TA-azure-blob-archiving Documentation

Release 1

Guilhem Marchand

Jan 18, 2022

Contents

This Add-on provides a robust and smart archiving framefork solution for Splunk Enterprise and Azure blob storage.

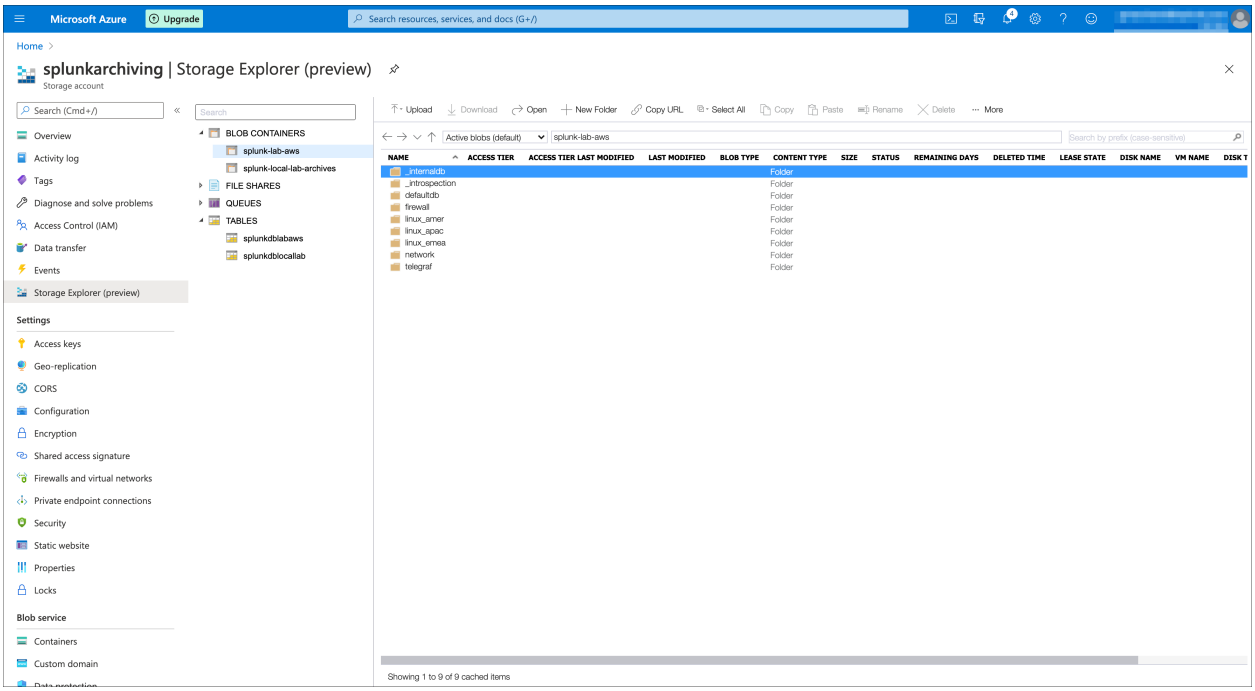
It relies on the Splunk built-in archiving capabilities and Azure blob storage and tables via the usage of the Python SDK for Azure:

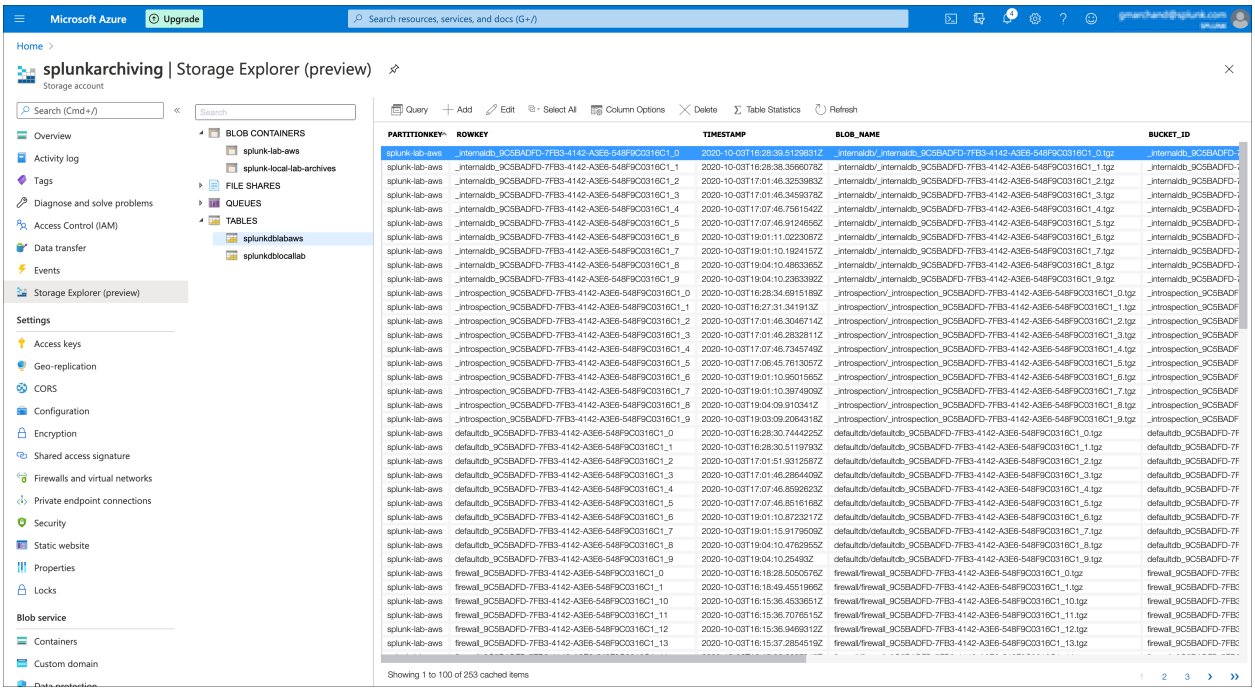
Splunk Documentation links:

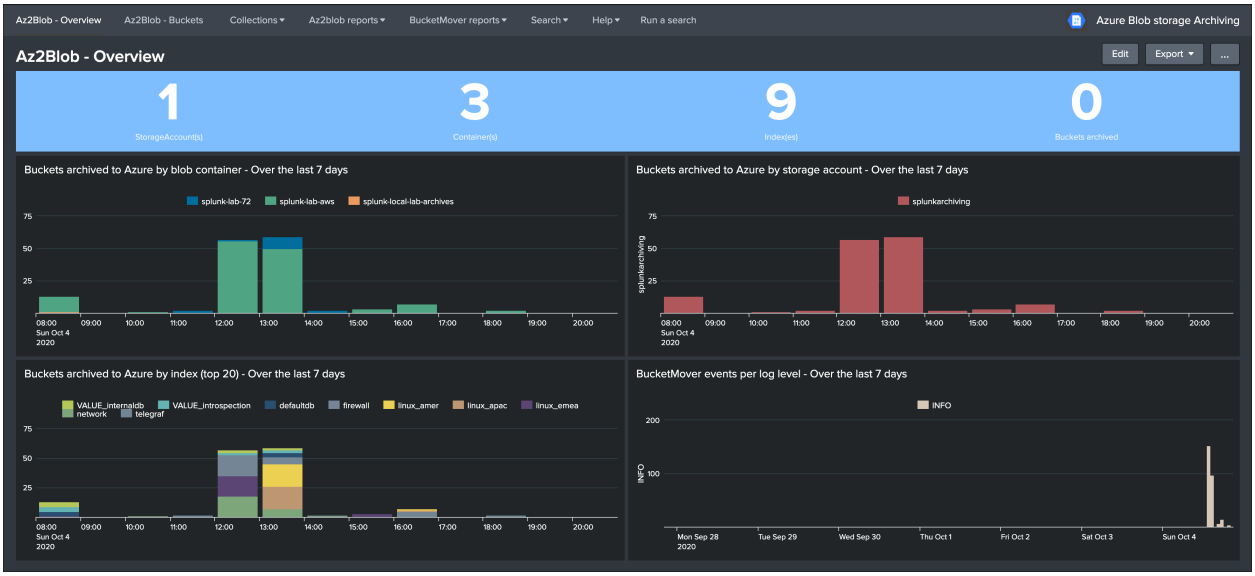
- https://docs.splunk.com/Documentation/Splunk/latest/Indexer/Automatearchiving
- https://docs.splunk.com/Documentation/Splunk/latest/Indexer/Setaretirementandarchivingpolicy

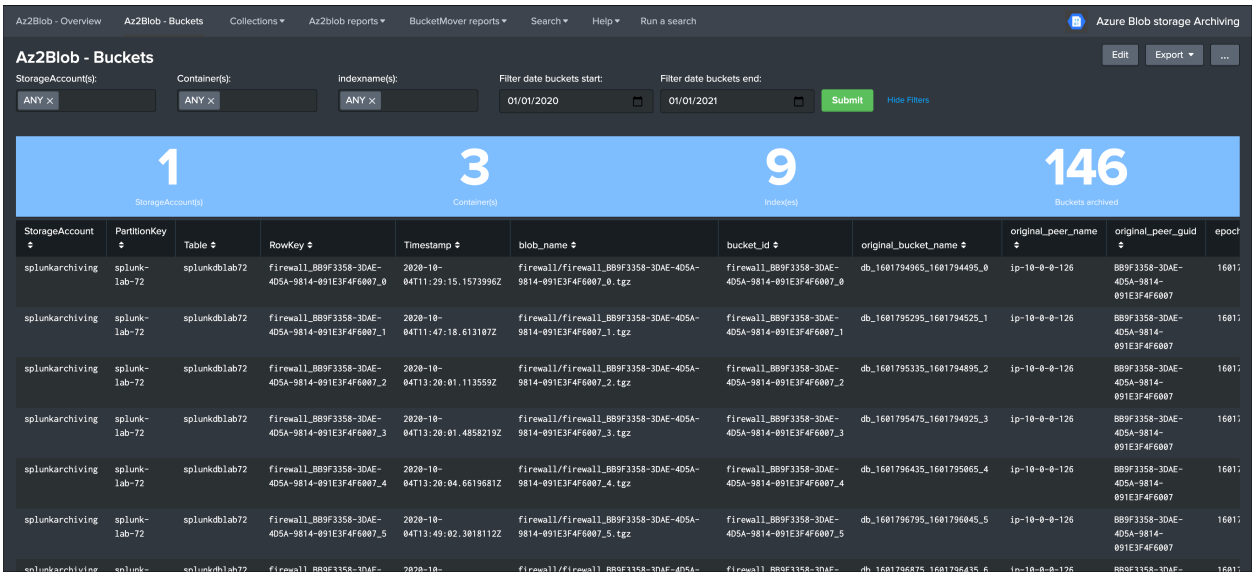
Azure links:

- https://azure.github.io/azure-sdk/releases/latest/python.html
- https://docs.microsoft.com/en-us/python/api/?view=azure-python
- https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-portal



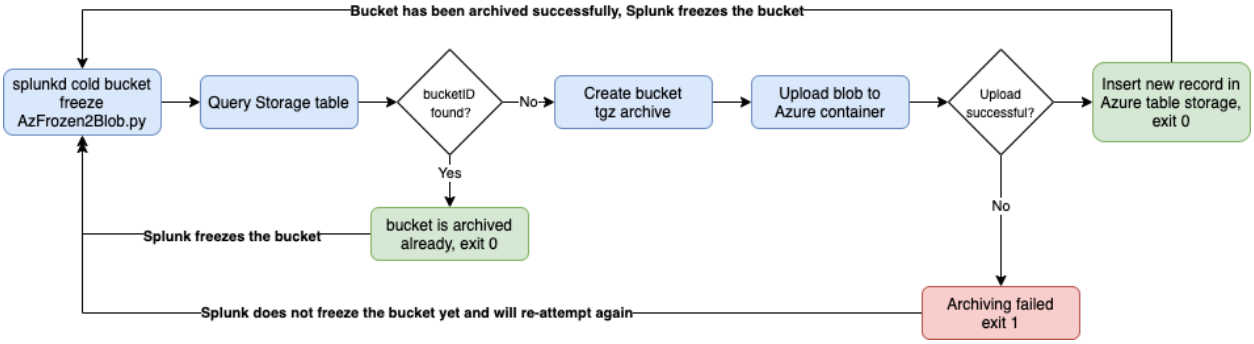

Microsoft Azure
Upgrade
Search resources, services, and docs (G+/)
Home
splunkarchiving | Storage Explorer (preview)
Storage account
Search (Cmd+/)
Overview
Activity log
Tags
Diagnose and solve problems
Access Control (IAM)
Data transfer
Events
Storage Explorer (preview)
Settings
Access keys
Geo-replication
CORS
Configuration
Encryption
Shared access signature
Firewalls and virtual networks
Private endpoint connections
Security
Static website
Properties
Locks
Blob service
Containers
Custom domain
BLOB CONTAINERS
splunk-lab-aws
splunk-local-lab-archives
FILE SHARES
QUEUES
TABLES
splunkdblabaws
splunkdblocallab
Query
Add
Edit
Select All
Column Options
Delete
Table Statistics
Refresh
PARTITIONKEY
ROWKEY
TIMESTAMP
BLOB_NAME
BUCKET_ID
Showing 1 to 100 of 253 cached items


Az2Blob - Overview
Az2Blob - Buckets
Collections
Az2blob reports
BucketMover reports
Search
Help
Run a search
Azure Blob storage Archiving
Az2Blob - Overview
Edit
Export
1
StorageAccount(s)
3
Container(s)
9
Index(es)
0
Buckets archived
Buckets archived to Azure by blob container - Over the last 7 days
splunk-lab-72
splunk-lab-aws
splunk-local-lab-archives
Buckets archived to Azure by storage account - Over the last 7 days
splunkarchiving
Buckets archived to Azure by index (top 20) - Over the last 7 days
VALUE_internaldb
VALUE_introspection
defaultdb
firewall
linux_amer
linux_apac
linux_emea
network
telegraf
BucketMover events per log level - Over the last 7 days
INFO



The framework and concept can be summarised the following way:

- Splunk automatically calls the AzFrozen2Blob.py Python script when a bucket is frozen from cold storage (assuming archiving is enabled on the index)
- The Python script accesses an Azure storage account and verifies in a pre-defined Azure storage table if that bucket ID has been archived already (management of buckets replication for Splunk indexers in cluster)
- If the bucket has not been archived yet, a tgz archive of the bucket is created and uploaded to the pre-defined container in Azure blob
- If the upload to blob is successful, the Python script inserts a new record in the Azure storage table with all the useful information related to this bucket
- If the upload is successful, the script exists with an error code=0 which instructs Splunk that the bucket can be frozen, otherwise the script exit=1 and a new attempt will be made automatically by Splunk



CHAPTER 1

Overview:

1.1 About

- Author: Guilhem Marchand, Splunk certified consultant and part of Splunk Professional Services
- First public release published in October 2020
- License: Apache License 2.0

1.2 Compatibility

1.2.1 Python compatibility

This application requires a Python 3 interpreter to perform archiving to Azure blob storage and the interractions with Azure storage tables.

1.2.2 Splunk compatibility

This application is compatible with Splunk 7.0.x and later.

1.2.3 Web Browser compatibility

The application can be used with any of the supported Web Browser by Splunk:

https://docs.splunk.com/Documentation/Splunk/latest/Installation/Systemrequirements

1.3 Support & donate

I am supporting my applications for free, for the good of everyone and on my own private time. As you can guess, this is a huge amount of time and efforts.

If you enjoy it, and want to support and encourage me, buy me a coffee (or a Pizza) and you will make me very happy!

This application is community supported.

To get support, use of one the following options:

1.3.1 Splunk Answers

Open a question in Splunk Community:

- https://community.splunk.com/

1.3.2 Splunk community slack

Contact me on Splunk community slack, and even better, ask the community!

- https://splunk-usergroups.slack.com

1.3.3 Open a issue in Git

To report an issue, request a feature change or improvement, please open an issue in Github:

- https://github.com/guilhemmarchand/TA-azure-blob-archiving/issues

1.3.4 Email support

- guilhem.marchand@gmail.com

However, previous options are far betters, and will give you all the chances to get a quick support from the community of fellow Splunkers.

1.4 Download

The Splunk application can be downloaded from:

1.4.1 Splunk base

- https://splunkbase.splunk.com/app/5274

1.4.2 GitHub

- https://github.com/guilhemmarchand/TA-azure-blob-archiving

CHAPTER 2

Deployment and configuration:

2.1 Deployment

2.1.1 Deployment matrix

Splunk roles	required
Search head	yes*
Indexer tiers	Yes

Search Head deployment is only required if you intend to use the front-end part of the application

If Splunk search heads are running in Search Head Cluster (SHC), the Splunk application must be deployed by the SHC deployer.

For Splunk indexers in cluster, the Splunk application must be deployed via the Splunk Cluster Master.

2.1.2 Dependencies

Search Head(s)

The front-end part of the application relies on indexing the content of the Azure storage tables via the Splunk Add-on for Microsoft Cloud Services:

- https://splunkbase.splunk.com/app/3110/
- https://docs.splunk.com/Documentation/AddOns/released/MSCloudServices/Configureinputs4

Search head(s) do not have direct interractions with Azure storage blob or tables, and do not need to satisfy any additional dependencies.

In a distributed deployment content, you would most likely deploy the Splunk Add-on for Microsoft Services on a heavy forwarder layer that you use for data collection purposes.

Indexer(s)

Azure blob storage archiving and table interractions happen on the indexer level, each indexer needs to have the following dependencies satisfied:

- A Python 3 interpreter must be available on the Operating Systen level (Out of Splunk space, the Add-on does not use the embedded Python interpreter that comes with Splunk)
- Azure SDK for Python must be deployed and available to the user name owning the Splunk processed (usually named splunk)

Azure SDK for Python

There are two SDKs used by the Addon:

- https://pypi.org/project/azure-storage-blob/
- https://pypi.org/project/azure-cosmosdb-table/

You can install the SDKs via pip:

```
sudo pip3 install azure-storage-blob
sudo pip3 install azure-cosmosdb-table
```

Depending on the context, you may prefer to run the pip module installation only for the user that owns the Splunk processes:

```
sudo su - splunk
pip3 install azure-storage-blob
pip3 install azure-cosmosdb-table
```

In some systens, you may need to install the modules with root permissions, see the first option.

You may as well install manually the Python modules instead of using pip if you cannot use it (but pip is strongly recommended), follow the PYpi links, download the packages, and run the installer as the splunk user.

Once you installed the Azure SDKs, you can very easily verify that the modules can be imported successfully:

- Open a Python3 interpreter
- Verify that you can import the Azure SDK modules:

```
from azure.storage.blob import BlobClient, BlobServiceClient
```

```
from azure.cosmosdb.table.tableservice import TableService
```

See bellow:

Connect to an indexer via SSH:

```
ubuntu@mylab:~$ sudo su - splunk
splunk@mylab:~$ which python3
/usr/bin/python3
splunk@mylab:~$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from azure.storage.blob import BlobClient, BlobServiceClient
>>> from azure.cosmosdb.table.tableservice import TableService
>>>
```

If the import is successful as the example above, the dependencies are statisfied successfully.

Do not continue if you are failing to import any of the two modules, until you fix the issue.

2.1.3 Initial deployment

The deployment of the Splunk application follows the usual process:

- By using the application manager in Splunk Web (Settings / Manages apps) for standalone instances
- Or by extracting the content of the tgz archive in the "apps" directory of Splunk
- For SHC configurations (Search Head Cluster), extract the tgz content in the SHC deployer and publish the SHC bundle
- For indexer in cluster deployment, extract the tgz content in the cluster master in master-apps and pubish the cluster bundle

2.1.4 Upgrades

Upgrading the Splunk application is the same operation than the initial deployment, extracting from a new release tgz will override any component that is built-in into the application.

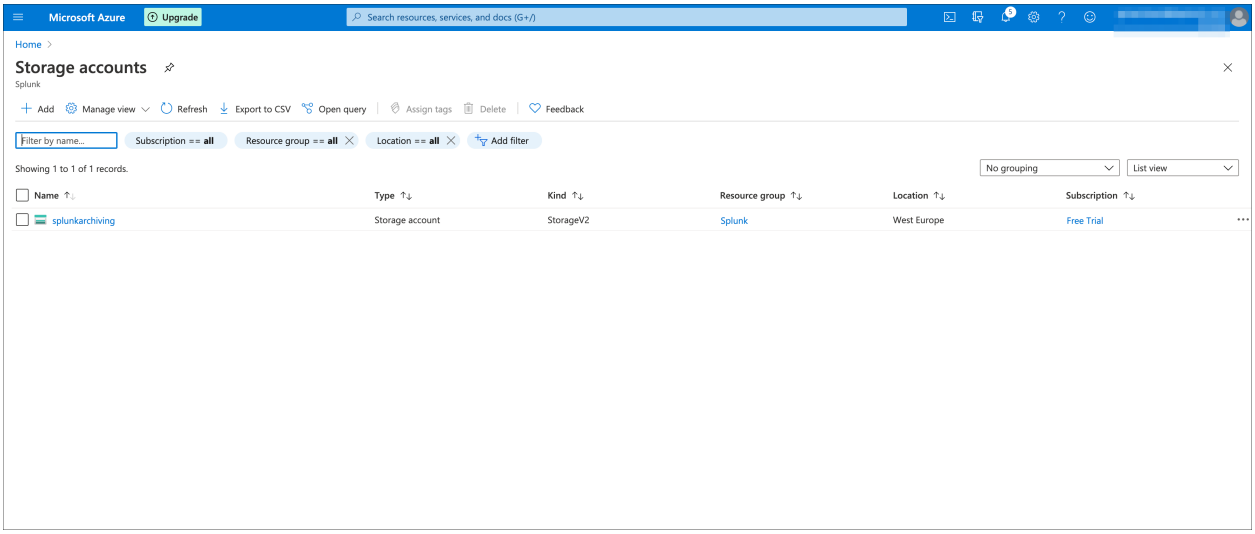
2.2 Configuration

2.2.1 Azure storage account Configuration

The first thing required is to have a storage account configured in Azure which will be used for the Splunk archiving blob storage and tables.

Follow the Azure documentation if you do not have a storage account yet:

- https://docs.microsoft.com/en-us/azure/storage/common/storage-account-create?tabs=azure-portal



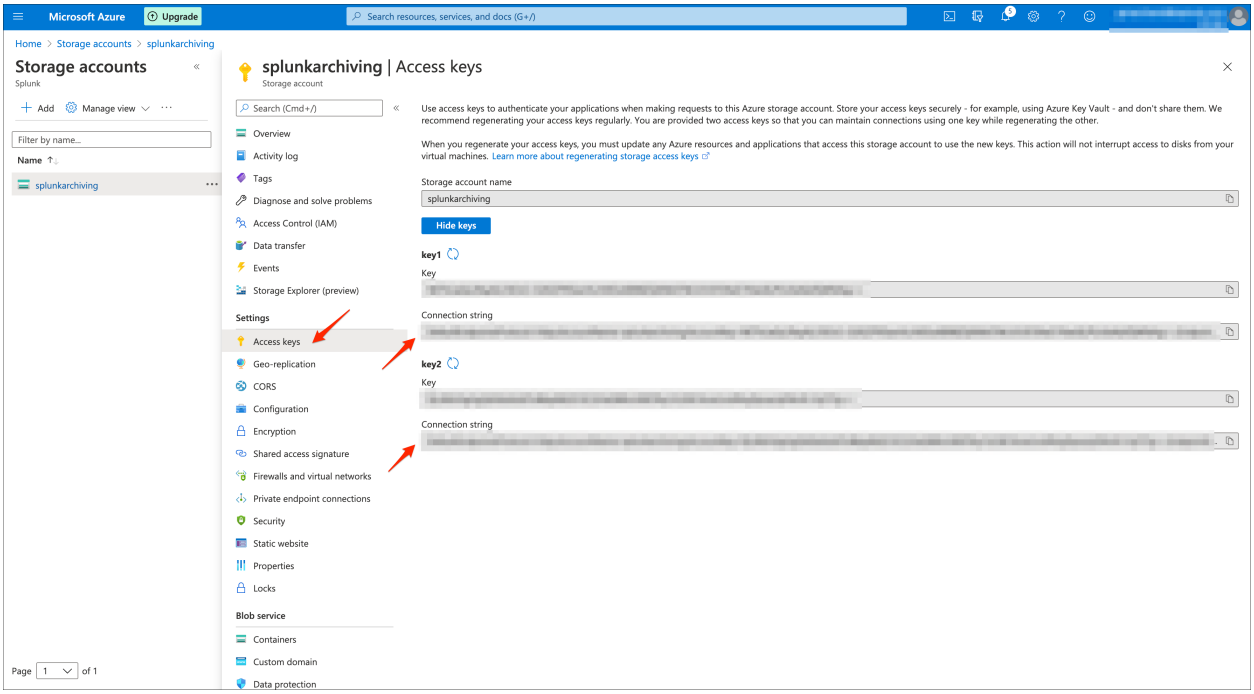
2.2.2 Azure storage account connection string

Once you have a storage account, the next things you need to retrieve are the following information to configure the Add-on:

You need to know the connection string for your user account, this information will be stored on the indexers in the configuration file local/azure2blob.conf:

```
AZ_BLOB_CONNECTION_STRING = connection_string_to_the_blob_storage
```

The connection string provides all the information required for the access to Azure blob storage and tables via the SDKs, you can find your connection string in Settings / Access keys in the Azure storage account portal:



You can use any of the two connection strings provided by Azure, store this value as you will configure it in the local/azure2blob.conf file.

2.2.3 Azure storage blob container

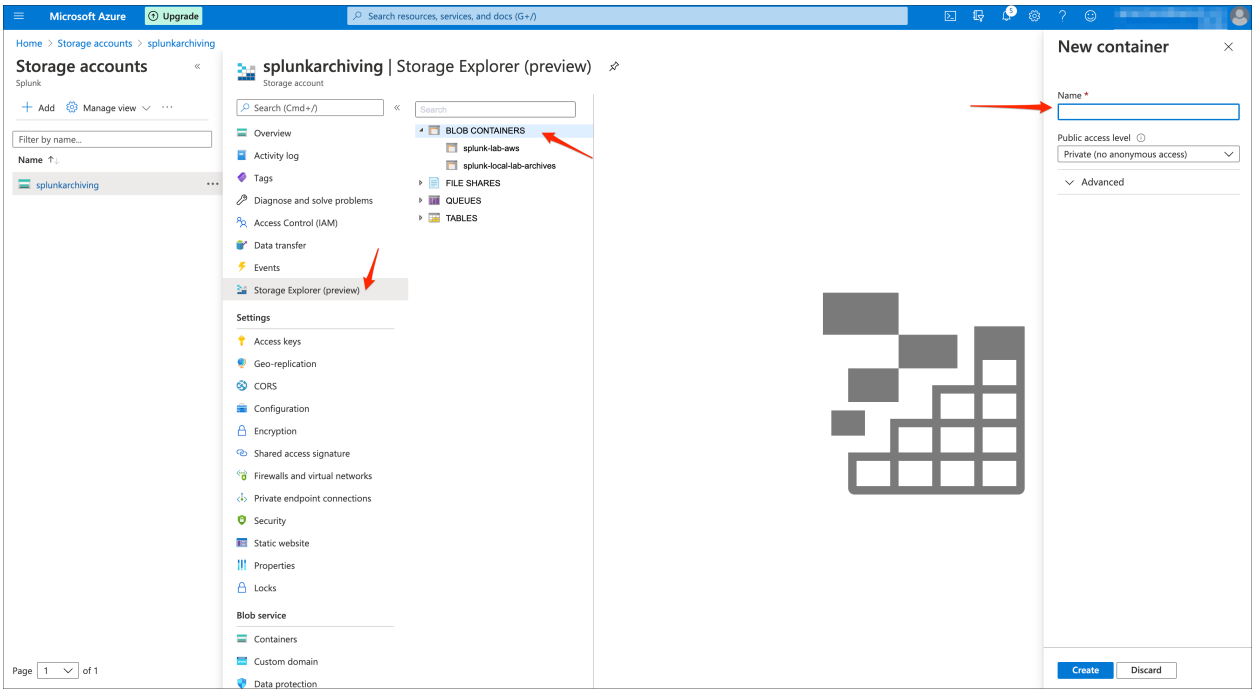
Decide what the container name will be to store the buckets archived from Splunk indexers, by default the container will be:

```
AZ_BLOB_CONTAINER = splunk-cold2frozen-archives
```

You can change this value to anything that suits your needs, if you have multiple environments to be archived in the same storage account, you likely will want to change this value to include the name of the environment.

Finally, it is `recommended` that you create the container manually in the Azure portal, however if you do not the Python backend will attempt to `create it automatically`.

Note: This name may only contain lowercase letters, numbers, and hyphens, and must begin with a letter or a number. Each hyphen must be preceded and followed by a non-hyphen character. The name must also be between 3 and 63 characters long.



2.2.4 Azure storage table

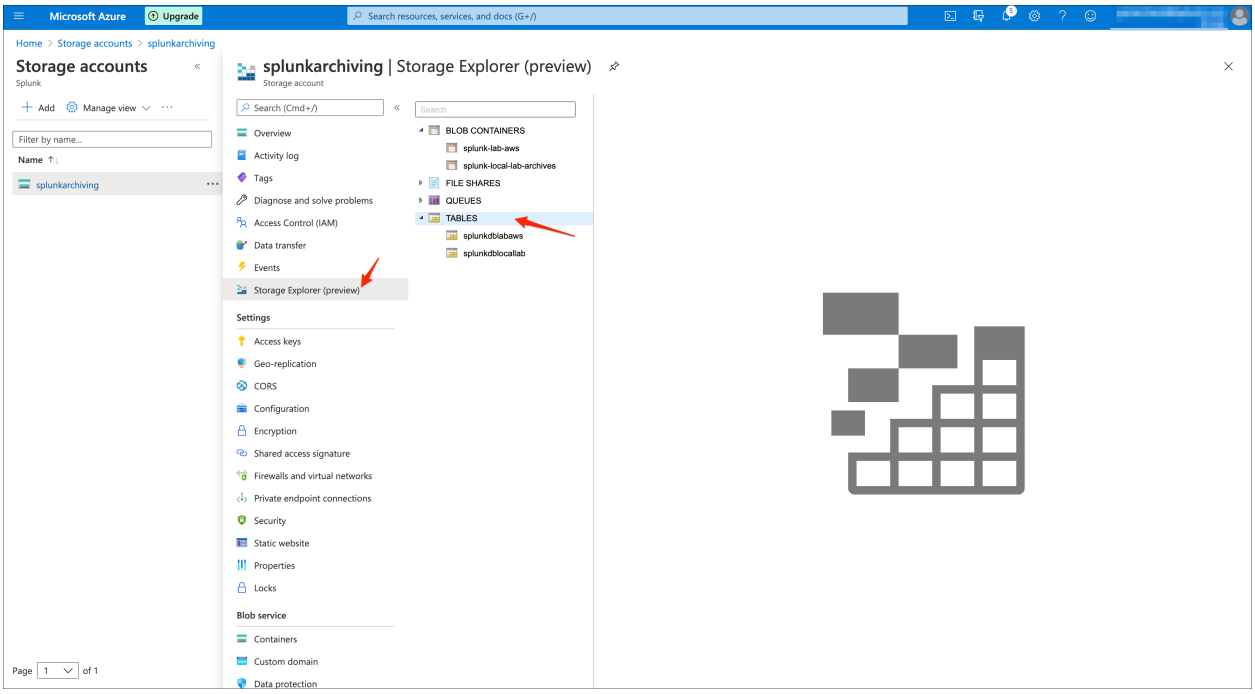
Decide what the Azure storage table name will be, by default the table name will be:

```
AZ_STORAGE_TABLE_NAME = splunkdb
```

You can change this value to anything that suits your needs, if you have multiple environments to be archived in the same storage account, you likely will want to change this value to include the name of the environment.

Finally, it is `recommended` that you create the table manually in the Azure portal, however if you do not the Python backend will attempt to `create it automatically`.

Note: able names must be alphanumeric, cannot begin with a number, and must be between 3 and 63 characters long.



2.2.5 Splunk indexer(s) configuration

Now that you have all information required, configure the Add-on local configuration, for instance when running in indexer clusters you have:

```
cd /opt/splunk/etc/master-apps
ls -ltrd TA-azure-blob-archiving
```

Create a local directory and copy default/azure2blob.conf:

```
mkdir TA-azure-blob-archiving/local
cp -p TA-azure-blob-archiving/default/azure2blob.conf TA-azure-blob-archiving/local/
```

Edit the file TA-azure-blob-archiving/local/azure2blob.conf and update the values according to your account and settings:

```
[azure2blob]
AZ_BLOB_CONTAINER = splunk-cold2frozen-archives
AZ_BLOB_CONNECTION_STRING = connection_string_to_the_blob_storage
AZ_STORAGE_TABLE_NAME = splunkdb
```

Finally, publish the cluster bundle, once the bundle is pushed the indexers are ready to start archiving to Azure blob storage.

See *Manually testing archiving a bucket* to verify that your configuration is successful.

2.2.6 Splunk indexe(s) configuration to enable archiving

To enable Azure blob archiving, you need to configure your indexes.conf to include the coldToFrozenScript parameter:

For Splunk version prior to 8.0, it is mandatory to use the shell wrapper to avoid Python import troubles:

`Cluster of indexers:`

Example:

```
[firewall_emea]
coldToFrozenScript = "$SPLUNK_HOME/etc/slave-apps/TA-azure-blob-archiving/bin/
↪AzFrozen2Blob.sh"
```

Splunk instances starting 8.0 can directly call the Python backend:

```
[firewall_emea]
coldToFrozenScript = "/usr/bin/python3" "$SPLUNK_HOME/etc/slave-apps/TA-azure-blob-
↪archiving/bin/AzFrozen2Blob.py"
```

`standalone indexers:`

```
[firewall_emea]
coldToFrozenScript = "$SPLUNK_HOME/etc/apps/TA-azure-blob-archiving/bin/AzFrozen2Blob.
↪sh"
```

Splunk instances starting 8.0 can directly call the Python backend:

```
[firewall_emea]
coldToFrozenScript = "/usr/bin/python3" "$SPLUNK_HOME/etc/apps/TA-azure-blob-
↪archiving/bin/AzFrozen2Blob.py"
```

Notes:

- If the system level Python3 interpreter is not available in `/usr/bin/python3`, you can either change this location or create a symbolic link as a best practice
- If you cannot define the symbolic link to `/usr/bin/python3` and you are running a Splunk version prior to Splunk 8.0, you will need to update the Python path in `AzFrozen2Blob.sh` (CAUTION: this is not upgrade resilient! A much better practice is to fix the OS)
- If you are configuring a standalone indexer, change `slave-apps` to `apps`
- Repeat this operation for every index you need aarchiving to be enabled

2.2.7 Splunk Search Head(s) configuration

The Add-on relies on the Splunk Add-on for Microsoft Cloud Services to provide insight on the archiving of buckets in Splunk.

Make sure the Add-on was installed and configured (register the storage account):

- https://splunkbase.splunk.com/app/3110/

Then enable indexing the storage table in Splunk:

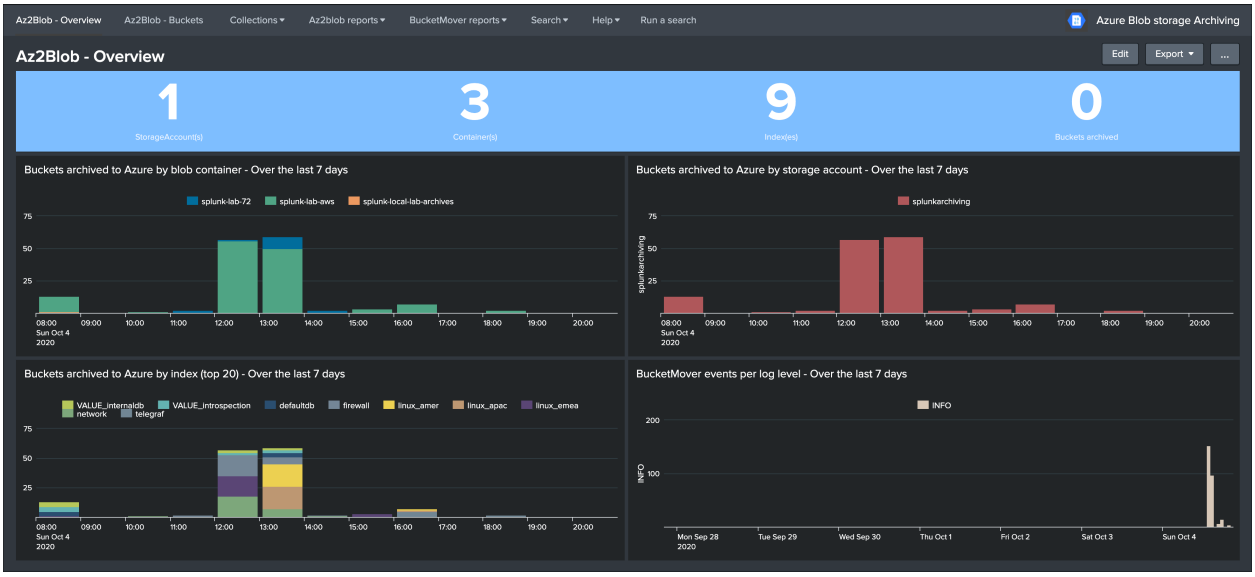
- https://docs.splunk.com/Documentation/AddOns/released/MSCloudServices/Configureinputs4

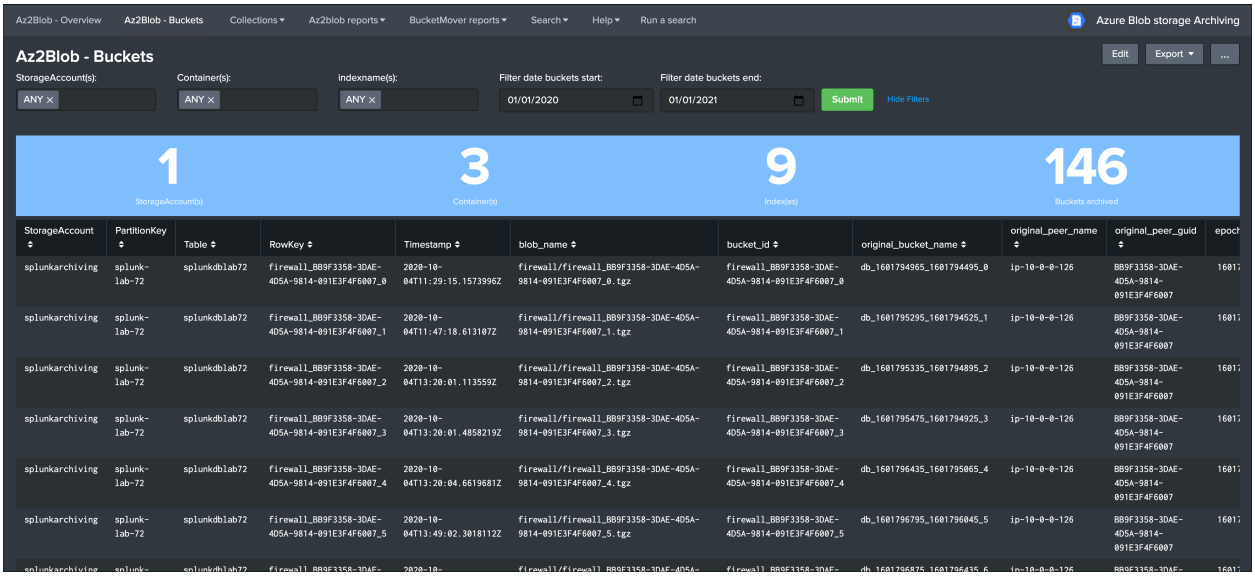
The Add-on used the following macro to define access to the Azure storage table data indexed in Splunk:

```
[az2blob_archive_root_search]
definition = index=* sourcetype="mscs:storage:table" source="*splunkdb*"
iseval = 0
```

Update this macro to match the index(es) where you are indexing the table data, and update the source constraint if it does not match your table naming convention.

Once you have started to index the Azure storage data, and if there has been buckets archived already, the UI will automatically expose the archives buckets information:





2.3 Tools

Additional command line tools are provided:

2.3.1 List blobs from containers

You can use the builtin Python script AzListBlob.py to list all blobs available in a container, run this command from any of the indexers:

```
AzListBlob.py
```

```
sudo su - splunk
cd /opt/splunk/etc/slaves-apps/TA-azure-blob-archiving/bin
export SPLUNK_HOME="/opt/splunk"
python3 AzListBlob.py
```

usage is returned:

```
usage: python3 AzListBlob.py <container_name>
```

Example of usage:

```
python3 AzListBlob.py splunk-local-lab-archives

#### Listing blobs... ####

linux_emea/linux_emea_F3AFBA7F-A0A7-4A91-97D0-753F5828B8BE_12.tgz
linux_emea/linux_emea_F3AFBA7F-A0A7-4A91-97D0-753F5828B8BE_15.tgz
...
```

2.3.2 Download blobs from containers (restore buckets)

You can use the builtin Python script AzDownloadBlob.py to retrieve and download a blob file from a container, which means retrieving the tgz archive of a bucket that was previously archived:

`AzDownloadBlob.py`

```
sudo su - splunk
cd /opt/splunk/etc/slaves-apps/TA-azure-blob-archiving/bin
export SPLUNK_HOME="/opt/splunk"
python3 AzDownloadBlob.py
```

usage is returned:

```
usage: python3 AzDownloadBlob.oy <container_name> <blob_name> <target_file>
```

Example of usage:

```
python3 AzDownloadBlob.py splunk-local-lab-archives linux_emea/linux_emea_F3AFBA7F-
↪A0A7-4A91-97D0-753F5828B8BE_12.tgz /tmp/linux_emea_F3AFBA7F-A0A7-4A91-97D0-
↪753F5828B8BE_12.tgz
container is splunk-local-lab-archives
blob_name is linux_emea/linux_emea_F3AFBA7F-A0A7-4A91-97D0-753F5828B8BE_12.tgz
target_file is /tmp/linux_emea_F3AFBA7F-A0A7-4A91-97D0-753F5828B8BE_12.tgz

Downloading blob to
        /tmp/linux_emea_F3AFBA7F-A0A7-4A91-97D0-753F5828B8BE_12.tgz
```

Once you have downloaded the blob, you can proceed to the extraction of the bucket in the thaweddb to restore the required data:

- https://docs.splunk.com/Documentation/Splunk/latest/Indexer/Restorearchiveddata

CHAPTER 3

Troubleshoot:

3.1 Troubleshoot

3.1.1 Manually testing archiving a bucket

You can test manually archiving a bucket to Azure blob the following way:

- Connect to an indexer via SSH
- Open a session as the user name owning the Splunk processes
- Identify a cold bucket you want to test for the archiving
- Use the following command to manually archive a bucket: (note: unless you manually set SPLUNK_HOME, to run an archiving manually you need to use the shell wrapper)

```
sudo su - Splunk
/opt/splunk/bin/splunk cmd /opt/splunk/etc/slave-apps/TA-azure-blob-archiving/bin/
↪AzFrozen2Blob.sh /opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_
↪88
```

Example of results:

```
bucket is /opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_88
idx_struct is <re.Match object; span=(0, 69), match='/opt/splunk/var/lib/splunk/
↪network/colddb/db_1601>
indexname is network
is it gz? True
is it zst? False
Archiving bucket: /opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_
↪88
/opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_88/Strings.data
/opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_88/bloomfilter
/opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_88/Sources.data
/opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_88/optimize.result
```

(continues on next page)

(continued from previous page)

```
/opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_88/bucket_info.csv
/opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_88/SourceTypes.data
/opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_88/.rawSize
/opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_88/1601751202-
↪1601751090-14340990455171772002.tsidx
/opt/splunk/var/lib/splunk/network/colddb/db_1601751202_1601751090_88/Hosts.data
peer_name is ip-10-0-0-75
bucket_name is db_1601751202_1601751090_88
bucket_epoch_start is 1601751090
bucket_epoch_end is 1601751202
bucket_id_list is ['88']
# This means it's a non replicated bucket, so need to grab the GUID from instance.cfg
original_peer_guid is 9C5BADFD-7FB3-4142-A3E6-548F9C0316C1
bucket_id is network_9C5BADFD-7FB3-4142-A3E6-548F9C0316C1_88
This bucket has not been archived yet, proceed to archiving now
the peer name is ip-10-0-0-75
Archive upload to Azure blob storage successful for bucket /opt/splunk/var/lib/splunk/
↪network/colddb/db_1601751202_1601751090_88
```

Any error will be clearly exposed in the output of the script.

If the archive operation is successful, the bucket tgz will be visible in the Azure portal, and a new record will have been created in the Azure table.

3.1.2 Investigating BuckerMover logs

Splunk traces the BucketMover logs in the _internal, as follows:

```
index=_internal sourcetype=splunkd BucketMover
```

Successful archiving appears as:

```
10-04-2020 13:38:41.653 +0000 INFO  BucketMover - will attempt to freeze bkt="/opt/
↪splunk/var/lib/splunk/linux_amer/colddb/db_1601744221_1601743940_85" reason="␣
↪maxTotalDataSize=104857600 bytes, diskSize=104902656 bytes"
10-04-2020 13:38:47.248 +0000 INFO  BucketMover - AsyncFreezer freeze succeeded for␣
↪bkt='/opt/splunk/var/lib/splunk/linux_amer/colddb/db_1601744221_1601743940_85'
```

If there are any failures from the Python backend, Splunk will log every trace from stderr, example:

Splunk search

```
index=_internal sourcetype=splunkd bucketmover error
```

Example of failure main message

```
10-04-2020 13:18:55.260 +0000 ERROR BucketMover - coldToFrozenScript cmd='"/opt/
↪splunk_72/splunk/etc/apps/TA-azure-blob-archiving/bin/AzFrozen2Blob.sh" /opt/splunk_
↪72/splunk/var/lib/splunk/network/colddb/db_1601796404_1601795154_4' exited with non-
↪zero status='exited with code 1'
```

For example if you attempt to run directly the Python backend on Splunk prior to Splunk 8.0, the following message would be visible:

```
10-04-2020 13:18:55.257 +0000 ERROR BucketMover - coldToFrozenScript ImportError:␣
↪This package should not be accessible on Python 3. Either you are trying to run␣
↪from the python-future src folder or your installation of python-future is␣
↪corrupted.
```

(continues on next page)

(continued from previous page)

The front-end part of the application provides buit-in dashboards and reports for this purpose.

CHAPTER 4

Versions and build history:

4.1 Release notes

4.1.1 Version 1.0.4

- Feature: Add archived volume Metadata to be stored in the Azure table, and used for analytic purposes in Splunk UIs

4.1.2 Version 1.0.3

- Change: minor improvement for the BucketMover errors report, use case sensitivy while search for ERROR patterns

4.1.3 Version 1.0.2

- Fix: buckets archived over last 7 days single form in Az2Blob - Overview dashboard remains at 0

4.1.4 Version 1.0.1

- Fix: spec file name is wrong

4.1.5 Version 1.0.0

- Initial version