objstor.init[]
q)tables[]
,`trade
q)select count sym from trade
sym
------
200000
q)5#select count sym by date from trade
date      | sym
----------| -----
2021.09.01| 10000
```



```
2021.09.02| 10000
2021.09.03| 10000
2021.09.04| 10000
2021.09.05| 10000
```

To see which URLs are used in the query, `export KX_TRACE_OBJSTR=1` and restart the `q` process.

Object Store Reference

This section documents URIs, environment variables, APIs, and configuration for the `objstor` module.

URI schema

Object storage is accessed using **URI-style prefixes** that identify the provider:

- `:s3://` — Amazon S3 buckets
- `:ms://` — Microsoft Azure Blob Storage
- `:gs://` — Google Cloud Storage

The prefix tells KDB-X which cloud provider and authentication method to use.

Note: S3-compatible services (such as MinIO) are also supported by combining the `:s3://` prefix with a custom endpoint.

.ojbstor.init

This API will initialize values based on the environment variables set. If no parameter it passed, it'll attempt to register with each of the 3 cloud vendors: AWS, Azure and GCP

Input parameters

Name	Type	Description	Required	Default
vendor	list	Options include aws, azr, gcp, none or leave empty to register with 3 cloud vendors	No	All

As a prerequisite, you must have the appropriate environment variables defined.

=== "Amazon Web Services"

````
To register with AWS, you can define the environment variables below, or populate
your credentials file. Typically this is ~/.aws/credentials

```bash
export AWS_ACCESS_KEY_ID="… "
export AWS_SECRET_ACCESS_KEY="…"
export AWS_SESSION_TOKEN="… "
```
````


=== "Microsoft Azure"

```bash
export AZURE_STORAGE_SHARED_KEY=".."
export AZURE_STORAGE_ACCOUNT=".."
```

=== "Google Cloud Platform"

```bash
gcloud init
export GCP_TOKEN=$(gcloud auth print-access-token)
```

Examples

```
.objstor.init[]; // register with AWS, Azure and GCP
.objstor.init`aws;
.objstor.init`none;
```

Environment variables

`KX_TRACE_OBJSTR`

: The URIs requested to the cloud are printed to STDERR if the following environment variable is set. e.g.

```
    export KX_TRACE_OBJSTR=1
```

`KX_OBJSTR_INVENTORY_FILE`

: Location of optional `inventory file`

Although the Kurl module can detect most of the necessary components for credentials from the environment, the following additional environment variables are required.

`AWS_REGION`

: If a region is not selected, then `us-east-1` is used by default.

`KX_S3_USE_PATH_REQUEST_STYLE`

: Alters the URL that is constructed for requests to S3 compatible storage.


```
By default, virtual host style is used. Virtual host style uses the bucket name to
construct the hostname from which data is requested. The virtual host style uses
the `https://[bucket-name].s3.[region].amazonaws.com/[key-name]` format for S3.
Setting `KX_S3_USE_PATH_REQUEST_STYLE=1` changes to path style, using the format
`https://s3.[region].amazonaws.com/[bucket-name]/[key-name]` where all requests go
to the same host. This is often used in S3-compatible object storage that are
alternatives to Amazon S3 (for example, MinIO).
```

`KX_S3_ENDPOINT`

: Changes the endpoint URL used for requests to S3 compatible storage, for example `KX_S3_ENDPOINT=https://play.min.io:9000`. When used with `KX_S3_USE_PATH_REQUEST_STYLE=1`, requests use the format `[KX_S3_ENDPOINT]/[bucket-name]/[key-name]`. This is often used in S3-compatible object storage that are alternatives to Amazon S3 (for example, MinIO).

`AZURE_STORAGE_ACCOUNT`

: The DNS prefix for your storage account; e.g. for `mystorage.blob.core.windows.net` the name would be `mystorage`. The list of your storage accounts can be displayed using the Azure CLI tool `az`:

```
    az storage account list | jq -r '.[] | .name'
```

`GCLOUD_PROJECT_ID`

: A unique, user-assigned ID that can be used as the request header `x-goog-project-id` in Google APIs. It is used as a request header that specifies which project you are working on. It may be any valid project number or name. This request header tells Cloud Storage which project to create a bucket in or which project to list buckets for. Examples:

```
    000111222333
    my-project-name
    example.com:my-google-apps-for-work-project-name

The list of your projects can be displayed using the Google CLI tool, Gcloud, via

    gcloud projects list
```

