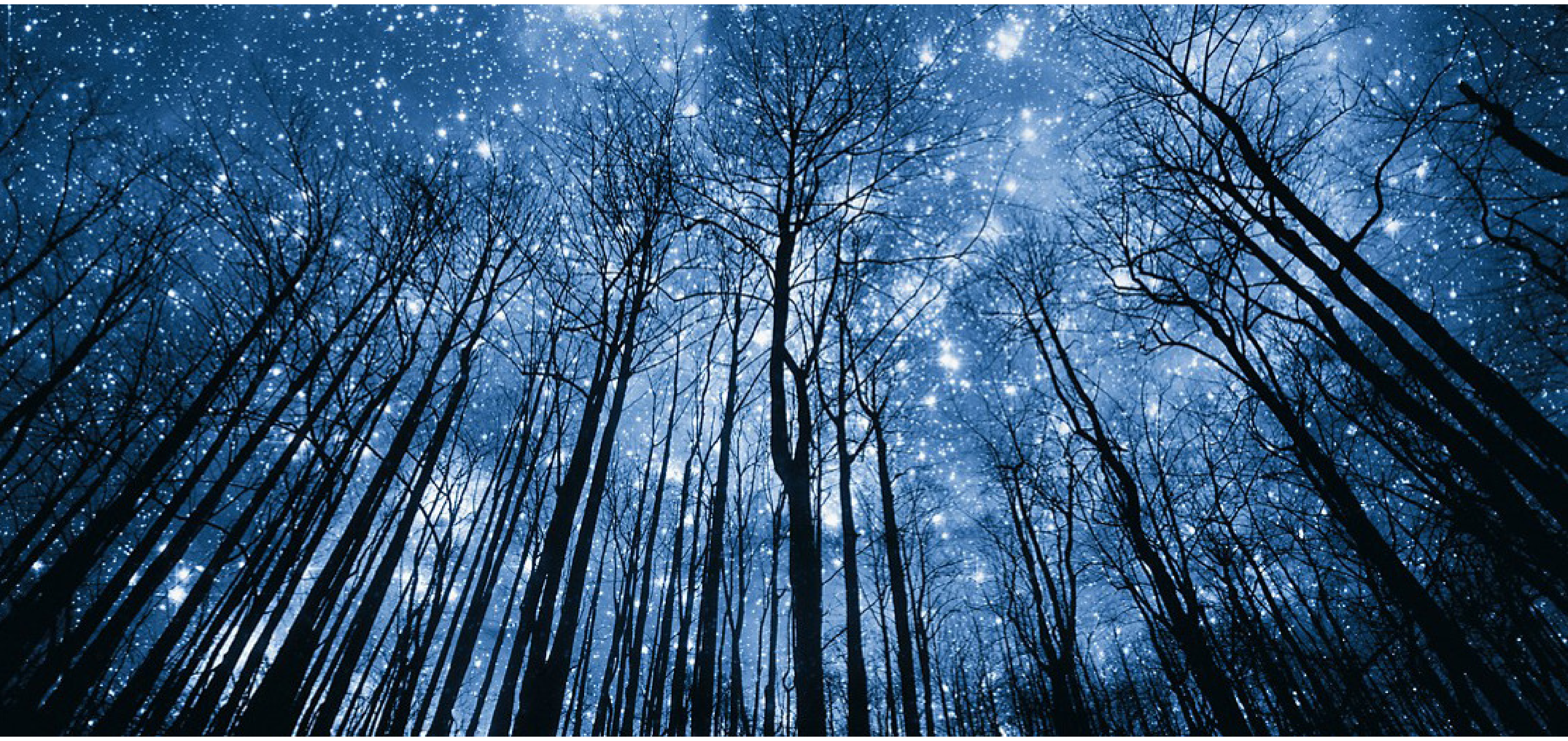
DATA PROTECTION SOLUTION FOR MULTICLOUDS



Pablo Calvo
Delivery Specialist
Dell Technologies
Pablo.calvo@dell.com

Mahmoud Elsayed
Senior Consultant
Dell Technologies
Mahmoud.elsayed@dell.com

David Cuesta
Principal Engineer
Dell Technologies
David.cuesta@dell.com

The Dell Technologies Proven Professional Certification program validates a wide range of skills and competencies across multiple technologies and products.

From Associate, entry-level courses to Expert-level, experience-based exams, all professionals in or looking to begin a career in IT benefit from industry-leading training and certification paths from one of the world's most trusted technology partners.

Proven Professional certifications include:

- Cloud
- Converged/Hyperconverged Infrastructure
- Data Protection
- Data Science
- Networking
- Security
- Servers
- Storage
- Enterprise Architect

Courses are offered to meet different learning styles and schedules, including self-paced On Demand, remote-based Virtual Instructor-Led and in-person Classrooms.

Whether you are an experienced IT professional or just getting started, Dell Technologies Proven Professional certifications are designed to clearly signal proficiency to colleagues and employers.

Learn more at www.dell.com/certification

Table of Contents

Introduction

Identifying the problem

Companies place a high priority on the value of their information. This information is allocated into different systems and managed in different ways.

Being safe from external attack or loss is one of the most important goals of companies.

With the new features and requirements of new technologies, many companies are moving their IT procedures to the public cloud as a solution where the resources are always available according to the customer necessities

These new solutions are deployed into different cloud solutions; sometimes, it is impossible to apply traditional backup techniques because this is the wrong way to solve it.

Industries that back off from on-prem and migrate their data to the Cloud reap tremendous rewards. However, many wrongly assume that their data is safe in terms of backup “data is stored in the Cloud, so that’s safe for recovery in case of data loss”. This is one of the most deceptive concepts of cloud computing. Just because data is hosted on the cloud doesn’t necessarily mean it can automatically be recovered during a disaster, user errors, application errors, ransomware, or other malicious activities resulting in data loss.

Most businesses also prefer the move to a multi-cloud operating model to maximize what different public clouds offer.

With every new cloud service, a training plan is required to enable the backup administrator team to handle it, which leads to a hassle.

All these preliminaries caused a need for a central cloud services solution that eliminates the need for new training every day and helps reduce the cost of additional cloud backup services.

Propose the use of modern technologies

Modern technologies enable faster deployment but require new solutions configured differently than traditional technologies.

Most new technologies are deployed over public clouds on Infrastructure as a Service, Software as a Service, or Application as a Service.

New technologies emerged and can significantly enhance automating the backup and deploying and installing the backup agents.

DevOps and GitOpts can help build a new automated cloud service backup solution framework. Also, technology such as containers can be quickly configured and deployed and used as backup building blocks for catalog-based backup solutions like Avamar.

Our solution

Public cloud backup of Infrastructure as a Service is done like the traditional method, but there was necessary to perform the backup of the other two ways, so our solution covers these gaps.

This solution installs and configures an auto-discover proxy backup solution with Avamar and Data Domain. The proxy is implemented in a container technology, taking advantage of portability, scaling and repeatability, and can be deployed by DevOps orchestration tool over Kubernetes or others. This proxy only requires a configuration file and script set up to deploy it. Proxy is considered another client for Avamar and Data Domain; for this reason, n it can be integrated into the flow of administration and control of protection tasks.

Terms

Use case: List of actions required to backup and restore a cloud resource.

Blob storage: Storage optimized for storing massive amounts of unstructured data.

Blobfuse, cifs-utils, mongodb-tools, postgresql and sqlpackage: Software used to interact with defined use cases

GitHub Repository: It contains project files including scripts, json files, and readme files.

DCI: Deploy Control Instance, Linux virtual machine used to create docker containers.

CVM: Container Virtual Machine, Linux virtual machine used to run docker containers.

Docker image: This is an immutable file that contains computational code and config files needed for running applications.

Docker container: A virtualized and isolated run-time environment where applications run.

Json File: A file used to interchange data.

Standard_B2s and t2.micro: An Azure or AWS VM.

AZ Cli and AWS Cli: Command line interfaces used to interact with Azure or AWS.

Trivy scan: Scanner for vulnerabilities in container images.

The high-level steps of the solution are:

(1) Retrieve Components: The solution will grab a repository of code with the following elements: scripts, dockerfile related to a specific Docker image.

(2) Deployment: The Docker server will deploy the specific Docker container with the previous elements.

(3) Service Catalog Queries: the backup solution will ask the current elements in the Azure subscription to get the configuration to be saved.

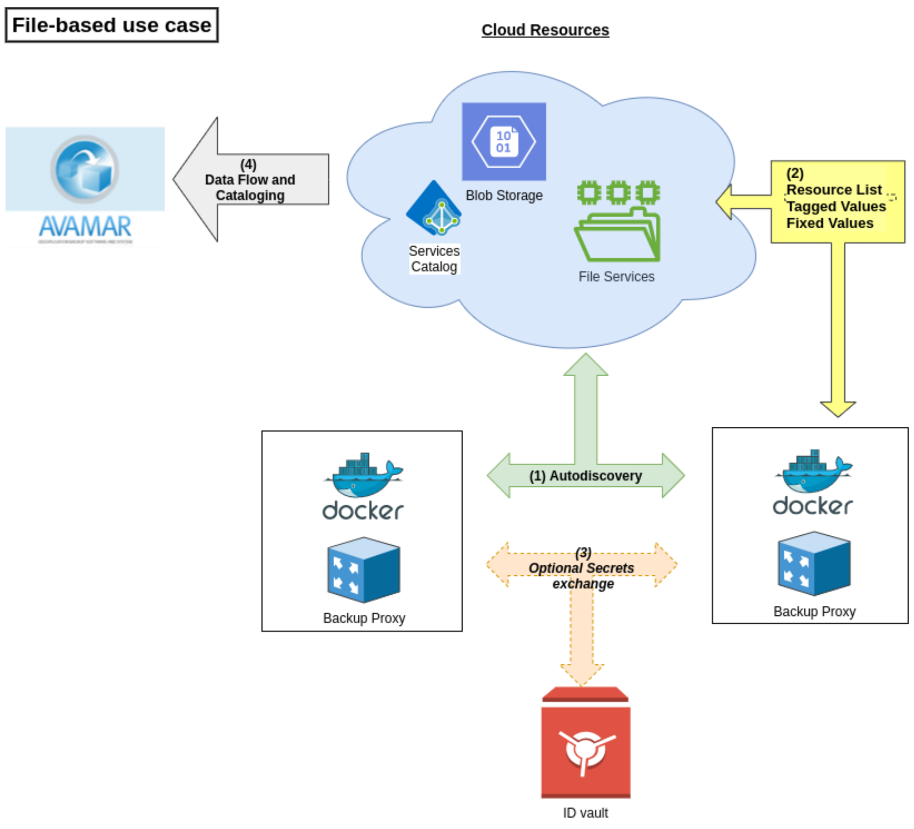
(4) Jobs: a scheduler will execute jobs in order to repeat the queries to the service catalog and preserve this using the Avamar file system plugin and the storage units located in Data Domain. By default, this scheduler is placed in the Avamar server. In the diagram, you can also see the movement of the data from Avamar and Data Domain.

File-based use case diagram

File-based use cases mount blob storage, cifs or nfs data as a regular file system using blobfuse or mount command. Avamar reads these files using a Linux plug-in. Blobfuse and Avamar clients reside in the Docker container.

This backup proxy is a rule-based engine used to discover cloud resources (1), resources are handled according to tagging or fixed values to decide which is backed or restored (2) or not, if the password is needed the proxy will access a key vault to get it (3). Backup data is sent to the backup server through the Avamar client (4).

File-base use cases are: Azure Data Lake Storage Gen 2 (**adls), **Azure **Blobstorage (blobstorage),** Azure Databricks **(databricks),** Azure Event Hubs **(eventhub),** Azure File Storage **(filestorage),** Azure HDInsight**,** Azure NetApp Storage **(netappstorage).** AWS s3 **(s3),** Azure Redis (**redis**)

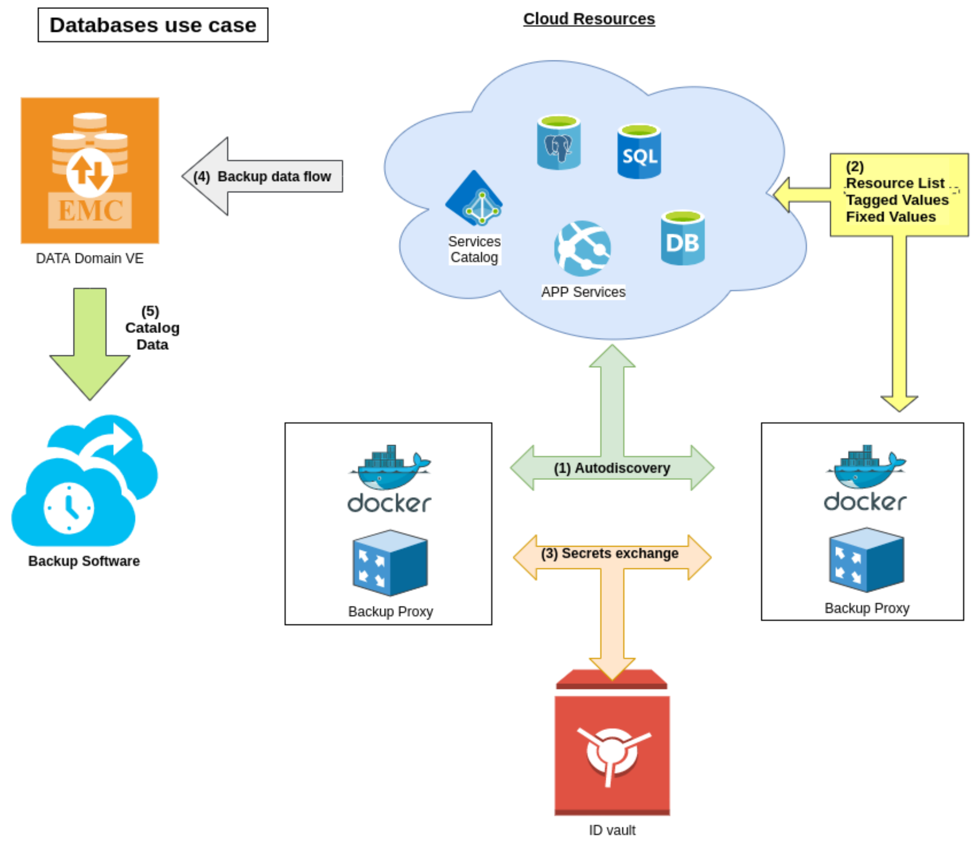


Database use case diagram

Database use cases dump exports data from source to a DDboost FS mounted Docker container. DD Boost FS client, Avamar client, use case-specific dump/export command reside in the Docker container.

Database use cases start discovering cloud resources (1) resources are handled according to tagging or fix values to decide which is backed up or restored (2) or not; if the password is needed the proxy will access a key vault to get it (3). Backup data is dumped/exported to the Data Domain through DD Boost protocol and sent to Avamar through the Avamar client (4).

File-base use cases are: Azure SQL **(azsql),** Azure Cosmo DocumentDB **(cosmosql),** Azure Datafactory **(datafactory),** Azure Cosmo MongoDB **(cosmosmg),** Atlas **DB (atlas),** Azure PostgreSQL DB **(postgresql)**



Special cases

Snapshot-based: AWS Elastic Seach **(elasticsearch)** ana AWS Redis **(redis)**
Local exports: Azure Key Vaults **(keyvault)**
Token-based: Azure Cognitive Services Custom Vision **(cvision)**

Infrastructure deployment

As support for the solution, two VMs must be deployed that will be used to generate the containers (DCI) and to execute them (CVM). The Docker containers will carry out the backup tasks of the different technologies (use cases).

See Appendix A to deploy DCI and CVM virtual machines in Azure or AWS.

Container deployment

The container will perform the following tasks:

- The first time it starts, it will register against the defined backup server. It will also register against the Data Domain server for database use cases.
- Learn which Azure cloud resources to support for the current subscription according to the technology to be protected, for example, PostgreSQL.
- Perform a database dump or export or mount the blob storage.
- Starts the backup job on Avamar.

These steps have been automated into the ***m-dps-setup.sh***[1] meta script explained below. Before doing any deployment, the JSON file must be filled with the prerequisite parameters.

m-dps-setup.sh usage

```
m-dps-setup.sh -d <docker type> -i <image> -v <version> -c <cloud_provider> -m
<mode> -a <Avamar port> -e <Ephimeral> -n <vm amount>
```

The options will be:

```
-d : docker type. You must select between the different technologies.
azsql|blobstorage|cosmosql|cvision|databriks|datafactory|adls|eventhub|filestorage|h
dinsight|keyvault|cosmosmg|atlas|netappstorage|postgresql|redis
-i: image to be used on the deployment. Must be loaded. You can see the loaded
images with command docker images
-v: Version of the image. By default we must use latest, but could be used any
loaded
-c: Cloud provider. Options: azure/aws
-m: Mode to be used for deployment. By default CLI will be used. Options: CLI/API
-a: Avamar port without last digit (e.g. 2804). See the table: Specific Ports Range
by integration
-e: About if the docker client VM will be ephemeral or not. Ephemeral means that the
docker client Vm only exists if the backup it's started. The NO option
only be used for test or high specific purposes. Options: yes/no
-n: Number of containers to deploy
```

Non ephemeral example for Azure SQL:

```
azsql: ./m-dps-setup.sh -d azsql -i centosaveddaz -v 1.0 -c azure -m CLI -a 2803 -e
no -n 1
```

Non ephemeral example or Azure SQL:

```
azsql: ./m-dps-setup.sh -d azsql -i centosaveddaz -v 1.0 -c azure -m CLI -a 2803 -e
yes -n 1
```

Non-ephemeral mode

Deploying more than one Docker of the same technology will require performing the following steps:

- After the first run of m-dps-setup, a new container azure-docker-<technology>-001 has been created (for example azure-docker-postgresql-001) and a Docker image with the same name.

- You can use the nonephemeral.sh[2] script if you want to create new containers from the same image.

- Being a non-ephemeral container, the entry point points to /bin/bash. You must connect to the container and run the /dockerclient/post_install.sh script to complete the registration against Avamar and Data Domain.

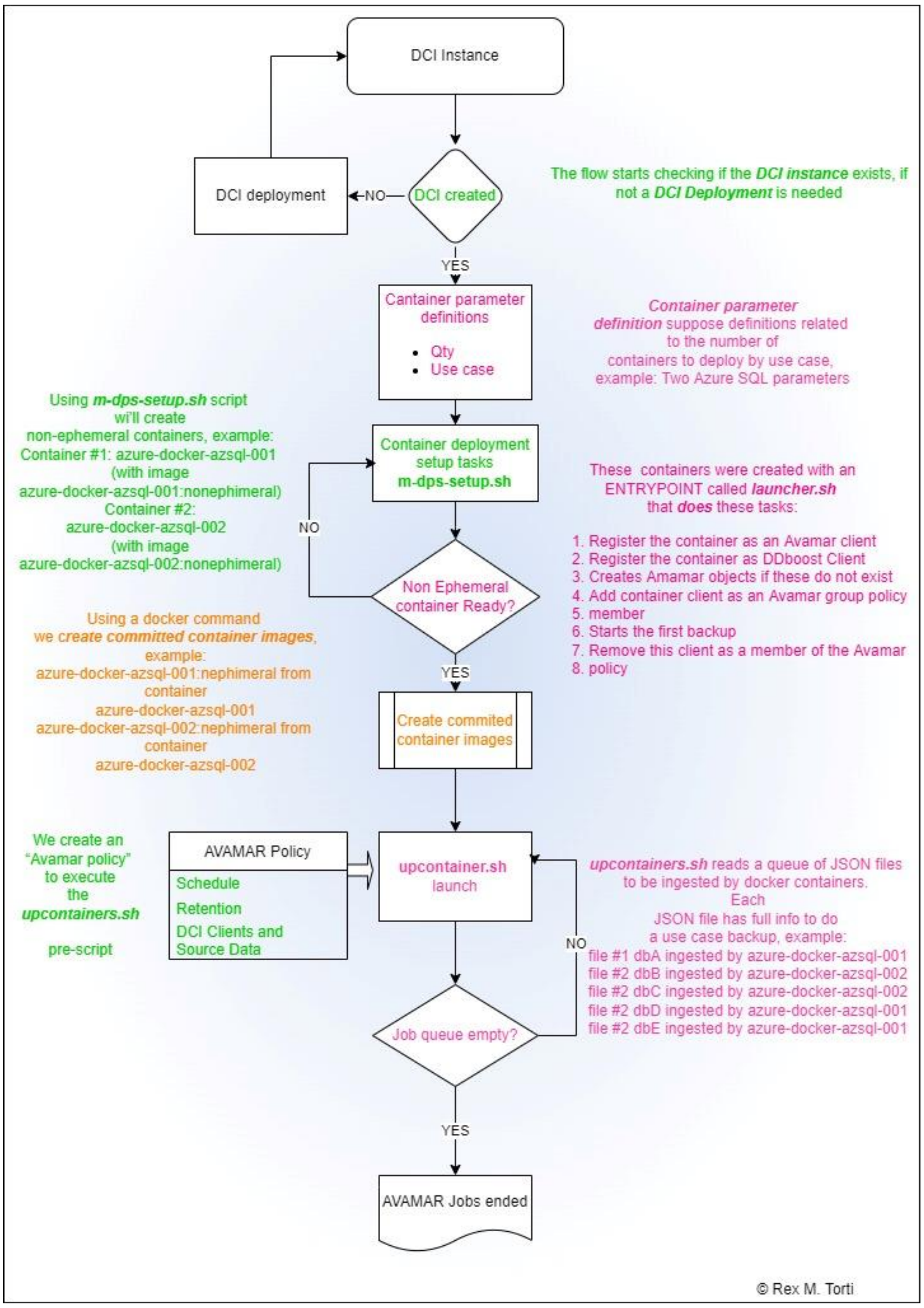
[1] Full code here: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/m-dps-setup.sh
[2] Full code here: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/main/nonephemeral.sh

Ephemeral

How ephemeral mode works: Ephemeral containers are an implementation technique that:

1. Defines the number of containers (slots) that will be started to save objects from the cloud.
2. Enable raised containers to ingest configuration files that indicate which resources will be backed up.
3. Removes the containers once the processing is finished.

The details are shown in the following.


DCI Instance
DCI deployment
NO
DCI created
YES
Cantainer parameter definitions
• Qty
• Use case
Container deployment setup tasks m-dps-setup.sh
NO
Non Ephemeral container Ready?
YES
Create commited container images
AVAMAR Policy
Schedule
Retention
DCI Clients and Source Data
upcontainer.sh launch
NO
Job queue empty?
YES
AVAMAR Jobs ended
© Rex M. Torti

The flow starts checking if the ***DCI instance*** exists, if not a ***DCI Deployment*** is needed

Container parameter definition suppose definitions related to the number of containers to deploy by use case, example: Two Azure SQL parameters

Using ***m-dps-setup.sh*** script wi'll create non-ephemeral containers, example:
Container #1: azure-docker-azsql-001 (with image azure-docker-azsql-001:nonephimeral)
Container #2: azure-docker-azsql-002 (with image azure-docker-azsql-002:nonephimeral)

These containers were created with an ENTRYPOINT called ***launcher.sh*** that ***does*** these tasks:

1. Register the container as an Avamar client
2. Register the container as DDboost Client
3. Creates Amamar objects if these do not exist
4. Add container client as an Avamar group policy
5. member
6. Starts the first backup
7. Remove this client as a member of the Avamar
8. policy

Using a docker command we ***create committed container images***, example:
azure-docker-azsql-001:nephimeral from container azure-docker-azsql-001
azure-docker-azsql-002:nephimeral from container azure-docker-azsql-002

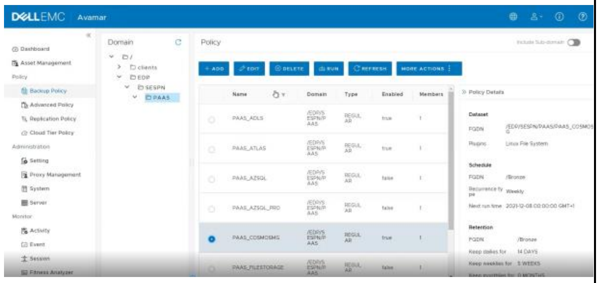
We create an "Avamar policy" to execute the ***upcontainers.sh*** pre-script

upcontainers.sh reads a queue of JSON files to be ingested by docker containers. Each JSON file has full info to do a use case backup, example:
file #1 dbA ingested by azure-docker-azsql-001
file #2 dbB ingested by azure-docker-azsql-002
file #2 dbC ingested by azure-docker-azsql-002
file #2 dbD ingested by azure-docker-azsql-001
file #2 dbE ingested by azure-docker-azsql-001

Backup and restore procedures

Avamar policy configuration

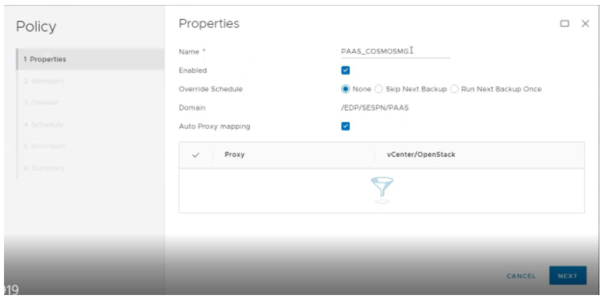
1) Locate the Avamar domain where you are going to configure the policy.



Name	Domain	Type	Enabled	Members
PAAS_ADLS	/EDP/SESPN/PAAS	REGULAR	true	1
PAAS_ATLAS	/EDP/SESPN/PAAS	REGULAR	true	1
PAAS_AZSQL	/EDP/SESPN/PAAS	REGULAR	false	1
PAAS_AZSQL_PRO	/EDP/SESPN/PAAS	REGULAR	false	1
PAAS_COSMOSMG	/EDP/SESPN/PAAS	REGULAR	true	1
PAAS_FILESTORAGE	/EDP/SESPN/PAAS	REGULAR	false	1

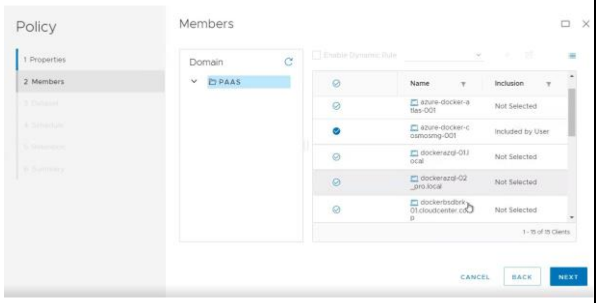
Dataset
FQDN /EDP/SESPN/PAAS/PAAS_COSMOSMG
Plugins Linux File System
Schedule
FQDN /Bronze
Recurrence type Weekly
Next run time 2021-12-08 00:00:00 GMT+1
Retention
FQDN /Bronze
Keep dailies for 14 DAYS
Keep weeklies for 5 WEEKS
Keep monthlies for 0 MONTHS

2) You must configure an Avamar policy. We'll use PAAS_COSMOSMG as an example.



Name * PAAS_COSMOSMG
Enabled ✓
Override Schedule [● None] ○ Skip Next Backup ○ Run Next Backup Once
Domain /EDP/SESPN/PAAS
Auto Proxy mapping ✓

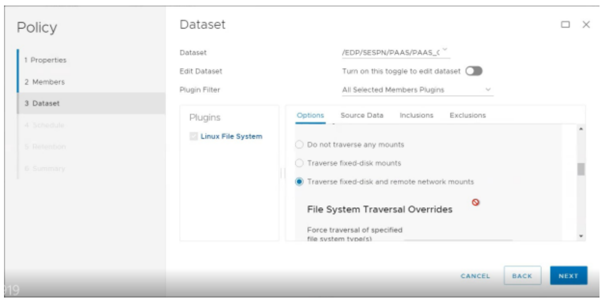
3) Select the members that you want to include in the policy The members are Docker containers previously registered with Avamar.



	Name	Inclusion
	azure-docker-atlas-001	Not Selected
✓	azure-docker-cosmosmg-001	Included by User
	dockerazql-01.local	Not Selected
	dockerazql-02_pro.local	Not Selected
	dockerbsdbrk01.cloudcenter.corp	Not Selected

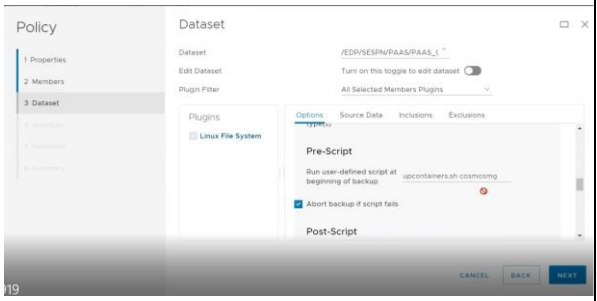
1 - 15 of 15 Clients

4) The Dataset must have the Traverse fixed-disk and remote network mounts option enabled for all file-based use cases.



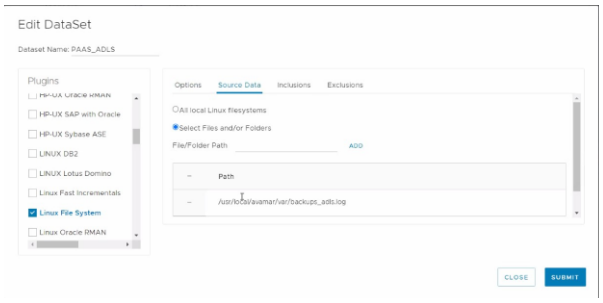
Dataset /EDP/SESPN/PAAS/PAAS_C
Edit Dataset Turn on this toggle to edit dataset
Plugin Filter All Selected Members Plugins
○ Do not traverse any mounts
○ Traverse fixed-disk mounts
● Traverse fixed-disk and remote network mounts
File System Traversal Overrides
Force traversal of specified file system type(s)

5) The Pre-Script must be upcontainers.sh[3] for the case of ephemeral containers implementation; otherwise you must choose the backup script that corresponds to your use case[4].



Dataset /EDP/SESPN/PAAS/PAAS_C
Edit Dataset Turn on this toggle to edit dataset
Plugin Filter All Selected Members Plugins
Pre-Script
Run user-defined script at beginning of backup upcontainers.sh cosmosmg
✓ Abort backup if script fails
Post-Script

6) The path will be /usr/local/avamar/var/backups_(dockerType) for containers implemented in ephemeral mode; otherwise it will be (RootBackupDir)/(dockerType)/backup.



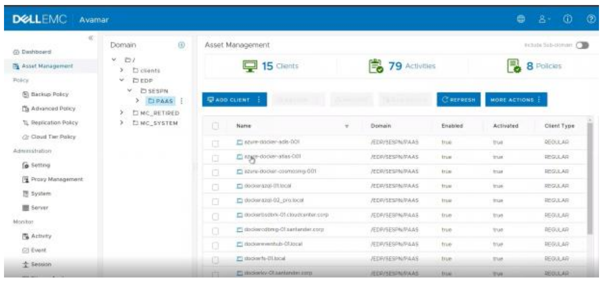
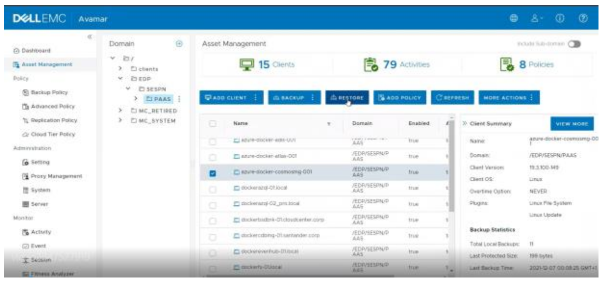
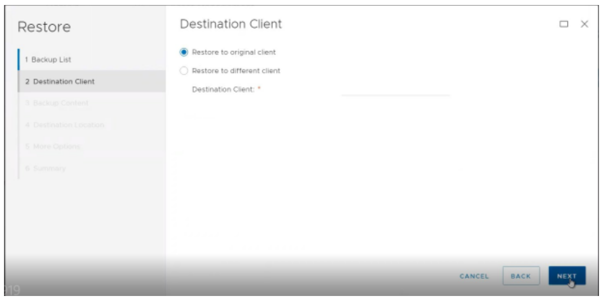
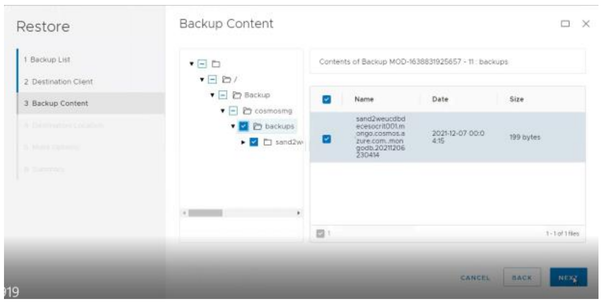
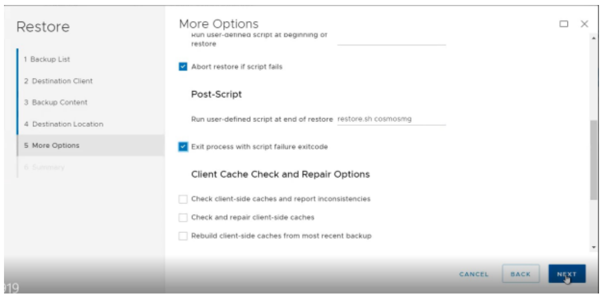
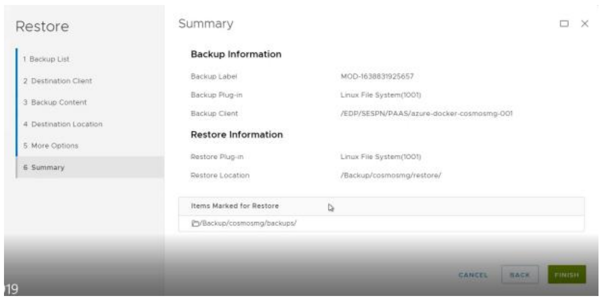
Dataset Name: PAAS_ADLS
○ All local Linux filesystems
● Select Files and/or Folders
File/Folder Path
Path
/usr/local/avamar/var/backups_adls.log

Don't forget to add container name DNS record (forward and reverse) to DNS Server.

[3] Upcontainers.sh full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/main/upcontainers.sh

[4] From here https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/main/upcontainers.sh or here https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/tree/main/src/avamar/azure

Restore procedure

1) Choose the asset you want to recover; the example is based on azure-docker-cosmosmg-001.	2) Press the RESTORE button.
	
3) Choose the option you want between Restore to original client or Restore to different client, if you choose the latter you will have to add the Destination Client.	4) Select the backup record that you consider appropriate.
 Restore 1 Backup List 2 Destination Client Destination Client Restore to original client Restore to different client Destination Client: *	 Restore 1 Backup List 2 Destination Client 3 Backup Content Backup Content Contents of Backup MOD-1638831925657 - 11 : backups Name: sand2weucdbdecesocrit001.mongo.cosmos.azure.com..mongodb.20211206230414 — Date: 2021-12-07 00:04:15 — Size: 199 bytes 1 - 1 of 1 files
5) The destination location will be (RootBackupDir)/(dockerType)/restore. If it is a restore using useBlobFuse or uses3fs (1), put in Pre-Script the script preres-(dockerType).sh that corresponds to your use case.	6) Summary before starting the restore.
 Restore 1 Backup List 2 Destination Client 3 Backup Content 4 Destination Location 5 More Options 6 Summary More Options Run user-defined script at beginning of restore [✓] Abort restore if script fails Post-Script Run user-defined script at end of restore: restore.sh cosmosmg [✓] Exit process with script failure exitcode Client Cache Check and Repair Options [ ] Check client-side caches and report inconsistencies [ ] Check and repair client-side caches [ ] Rebuild client-side caches from most recent backup	 Restore 1 Backup List 2 Destination Client 3 Backup Content 4 Destination Location 5 More Options 6 Summary Summary **Backup Information** Backup Label: MOD-1638831925657 Backup Plug-in: Linux File System(1001) Backup Client: /EDP/SESPN/PAAS/azure-docker-cosmosmg-001 **Restore Information** Restore Plug-in: Linux File System(1001) Restore Location: /Backup/cosmosmg/restore/ Items Marked for Restore /Backup/cosmosmg/backups/

For the database case you can use the universal database restore script called dbrestore.sh[5] that extracts and copies objects stored in the dump or export file to a new or existing instance.

```
-t <db type>
 -u <username>
 -p <password>
```

[5] dbrestore.sh script full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/main/src/avamar/azure/CLI/dbrestore.sh

```
    -s <server hostname or IP address>
    -d <db name>
    -a <table name> optional
    -n only used to indicate wwhetherthe restore is for a new database or ean xisting
      one;    can only be YES or NO
    -r <restore path to backup folder>
```

Examples:

```
    1- cosmosmg: ./dbrestore.sh -t cosmosmg -u <username> -p <password> -s <servername>
-d <db name> -r <path to db backup folder>
    2- azsql: ./dbrestore.sh -t azsql -u <username> -p <password> -s <servername> -d
<db name> -r <path to db backup folder> -o <port number>
    3- PostgreSQL: ./dbrestore.sh -t postgresql -u <username> -p <password> -s
<servername> -n yes -d <db name> -r <path to db backup folder>
    4- Atlas: ./dbrestore.sh -t atlas -u <username> -p <password> -s <servername> -d
<db name> -t <table name> -r <path to db backup folder>
```

Image Builder

Description

This is an internal utility used to generate custom Docker images from CentOS. We are using Trivy to check vulnerabilities (https://aquasecurity.github.io/trivy/v0.19.2/getting-started/installation/)

Usage

```
$ ./imagebuilder.sh --help
Please type -b | --build  <image> <tag> <cloud> to build a new docker image
-s | --save to <image> <tag> save a tar format image
-t | --test to <image> <tag> create and start a test container
-r | --removetest <image> to remove a test container
-sc | --scan <image> to scan with trivy

./build.sh -b <image> <tag>
./build.sh -b netappstorage latest

[userid@ dci CustomImages]$ sudo docker images | grep netappstorage

localhost/centosaveddaznetappstorage      latest    d3c11659ce91    30 minutes ago    2.14 GB



./build.sh -s <dockertype> <imageversion>
./build.sh -s netappstorage latest

[userid@ dci CustomImages]$ ls -lrt ../imagesTarFormat/*netappstorage*

 -rw-r--r--. 1 root root 2142516224 Jul  5 15:40
../imagesTarFormat/centosaveddaznetappstorage.latest.tar
```

Where to place packages and dockerfiles

```
CustomImages/<dockertype>/Azure-<dockertype>-CustomImage.dockerfile
   --> Used by imagebuilder.sh. See "dockertypes" values
CustomImages/imagebuilder.sh --> Image builder script
packages/DockerEmbebed/avamar/19.3/avamar --> Avamar client and MCCLI packages
packages/DockerEmbebed/azcli --> Azure command line interface
packages/DockerEmbebed/blobstorage --> To mount blobs as file system when necessary
packages/DockerEmbebed/ddboostfs --> DDboostFS client
packages/DockerEmbebed/avamar/19.3/<dockertype>
      --> Image specific package, Examples: NetApp, Azure SQL, and so on
imagesTarFormat --> Output folder with .tar format docker images
```

Prerequisites

Requires internet access.

Conclusion

This development is a universal backup solution for the cloud, capable of adapting to different providers such as Azure Cloud and AWS Cloud.

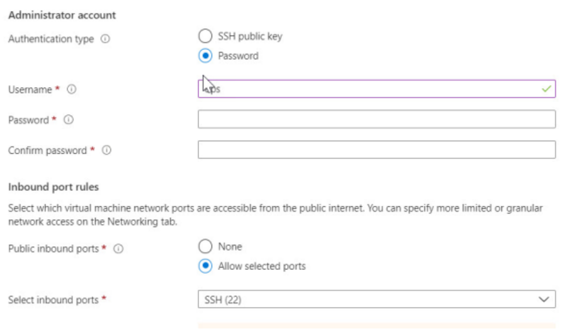
The code understands all the technologies deployed, whether blob storage or structured and unstructured databases. It allows you to include native backup commands either through the command line interface or Rest API calls.

It can integrate with world-class backup solutions and is surely the lowest cost solution that has been achieved, both because of the few resources needed to deploy it and the open software integration capacities.

Appendix A: DCI and CVM deployments

DCI and CVM Azure: Deploy a Red Hat 7.9 or higher ***Standard_B2s*** VM with these properties.

Authentication type



Administrator account

Authentication type: ○ SSH public key [✓ Password]

Username *: ps

Password *:

Confirm password *:

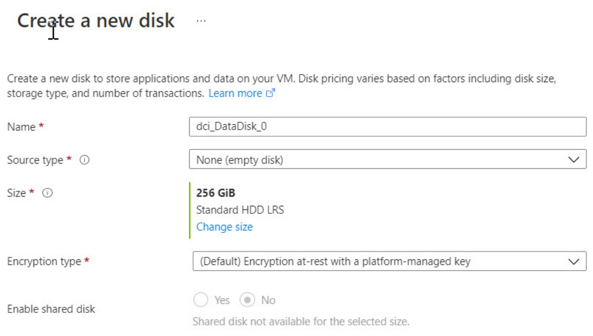
Inbound port rules

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports *: ○ None [✓ Allow selected ports]

Select inbound ports *: SSH (22)

Disk config



Create a new disk ···

Create a new disk to store applications and data on your VM. Disk pricing varies based on factors including disk size, storage type, and number of transactions. Learn more

Name *: dci_DataDisk_0

Source type *: None (empty disk)

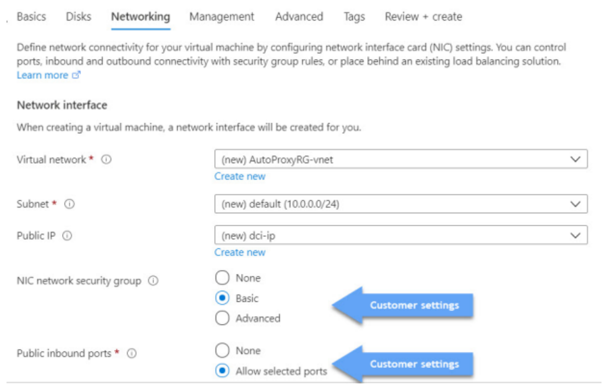
Size *: **256 GiB** Standard HDD LRS Change size

Encryption type *: (Default) Encryption at-rest with a platform-managed key

Enable shared disk: ○ Yes [✓ No]

Shared disk not available for the selected size.

Network config



Basics | Disks | **Networking** | Management | Advanced | Tags | Review + create

Define network connectivity for your virtual machine by configuring network interface card (NIC) settings. You can control ports, inbound and outbound connectivity with security group rules, or place behind an existing load balancing solution. Learn more

Network interface

When creating a virtual machine, a network interface will be created for you.

Virtual network *: (new) AutoProxyRG-vnet Create new

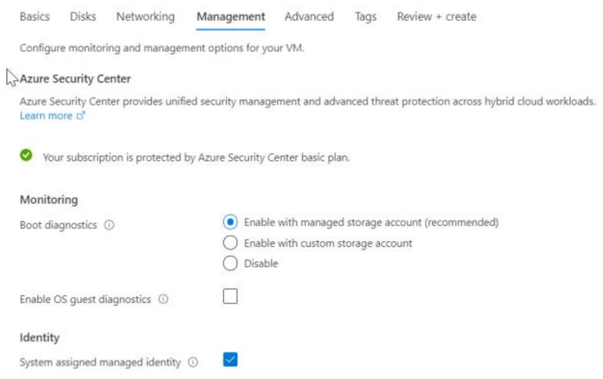
Subnet *: (new) default (10.0.0.0/24)

Public IP: (new) dci-ip Create new

NIC network security group: ○ None [✓ Basic] ○ Advanced

Public inbound ports *: ○ None [✓ Allow selected ports]

Identity config



Basics | Disks | Networking | **Management** | Advanced | Tags | Review + create

Configure monitoring and management options for your VM.

Azure Security Center

Azure Security Center provides unified security management and advanced threat protection across hybrid cloud workloads. Learn more

✓ Your subscription is protected by Azure Security Center basic plan.

Monitoring

Boot diagnostics: [✓ Enable with managed storage account (recommended)] ○ Enable with custom storage account ○ Disable

Enable OS guest diagnostics: ☐

Identity

System assigned managed identity: ✓

DCI and CVM AWS: Please see this guide https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/docs/aws_dci_config.md for details.

DCI tasks:

Install this repo after the provisioning of the Deploy Control Instance from Azure, AWS or Google Cloud:

1. Install git on Deploy Control Instance
   `sudo yum install -y git`
2. Install code repo (choose one of them)
   a. Use git to clone the repo
      `git clone https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup.git`
   b. Download, copy into DCI and unzip.
      `https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/archive/refs/heads/main.zip`

CVM task:

Run `sudo yum install -y docker` to install Docker. Each Docker consumes ~100 MB RAM memory; this is a very lightweight solution.

Run dps-setup.sh -s [6] to set up the DCI environment. The parameter -s requires internet access. Only must be run one time by DCI or CVM.

[6] Full code here: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/dps-setup.sh

Appendix B: Repo structure

Repo structure: This repository contains the following files and folders.

```
{home} : Setup script, ReadME and .gitignore files.
CustomImages/packages/DockerEmbebed: Base packages to build preinstalled docker
 image
CustomImages/(use case): Dockerfile to create use case docker image
 No directory to: adls blobstorage cosmosql cvision databricks eventhub hdinsigth
 keyvault redis
docs : Documents
imagesTarFormat: Temporary folder user by imagebuilder.sh -s | --save option
jsonfiles: File to process by upcontainers.sh script
jsonfilesTemplates: One for each use case
sources: Secondary scripts
src/avamar/(aws|azure|ibm)/(CLI|API) : Backup scripts
src/avamar/(aws|azure|ibm)/(CLI|API)/(PowerShell|bash): Several automation scripts
src/azure/  : Pem file to avoid expouse password
src/dockerfiles/current/ : Docker file work space
src/packages/DockerEmbebed/Certificates : Contains Certificates
```

This repo uses the following technologies.

```
- Docker(or Podman ) config files called "dockerfiles"
- JSON files  with .json extension
- Shell scripts with .sh extension
- ReadMe files with .md extension
- Client packages with .rpm extension
- Certificates to connect to the cloud provider (PEM files with extension .pem )
```

Appendix C: Use case info-related

This appendix presents information related to each use case. The third and fourth tables inform about the commands and access roles used in each case to build the backup scripts.

Terms

dockerType: Long name to identify properties related to a particular technology.

dockerTypeName: Sort names to identify properties related to a particular technology.

useDDBoost: If DDboost protocol is used to store dump and export in Data Domain before moving to Avamar.

useBlobFuse: If the blobfuse utility is used to mount blob storage as a linux file system to backup files.

useCommand: If the backup/restore script is based on commands or ad-hoc utilities.

useAPI: If the backup/restore script is based on Rest API calls.

backupContent: If backup records contain user data, user config, or both.

Azure use case-specific

#	1	2	3	4	5	
dockerType	adls	azsql	blobstorage	cosmosql	cvision	
dockerTypeName	DL	azsql	BS	CSQL	CV	
useDDBoost	no	yes	no	yes	yes	
useBlobFuse	yes	no	yes	no	no	
useCommand	yes	yes	yes	yes	yes	
useAPI	no	no	no	no	yes	
packages	blobfuse	azsql	blobfuse	no	no	
backupContent (data/config)	data	data	data	data	data	
#	6	7	8	9	10	
dockerType	databricks	datafactory	eventhub	filestorage	hdinsigth	
dockerTypeName	DB	DF	EH	FS	HD	
useDDBoost	no	yes	no	no	no	
useBlobFuse	yes	no	yes	no	yes	
useCommand	yes	yes	yes	yes	yes	
useAPI	no	no	no	no	no	
packages	blobfuse	git	no	cifs-utils	blobfuse	
backupContent (data/config)	data	data	data	data	data	
#	11	12	13	14	15	16
dockerType	keyvault	cosmosmg	atlas	netappstorage	postgresql	redis

dockerTypeName	KV	CMG	AMG	NS	PG	RD
useDDBoost	yes	yes	yes	no	yes	yes
useBlobFuse	no	no	no	no	no	yes
useCommand	yes	yes	yes	yes	yes	yes
useAPI	no	no	yes	no	no	no
packages	no	mongodb-tools	no	nfs-utils	postgresql	blobfuse
backupContent (data/config)	data	data	data	data	data	data

AWS use case-specific

#	1	2	3
dockerType	S3	elasticsearch	redis
dockerTypeName	S3	ES	AWSRD
useDDBoost	no	no	no
useBlobFuse	yes	yes	yes
useCommand	yes	yes	yes
useAPI	no	no	no
packages			
backupContent (data/config)	data	data	data

Please see the access requirement by use case. This box does not apply if managed identity is used:

	Command	Role/identity/permission required for backup
Azure env	az login --service-principal	Rol
	az account set --subscription	N/A
	az resource list	Reader
	nslookup	N/A
	az logout	N/A
	az ad sp create-for-rbac	Owner
adls		
azsql	az sql db list	SQL DB Contributor/SQL db_datareader
		SQL loginmanager/VIEW DEFINITION grant
	az keyvault secret show / curl	Key Vault Secrets User
	sqlpackage	Set up AD Admin on each of the SQL Servers
blobstorage	az storage account list	Storage Blob Data Reader

	az storage account show	Storage Blob Data Reader
	az storage account keys list	Reader and Data Access
	az storage container list	Storage Blob Data Reader
	blobfuse	Storage Blob Data Owner
cosmosql	az datafactory pipeline list	Data Factory Contributor
	az datafactory pipeline create-run	Data Factory Contributor
	az datafactory pipeline-run show	Data Factory Contributor
cvision	az cognitiveservices account keys list	Cognitive Services Custom Vision Reader
	az resource show	Reader
	curl	Cognitive Services Custom Vision Contributor
databricks	az databricks workspace show	Reader
	az storage container list	Storage Blob Data Reader
	blobfuse	Storage Blob Data Owner
datafactory	az datafactory show	Data Factory Contributor
	az resource list	Contributor
	git clone	Data Factory Contributor
	tar	N/A
eventhub	az eventhubs eventhub list	Rol: Storage Account Key Operator Service Role
	az eventhubs eventhub show	Reader
	blobfuse	Storage Blob Data Owner
filestorage	az storage account show	Storage Blob Data Reader
	az storage account keys list	Reader and Data Access
	az storage share list	Storage File Data SMB Share Reader
	mount -t cifs	Storage Blob Data Owner
hdinsigth	az hdinsight show	HDInsight Cluster Operator
	az storage account keys list	Reader and Data Access
	blobfuse	Storage Blob Data Owner
keyvault	az keyvault list	Key Vault Crypto User
	az keyvault secret list	Key Vault Secrets User
	az keyvault secret backup	Key Vault Crypto User
	az keyvault certificate backup	Key Vault Crypto User
		GET, LIST and BACKUP
		GET, LIST and RESTORE
mongodb	az cosmosdb list	Reader
	curl	Key Vault Secrets User

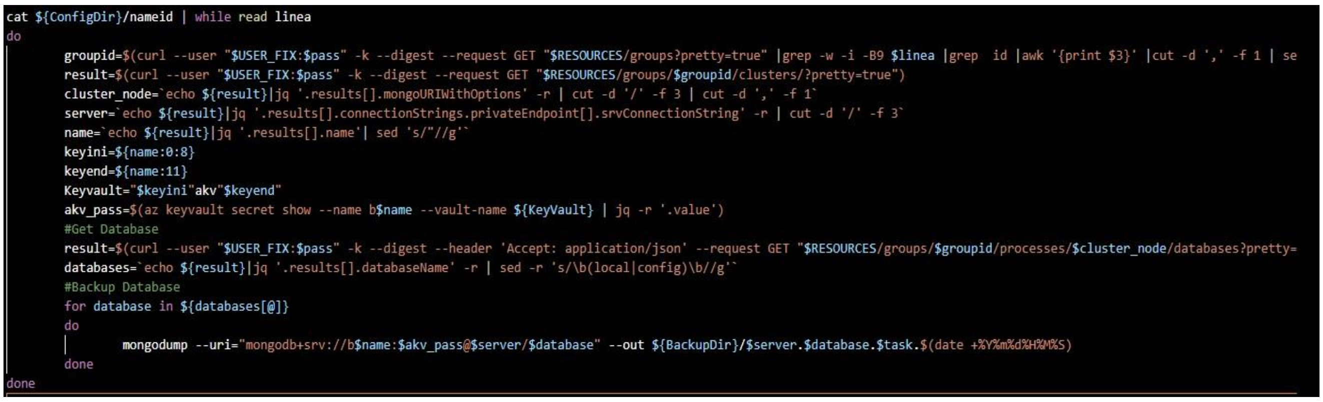
	az cosmosdb mongodb database list	Reader
	mongodump	Read-only Key
	az cosmosdb list	Reader
	mongodump	Reader
	mongorestore	Reader
	az lock create	Microsoft.Authorization/locks/*'
	az lock delete	Microsoft.Authorization/locks/*'
mongodb -atlas	az keyvault secret show / curl	Key Vault Secrets User
	mongodump	adminBackup (cloud.mongo.com)
netappstorage	mount -t nfs	Storage Blob Data Owner
postgresql	az postgres db list	Reader
	az network private-endpoint list	Reader
	az keyvault secret show / curl	Key Vault Secrets User
	pg_dump / pg_dumpall	connect on database/select on all tables in schema
redis	az storage account list	Storage Blob Data Reader
	az storage account create	Storage Blob Contributor
	az storage container list	Storage Blob Data Reader
	az storage container create	Storage Blob Contributor
	az role assignment list	Storage Blob Data Contributor
	az role assignment create	Storage Blob Data Contributor
	az storage container generate-sas	Storage Blob Data Contributor
	az redis export	Redis Cache Contributor
	blobfuse	Storage Blob Data Owner

	Command	Role/identity/permission required for backup
AWS env	AWS login	Rol
	Subscription setting	N/A
	Resource list and tag query	Reader
	Reverse lookup	N/A
	Azure logout	N/A
	Create Service principal account	Owner
redis	aws configure set default.region	(all command tested using identity-based)
	aws configure set aws_access_key_id	
	aws configure set aws_secret_access_key	

	aws elasticache describe-cache-clusters	
	s3fs	
elasticsearch	aws configure set default.region	(all command tested using identity-based)
	aws es list-domain-names	
	aws es describe-elasticsearch-domain	
	s3fs	
	aws elasticache create-snapshot	
	aws elasticache describe-cache-clusters	
s3	aws configure set default.region	(all command tested using identity-based)
	s3fs	

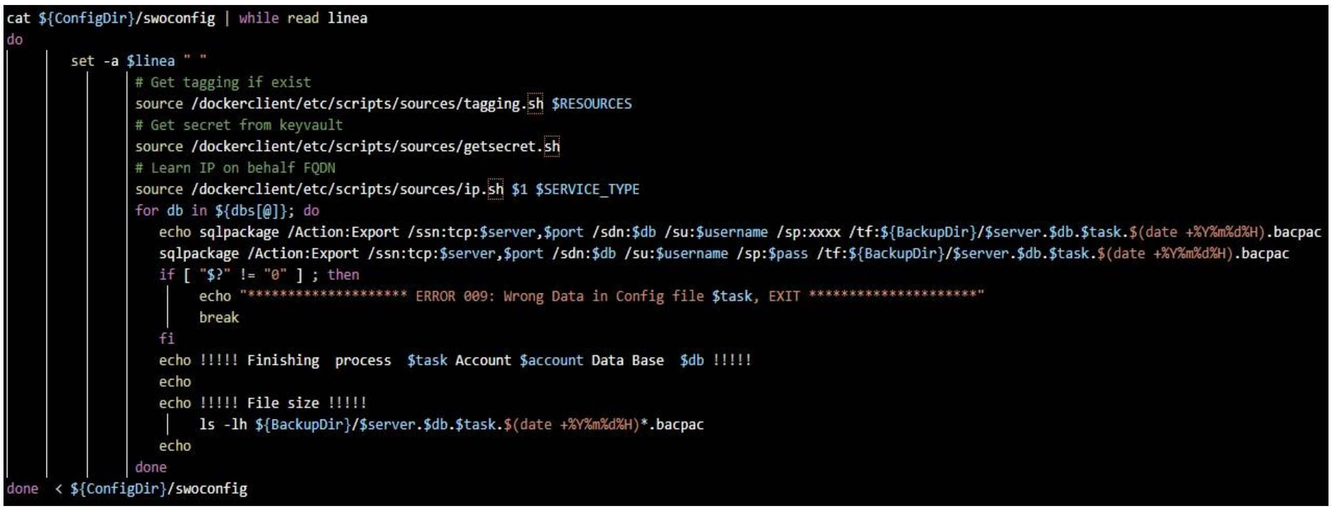
Appendix D: Business logic by use case

Atlas[7]: Curl statements locate the database to back up with the mongodump command



```
cat ${ConfigDir}/nameid | while read linea
do
        groupid=$(curl --user "$USER_FIX:$pass" -k --digest --request GET "$RESOURCES/groups?pretty=true" |grep -w -i -B9 $linea |grep  id |awk '{print $3}' |cut -d ',' -f 1 | se
        result=$(curl --user "$USER_FIX:$pass" -k --digest --request GET "$RESOURCES/groups/$groupid/clusters/?pretty=true")
        cluster_node=`echo ${result}|jq '.results[].mongoURIWithOptions' -r | cut -d '/' -f 3 | cut -d ',' -f 1`
        server=`echo ${result}|jq '.results[].connectionStrings.privateEndpoint[].srvConnectionString' -r | cut -d '/' -f 3`
        name=`echo ${result}|jq '.results[].name'| sed 's/"//g'`
        keyini=${name:0:8}
        keyend=${name:11}
        Keyvault="$keyini"akv"$keyend"
        akv_pass=$(az keyvault secret show --name b$name --vault-name ${KeyVault} | jq -r '.value')
        #Get Database
        result=$(curl --user "$USER_FIX:$pass" -k --digest --header 'Accept: application/json' --request GET "$RESOURCES/groups/$groupid/processes/$cluster_node/databases?pretty=
        databases=`echo ${result}|jq '.results[].databaseName' -r | sed -r 's/\b(local|config)\b//g'`
        #Backup Database
        for database in ${databases[@]}
        do
                mongodump --uri="mongodb+srv://b$name:$akv_pass@$server/$database" --out ${BackupDir}/$server.$database.$task.$(date +%Y%m%d%H%M%S)
        done
done
```

AZ SQL[8]: We use the sqlpackage command to export the previously located database with AZ CLI commands (not shown in the figure).



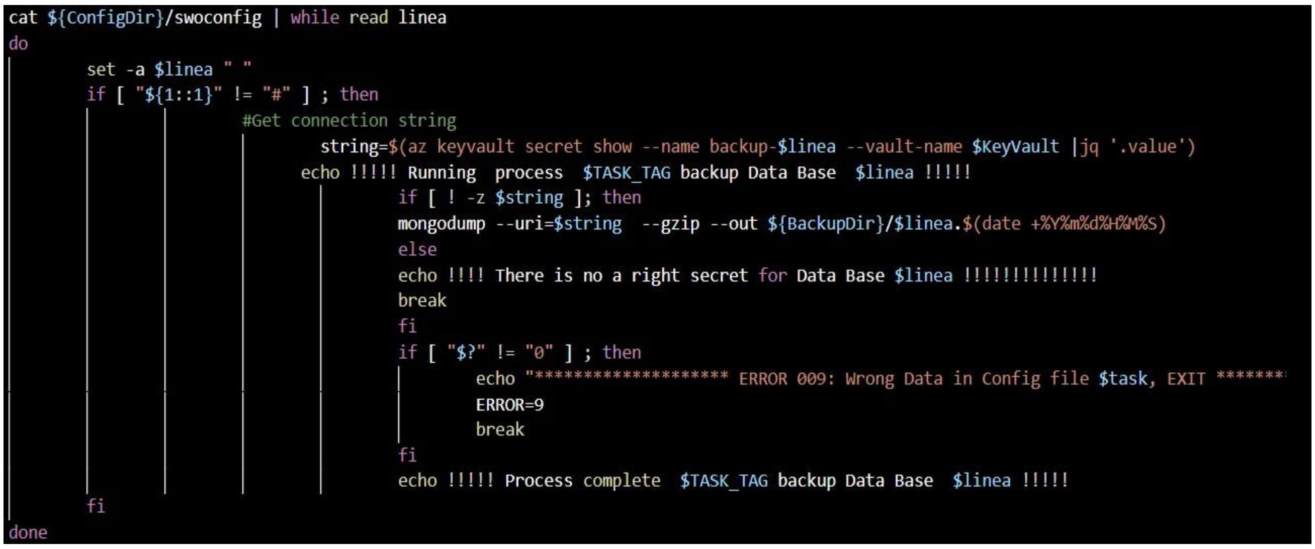
```
cat ${ConfigDir}/swoconfig | while read linea
do
        set -a $linea " "
                # Get tagging if exist
                source /dockerclient/etc/scripts/sources/tagging.sh $RESOURCES
                # Get secret from keyvault
                source /dockerclient/etc/scripts/sources/getsecret.sh
                # Learn IP on behalf FQDN
                source /dockerclient/etc/scripts/sources/ip.sh $1 $SERVICE_TYPE
                for db in ${dbs[@]}; do
                   echo sqlpackage /Action:Export /ssn:tcp:$server,$port /sdn:$db /su:$username /sp:xxxx /tf:${BackupDir}/$server.$db.$task.$(date +%Y%m%d%H).bacpac
                   sqlpackage /Action:Export /ssn:tcp:$server,$port /sdn:$db /su:$username /sp:$pass /tf:${BackupDir}/$server.$db.$task.$(date +%Y%m%d%H).bacpac
                   if [ "$?" != "0" ] ; then
                        echo "******************* ERROR 009: Wrong Data in Config file $task, EXIT *********************"
                        break
                   fi
                   echo !!!!! Finishing  process  $task Account $account Data Base  $db !!!!!
                   echo
                   echo !!!!! File size !!!!!
                        ls -lh ${BackupDir}/$server.$db.$task.$(date +%Y%m%d%H)*.bacpac
                   echo
                done
done  < ${ConfigDir}/swoconfig
```

CosmoMG[9]: We use the mongodump command to dump the previously located database with AZ CLI commands (not shown in the figure).

[7] Atlas full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/backup-atlas.sh

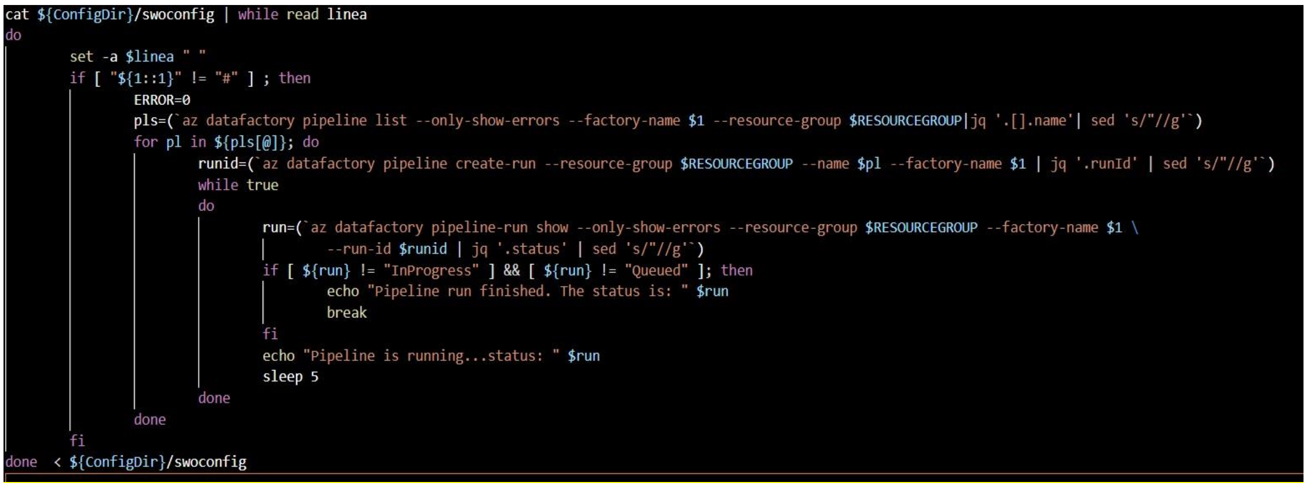
[8] Azure SQL full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/backup-azsql.sh

[9] Cosmo Mongo code here:https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/backup-azsql.sh



```
cat ${ConfigDir}/swoconfig | while read linea
do
        set -a $linea " "
        if [ "${1::1}" != "#" ] ; then
                        #Get connection string
                                string=$(az keyvault secret show --name backup-$linea --vault-name $KeyVault |jq '.value')
                            echo !!!!! Running  process  $TASK_TAG backup Data Base  $linea !!!!!
                                        if [ ! -z $string ]; then
                                        mongodump --uri=$string  --gzip --out ${BackupDir}/$linea.$(date +%Y%m%d%H%M%S)
                                        else
                                        echo !!!! There is no a right secret for Data Base $linea !!!!!!!!!!!!!!
                                        break
                                        fi
                                        if [ "$?" != "0" ] ; then
                                                echo "******************** ERROR 009: Wrong Data in Config file $task, EXIT *******
                                                ERROR=9
                                                break
                                        fi
                                        echo !!!!! Process complete  $TASK_TAG backup Data Base  $linea !!!!!
        fi
done
```

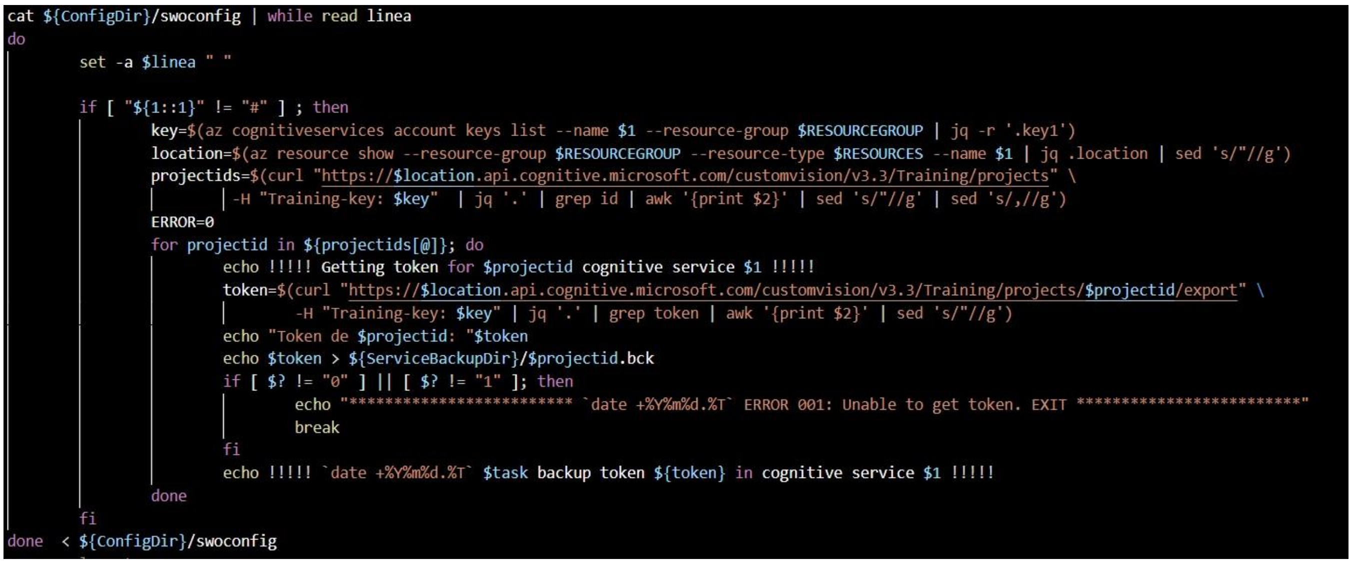
CosmoSQL[10]: We use datafactory as a bridge to protect a documentary database.



```
cat ${ConfigDir}/swoconfig | while read linea
do
        set -a $linea " "
        if [ "${1::1}" != "#" ] ; then
                ERROR=0
                pls=(`az datafactory pipeline list --only-show-errors --factory-name $1 --resource-group $RESOURCEGROUP|jq '.[].name'| sed 's/"//g'`)
                for pl in ${pls[@]}; do
                        runid=(`az datafactory pipeline create-run --resource-group $RESOURCEGROUP --name $pl --factory-name $1 | jq '.runId' | sed 's/"//g'`)
                        while true
                        do
                                run=(`az datafactory pipeline-run show --only-show-errors --resource-group $RESOURCEGROUP --factory-name $1 \
                                        --run-id $runid | jq '.status' | sed 's/"//g'`)
                                if [ ${run} != "InProgress" ] && [ ${run} != "Queued" ]; then
                                        echo "Pipeline run finished. The status is: " $run
                                        break
                                fi
                                echo "Pipeline is running...status: " $run
                                sleep 5
                        done
                done
        fi
done  < ${ConfigDir}/swoconfig
```

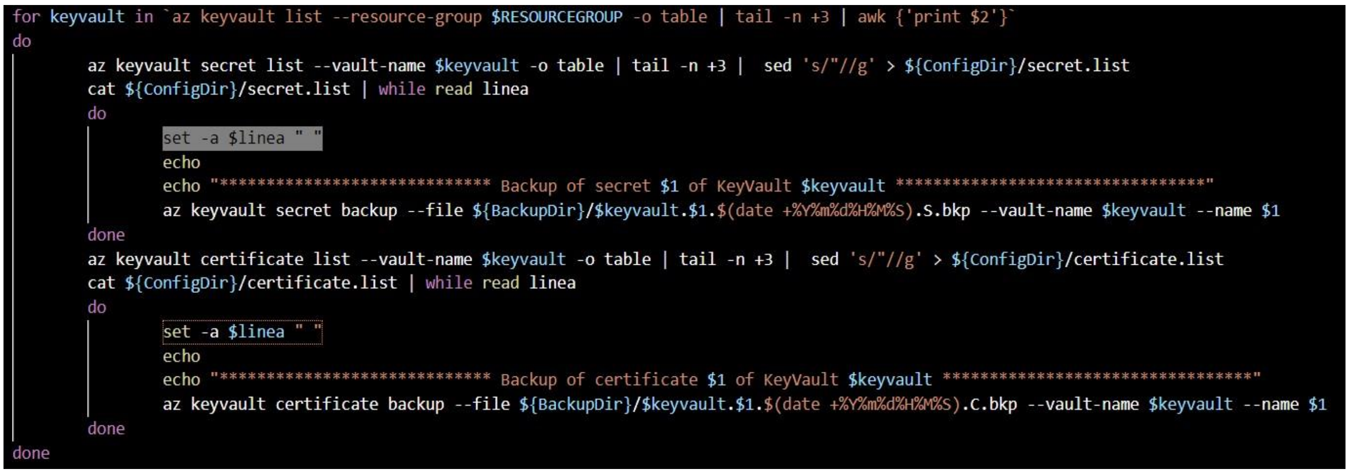
[10] Cosmo SQL full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/backup-cosmosql.sh

cvision[11]: We export the project with curl statements; the project is located via AZ CLI.



```
cat ${ConfigDir}/swoconfig | while read linea
do
        set -a $linea " "

        if [ "${1::1}" != "#" ] ; then
                key=$(az cognitiveservices account keys list --name $1 --resource-group $RESOURCEGROUP | jq -r '.key1')
                location=$(az resource show --resource-group $RESOURCEGROUP --resource-type $RESOURCES --name $1 | jq .location | sed 's/"//g')
                projectids=$(curl "https://$location.api.cognitive.microsoft.com/customvision/v3.3/Training/projects" \
                        -H "Training-key: $key"  | jq '.' | grep id | awk '{print $2}' | sed 's/"//g' | sed 's/,//g')
                ERROR=0
                for projectid in ${projectids[@]}; do
                        echo !!!!! Getting token for $projectid cognitive service $1 !!!!!
                        token=$(curl "https://$location.api.cognitive.microsoft.com/customvision/v3.3/Training/projects/$projectid/export" \
                                -H "Training-key: $key" | jq '.' | grep token | awk '{print $2}' | sed 's/"//g')
                        echo "Token de $projectid: "$token
                        echo $token > ${ServiceBackupDir}/$projectid.bck
                        if [ $? != "0" ] || [ $? != "1" ]; then
                                echo "************************ `date +%Y%m%d.%T` ERROR 001: Unable to get token. EXIT ************************"
                                break
                        fi
                        echo !!!!! `date +%Y%m%d.%T` $task backup token ${token} in cognitive service $1 !!!!!
                done
        fi
done  < ${ConfigDir}/swoconfig
```

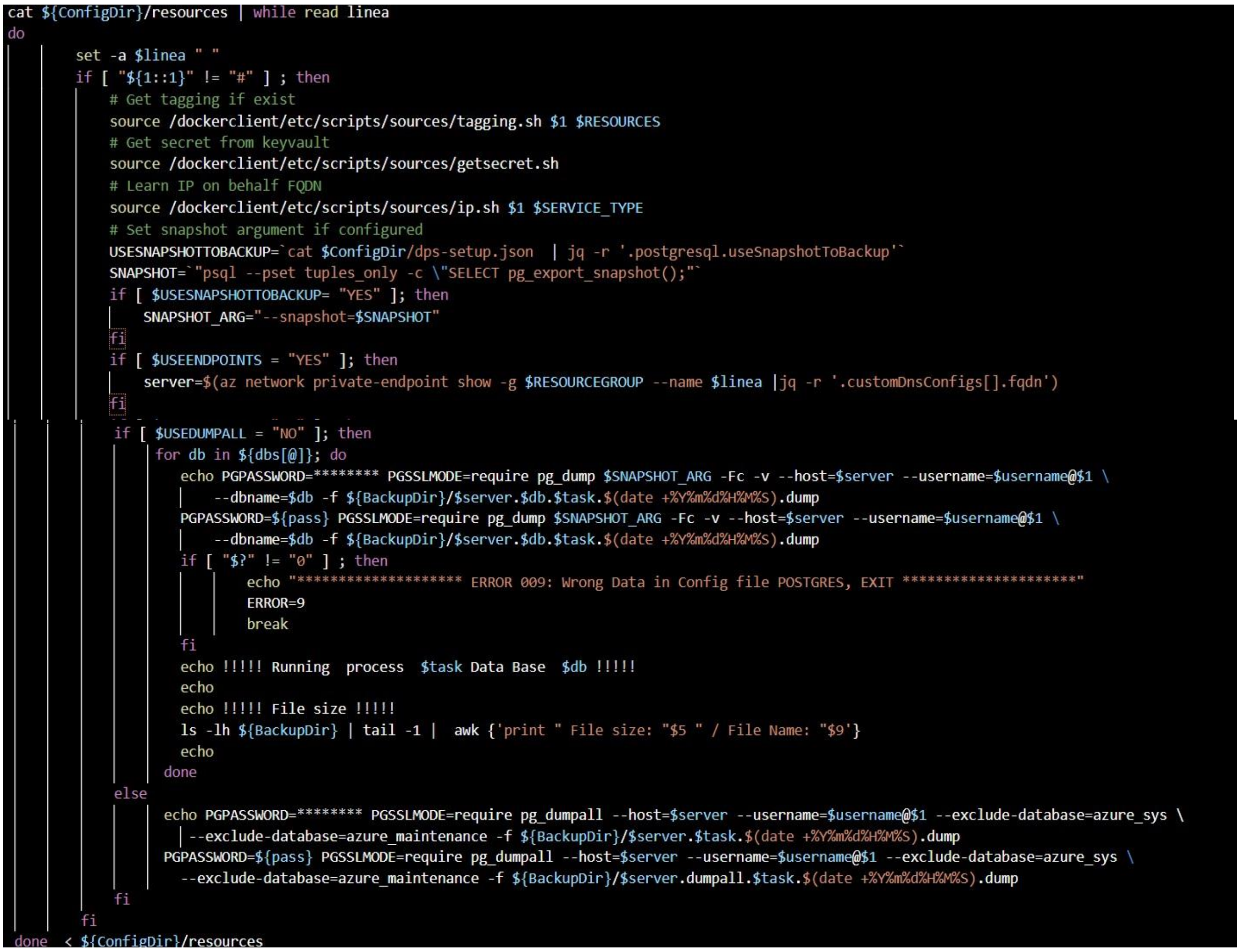
Keyvault[12]**:** We safeguard secrets and certificates using AZ CLI.



```
for keyvault in `az keyvault list --resource-group $RESOURCEGROUP -o table | tail -n +3 | awk {'print $2'}`
do
        az keyvault secret list --vault-name $keyvault -o table | tail -n +3 |  sed 's/"//g' > ${ConfigDir}/secret.list
        cat ${ConfigDir}/secret.list | while read linea
        do
                set -a $linea " "
                echo
                echo "***************************** Backup of secret $1 of KeyVault $keyvault *********************************"
                az keyvault secret backup --file ${BackupDir}/$keyvault.$1.$(date +%Y%m%d%H%M%S).S.bkp --vault-name $keyvault --name $1
        done
        az keyvault certificate list --vault-name $keyvault -o table | tail -n +3 |  sed 's/"//g' > ${ConfigDir}/certificate.list
        cat ${ConfigDir}/certificate.list | while read linea
        do
                set -a $linea " "
                echo
                echo "***************************** Backup of certificate $1 of KeyVault $keyvault *********************************"
                az keyvault certificate backup --file ${BackupDir}/$keyvault.$1.$(date +%Y%m%d%H%M%S).C.bkp --vault-name $keyvault --name $1
        done
done
```

[11] Cognitive service custom vision full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/backup-cvision.sh
[12] Key vault full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/backup-cvision.sh

PostgreSQL[13]: We use pg_dump or pg_dumpall to back up the previously discovered database(s) with AZ CLI.



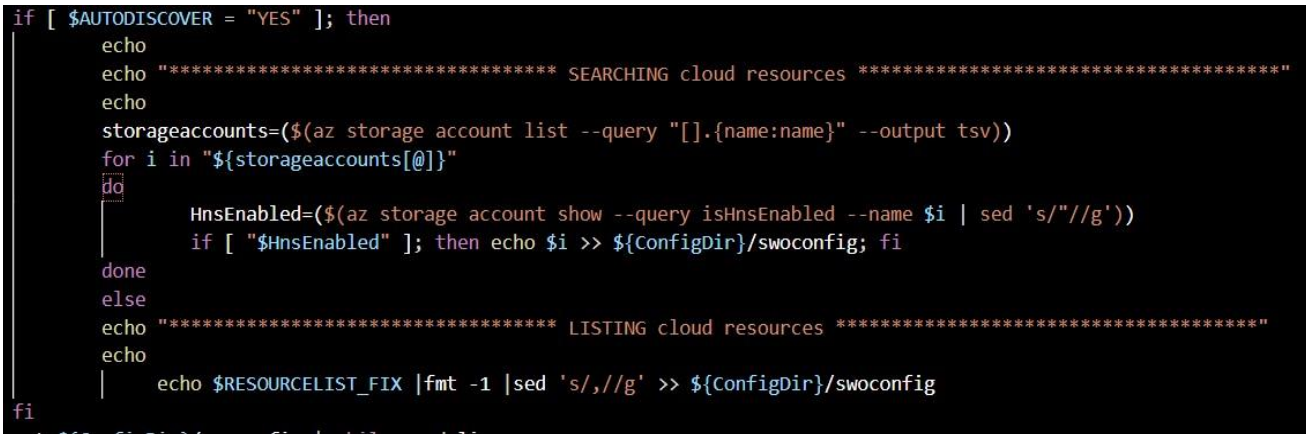
```
cat ${ConfigDir}/resources | while read linea
do
        set -a $linea " "
        if [ "${1::1}" != "#" ] ; then
            # Get tagging if exist
            source /dockerclient/etc/scripts/sources/tagging.sh $1 $RESOURCES
            # Get secret from keyvault
            source /dockerclient/etc/scripts/sources/getsecret.sh
            # Learn IP on behalf FQDN
            source /dockerclient/etc/scripts/sources/ip.sh $1 $SERVICE_TYPE
            # Set snapshot argument if configured
            USESNAPSHOTTOBACKUP=`cat $ConfigDir/dps-setup.json  | jq -r '.postgresql.useSnapshotToBackup'`
            SNAPSHOT=`"psql --pset tuples_only -c \"SELECT pg_export_snapshot();"`
            if [ $USESNAPSHOTTOBACKUP= "YES" ]; then
                SNAPSHOT_ARG="--snapshot=$SNAPSHOT"
            fi
            if [ $USEENDPOINTS = "YES" ]; then
                server=$(az network private-endpoint show -g $RESOURCEGROUP --name $linea |jq -r '.customDnsConfigs[].fqdn')
            fi
            if [ $USEDUMPALL = "NO" ]; then
                for db in ${dbs[@]}; do
                   echo PGPASSWORD=******** PGSSLMODE=require pg_dump $SNAPSHOT_ARG -Fc -v --host=$server --username=$username@$1 \
                       --dbname=$db -f ${BackupDir}/$server.$db.$task.$(date +%Y%m%d%H%M%S).dump
                   PGPASSWORD=${pass} PGSSLMODE=require pg_dump $SNAPSHOT_ARG -Fc -v --host=$server --username=$username@$1 \
                       --dbname=$db -f ${BackupDir}/$server.$db.$task.$(date +%Y%m%d%H%M%S).dump
                   if [ "$?" != "0" ] ; then
                           echo "******************** ERROR 009: Wrong Data in Config file POSTGRES, EXIT *********************"
                           ERROR=9
                           break
                   fi
                   echo !!!!! Running  process  $task Data Base  $db !!!!!
                   echo
                   echo !!!!! File size !!!!!
                   ls -lh ${BackupDir} | tail -1 |  awk {'print " File size: "$5 " / File Name: "$9'}
                   echo
                 done
            else
                 echo PGPASSWORD=******** PGSSLMODE=require pg_dumpall --host=$server --username=$username@$1 --exclude-database=azure_sys \
                   --exclude-database=azure_maintenance -f ${BackupDir}/$server.$task.$(date +%Y%m%d%H%M%S).dump
                 PGPASSWORD=${pass} PGSSLMODE=require pg_dumpall --host=$server --username=$username@$1 --exclude-database=azure_sys \
                   --exclude-database=azure_maintenance -f ${BackupDir}/$server.dumpall.$task.$(date +%Y%m%d%H%M%S).dump
            fi
        fi
done  < ${ConfigDir}/resources
```

Redis[14]

This is a long script, please see in repo. https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/backup-redis.sh

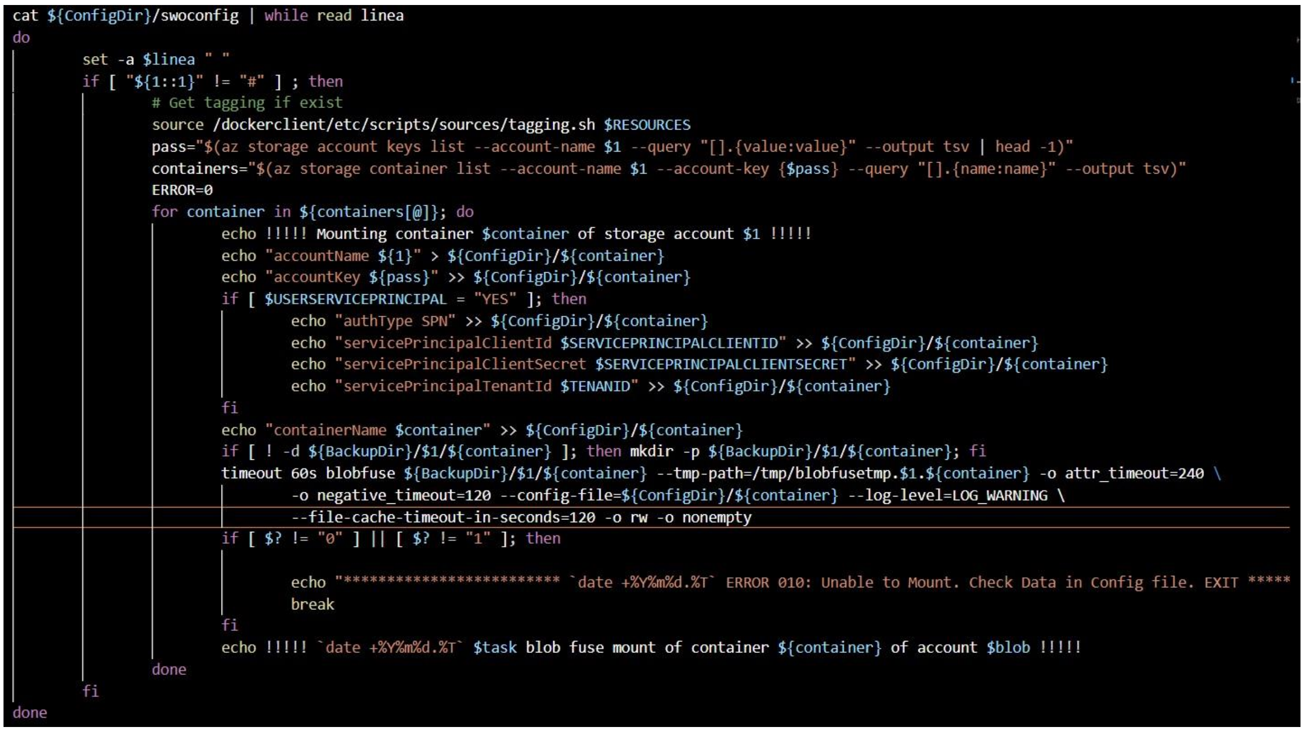
[13] PostgreSQL full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/backup-postgresql.sh
[14] Redis full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/backup-postgresql.sh

ADLS[15]: Here we show the filter we use to protect only Data Lake Gen2 type blob storage. Then the logic of a blob storage is used (not shown in the figure).



```
if [ $AUTODISCOVER = "YES" ]; then
        echo
        echo "*********************************** SEARCHING cloud resources ***************************************"
        echo
        storageaccounts=($(az storage account list --query "[].{name:name}" --output tsv))
        for i in "${storageaccounts[@]}"
        do
                HnsEnabled=($(az storage account show --query isHnsEnabled --name $i | sed 's/"//g'))
                if [ "$HnsEnabled" ]; then echo $i >> ${ConfigDir}/swoconfig; fi
        done
        else
        echo "*********************************** LISTING cloud resources ***************************************"
        echo
            echo $RESOURCELIST_FIX |fmt -1 |sed 's/,//g' >> ${ConfigDir}/swoconfig
fi
```

Blobstorage[16]**:** In this code snippet we can see how to use blobfuse to mount a blob storage.



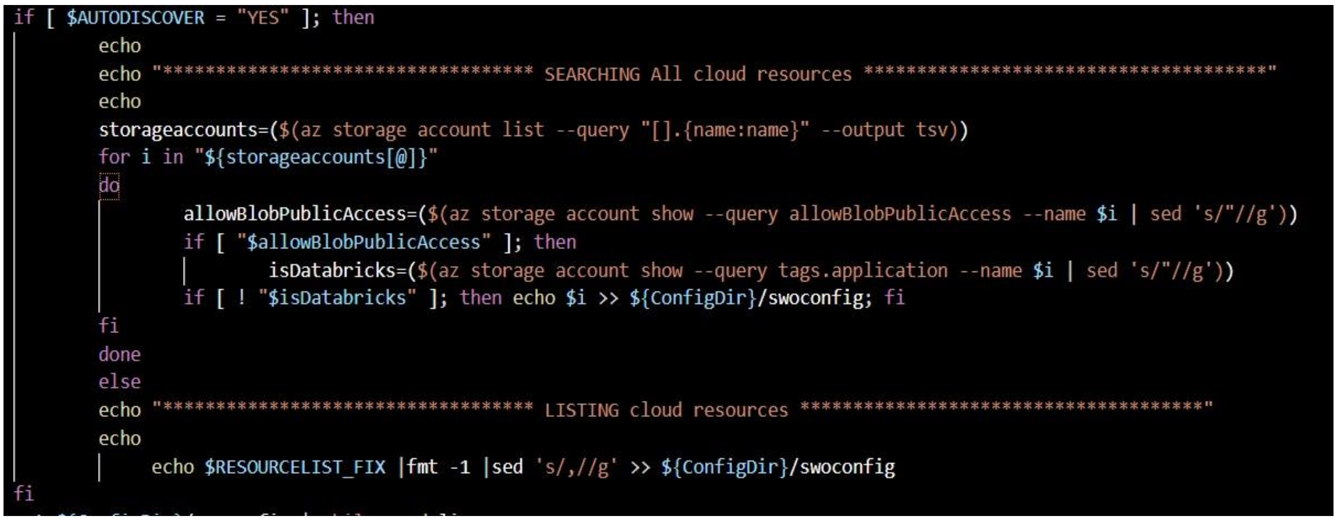
```
cat ${ConfigDir}/swoconfig | while read linea
do
        set -a $linea " "
        if [ "${1::1}" != "#" ] ; then
                # Get tagging if exist
                source /dockerclient/etc/scripts/sources/tagging.sh $RESOURCES
                pass="$(az storage account keys list --account-name $1 --query "[].{value:value}" --output tsv | head -1)"
                containers="$(az storage container list --account-name $1 --account-key {$pass} --query "[].{name:name}" --output tsv)"
                ERROR=0
                for container in ${containers[@]}; do
                        echo !!!!! Mounting container $container of storage account $1 !!!!!
                        echo "accountName ${1}" > ${ConfigDir}/${container}
                        echo "accountKey ${pass}" >> ${ConfigDir}/${container}
                        if [ $USERSERVICEPRINCIPAL = "YES" ]; then
                                echo "authType SPN" >> ${ConfigDir}/${container}
                                echo "servicePrincipalClientId $SERVICEPRINCIPALCLIENTID" >> ${ConfigDir}/${container}
                                echo "servicePrincipalClientSecret $SERVICEPRINCIPALCLIENTSECRET" >> ${ConfigDir}/${container}
                                echo "servicePrincipalTenantId $TENANID" >> ${ConfigDir}/${container}
                        fi
                        echo "containerName $container" >> ${ConfigDir}/${container}
                        if [ ! -d ${BackupDir}/$1/${container} ]; then mkdir -p ${BackupDir}/$1/${container}; fi
                        timeout 60s blobfuse ${BackupDir}/$1/${container} --tmp-path=/tmp/blobfusetmp.$1.${container} -o attr_timeout=240 \
                                -o negative_timeout=120 --config-file=${ConfigDir}/${container} --log-level=LOG_WARNING \
                                --file-cache-timeout-in-seconds=120 -o rw -o nonempty
                        if [ $? != "0" ] || [ $? != "1" ]; then

                                echo "************************ `date +%Y%m%d.%T` ERROR 010: Unable to Mount. Check Data in Config file. EXIT *****
                                break
                        fi
                        echo !!!!! `date +%Y%m%d.%T` $task blob fuse mount of container ${container} of account $blob !!!!!
                done
        fi
done
```

Databricks[17]**:** Here we show the filter we use to protect only Databriks type blob storage. Then the logic of a blob storage is used (not shown in the figure).

[15] Azure Data Lake Gen 2 full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/prebck-adls.sh

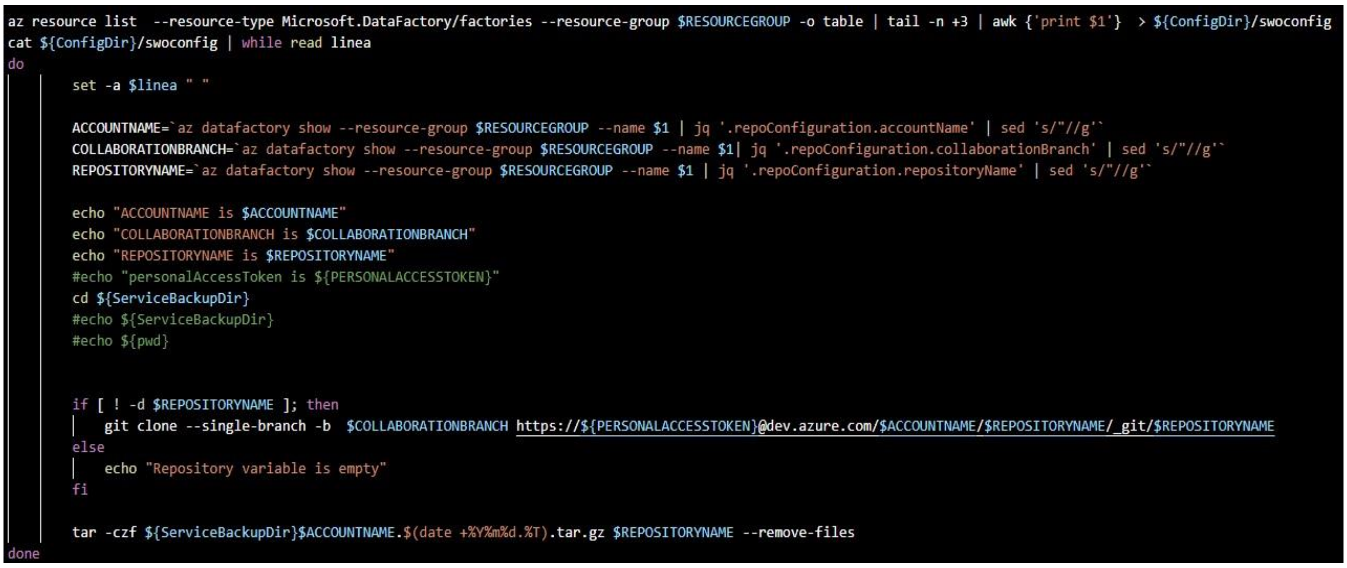
[16] Generic blob storage full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/prebck-blobstorage.sh

[17] Databricks full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/preres-databrick.sh



```
if [ $AUTODISCOVER = "YES" ]; then
        echo
        echo "*********************************** SEARCHING All cloud resources **************************************"
        echo
        storageaccounts=($(az storage account list --query "[].{name:name}" --output tsv))
        for i in "${storageaccounts[@]}"
        do
                allowBlobPublicAccess=($(az storage account show --query allowBlobPublicAccess --name $i | sed 's/"//g'))
                if [ "$allowBlobPublicAccess" ]; then
                        isDatabricks=($(az storage account show --query tags.application --name $i | sed 's/"//g'))
                if [ ! "$isDatabricks" ]; then echo $i >> ${ConfigDir}/swoconfig; fi
        fi
        done
        else
        echo "*********************************** LISTING cloud resources **************************************"
        echo
             echo $RESOURCELIST_FIX |fmt -1 |sed 's/,//g' >> ${ConfigDir}/swoconfig
fi
```

Datafactory[18]: We use the git command to protect the configuration of a Datafactory.

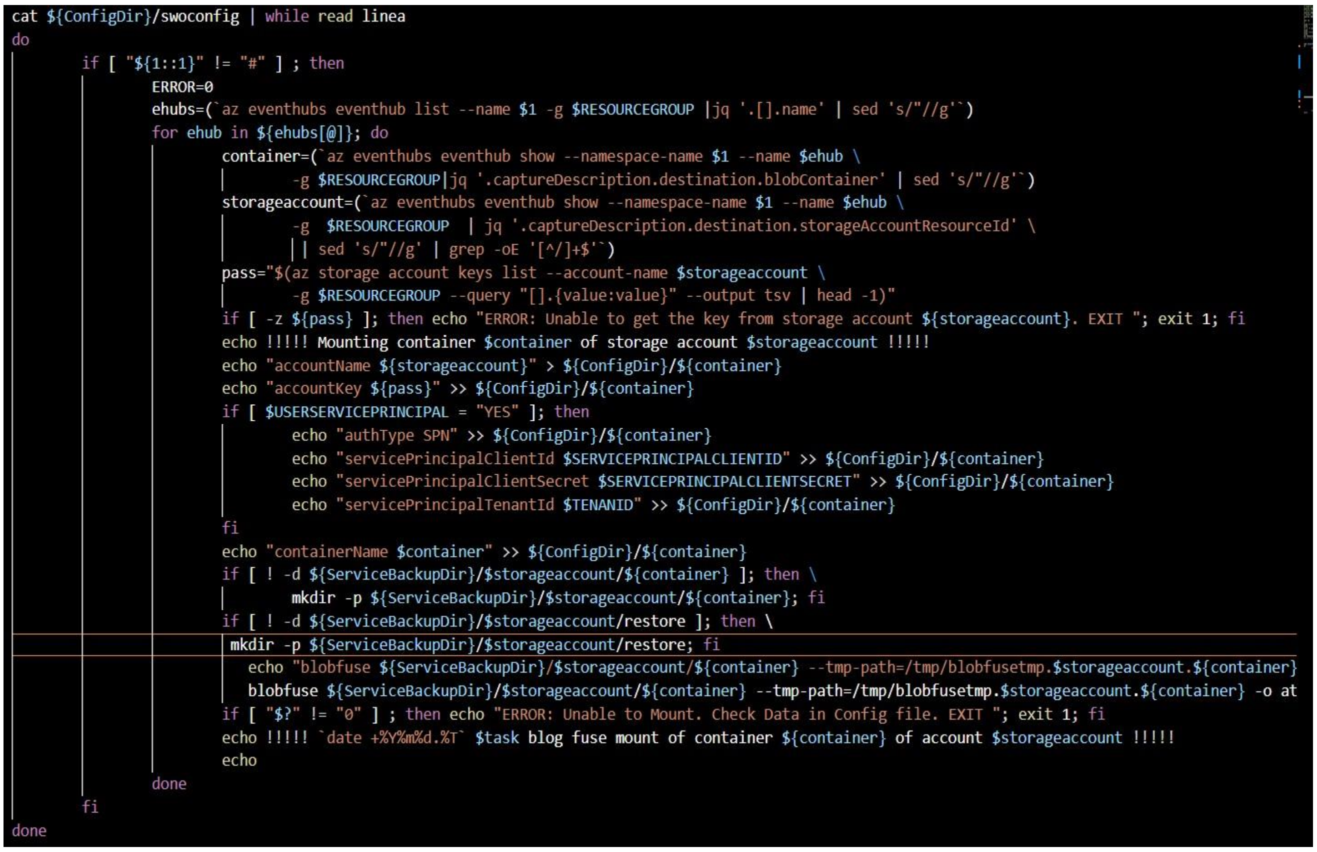


```
az resource list  --resource-type Microsoft.DataFactory/factories --resource-group $RESOURCEGROUP -o table | tail -n +3 | awk {'print $1'}  > ${ConfigDir}/swoconfig
cat ${ConfigDir}/swoconfig | while read linea
do
        set -a $linea " "

        ACCOUNTNAME=`az datafactory show --resource-group $RESOURCEGROUP --name $1 | jq '.repoConfiguration.accountName' | sed 's/"//g'`
        COLLABORATIONBRANCH=`az datafactory show --resource-group $RESOURCEGROUP --name $1| jq '.repoConfiguration.collaborationBranch' | sed 's/"//g'`
        REPOSITORYNAME=`az datafactory show --resource-group $RESOURCEGROUP --name $1 | jq '.repoConfiguration.repositoryName' | sed 's/"//g'`

        echo "ACCOUNTNAME is $ACCOUNTNAME"
        echo "COLLABORATIONBRANCH is $COLLABORATIONBRANCH"
        echo "REPOSITORYNAME is $REPOSITORYNAME"
        #echo "personalAccessToken is ${PERSONALACCESSTOKEN}"
        cd ${ServiceBackupDir}
        #echo ${ServiceBackupDir}
        #echo ${pwd}


        if [ ! -d $REPOSITORYNAME ]; then
            git clone --single-branch -b  $COLLABORATIONBRANCH https://${PERSONALACCESSTOKEN}@dev.azure.com/$ACCOUNTNAME/$REPOSITORYNAME/_git/$REPOSITORYNAME
        else
            echo "Repository variable is empty"
        fi

        tar -czf ${ServiceBackupDir}$ACCOUNTNAME.$(date +%Y%m%d.%T).tar.gz $REPOSITORYNAME --remove-files
done
```

Event Hub[19]: Here we show the filter we use to protect only Event Hubs type blob storage. Then the logic of a blob storage is used (not shown in the figure).

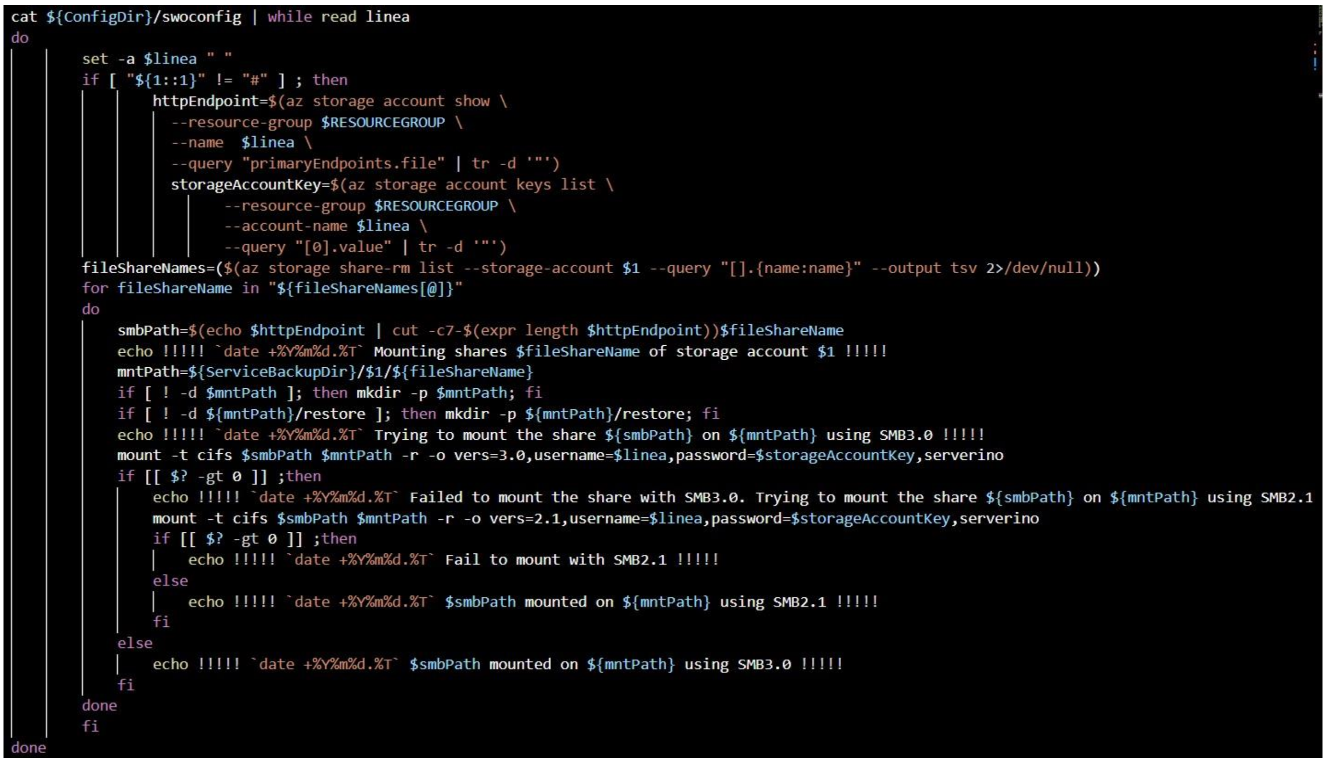
[18] Data Factory full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/main/src/avamar/azure/CLI/backup-datafactory.sh

[19] Event Hubs full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/prebck-eventhub.sh



```
cat ${ConfigDir}/swoconfig | while read linea
do
        if [ "${1::1}" != "#" ] ; then
                ERROR=0
                ehubs=(`az eventhubs eventhub list --name $1 -g $RESOURCEGROUP |jq '.[].name' | sed 's/"//g'`)
                for ehub in ${ehubs[@]}; do
                        container=(`az eventhubs eventhub show --namespace-name $1 --name $ehub \
                                -g $RESOURCEGROUP|jq '.captureDescription.destination.blobContainer' | sed 's/"//g'`)
                        storageaccount=(`az eventhubs eventhub show --namespace-name $1 --name $ehub \
                                -g  $RESOURCEGROUP  | jq '.captureDescription.destination.storageAccountResourceId' \
                                | sed 's/"//g' | grep -oE '[^/]+$'`)
                        pass="$(az storage account keys list --account-name $storageaccount \
                                -g $RESOURCEGROUP --query "[].{value:value}" --output tsv | head -1)"
                        if [ -z ${pass} ]; then echo "ERROR: Unable to get the key from storage account ${storageaccount}. EXIT "; exit 1; fi
                        echo !!!!! Mounting container $container of storage account $storageaccount !!!!!
                        echo "accountName ${storageaccount}" > ${ConfigDir}/${container}
                        echo "accountKey ${pass}" >> ${ConfigDir}/${container}
                        if [ $USERSERVICEPRINCIPAL = "YES" ]; then
                                echo "authType SPN" >> ${ConfigDir}/${container}
                                echo "servicePrincipalClientId $SERVICEPRINCIPALCLIENTID" >> ${ConfigDir}/${container}
                                echo "servicePrincipalClientSecret $SERVICEPRINCIPALCLIENTSECRET" >> ${ConfigDir}/${container}
                                echo "servicePrincipalTenantId $TENANID" >> ${ConfigDir}/${container}
                        fi
                        echo "containerName $container" >> ${ConfigDir}/${container}
                        if [ ! -d ${ServiceBackupDir}/$storageaccount/${container} ]; then \
                                mkdir -p ${ServiceBackupDir}/$storageaccount/${container}; fi
                        if [ ! -d ${ServiceBackupDir}/$storageaccount/restore ]; then \
                         mkdir -p ${ServiceBackupDir}/$storageaccount/restore; fi
                           echo "blobfuse ${ServiceBackupDir}/$storageaccount/${container} --tmp-path=/tmp/blobfusetmp.$storageaccount.${container}
                           blobfuse ${ServiceBackupDir}/$storageaccount/${container} --tmp-path=/tmp/blobfusetmp.$storageaccount.${container} -o at
                        if [ "$?" != "0" ] ; then echo "ERROR: Unable to Mount. Check Data in Config file. EXIT "; exit 1; fi
                        echo !!!!! `date +%Y%m%d.%T` $task blog fuse mount of container ${container} of account $storageaccount !!!!!
                        echo
                done
        fi
done
```

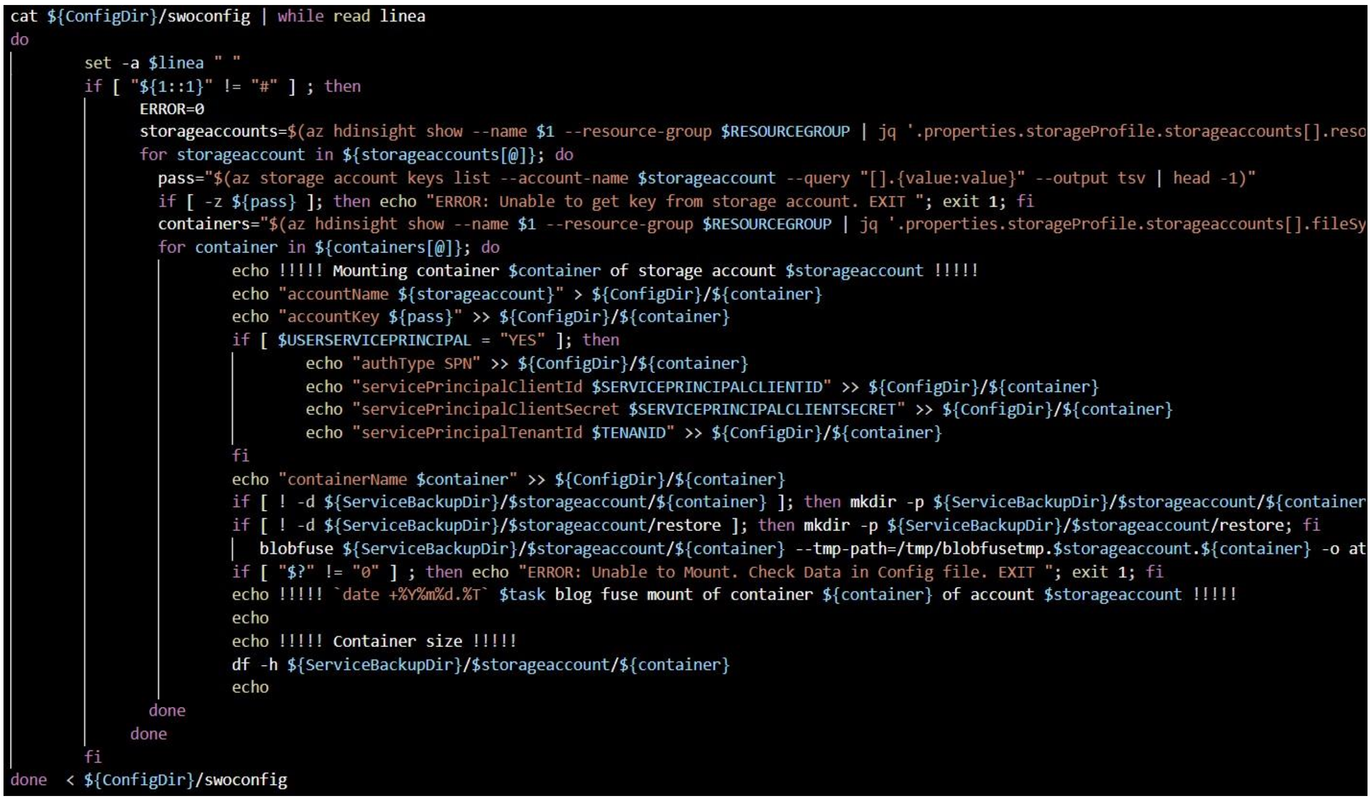
Filestorage[20]: We mount the shared folders through the cifs protocol.



```
cat ${ConfigDir}/swoconfig | while read linea
do
        set -a $linea " "
        if [ "${1::1}" != "#" ] ; then
                httpEndpoint=$(az storage account show \
                  --resource-group $RESOURCEGROUP \
                  --name  $linea \
                  --query "primaryEndpoints.file" | tr -d '"')
                  storageAccountKey=$(az storage account keys list \
                        --resource-group $RESOURCEGROUP \
                        --account-name $linea \
                        --query "[0].value" | tr -d '"')
        fileShareNames=($(az storage share-rm list --storage-account $1 --query "[].{name:name}" --output tsv 2>/dev/null))
        for fileShareName in "${fileShareNames[@]}"
        do
            smbPath=$(echo $httpEndpoint | cut -c7-$(expr length $httpEndpoint))$fileShareName
            echo !!!!! `date +%Y%m%d.%T` Mounting shares $fileShareName of storage account $1 !!!!!
            mntPath=${ServiceBackupDir}/$1/${fileShareName}
            if [ ! -d $mntPath ]; then mkdir -p $mntPath; fi
            if [ ! -d ${mntPath}/restore ]; then mkdir -p ${mntPath}/restore; fi
            echo !!!!! `date +%Y%m%d.%T` Trying to mount the share ${smbPath} on ${mntPath} using SMB3.0 !!!!!
            mount -t cifs $smbPath $mntPath -r -o vers=3.0,username=$linea,password=$storageAccountKey,serverino
            if [[ $? -gt 0 ]] ;then
                echo !!!!! `date +%Y%m%d.%T` Failed to mount the share with SMB3.0. Trying to mount the share ${smbPath} on ${mntPath} using SMB2.1
                mount -t cifs $smbPath $mntPath -r -o vers=2.1,username=$linea,password=$storageAccountKey,serverino
                if [[ $? -gt 0 ]] ;then
                    echo !!!!! `date +%Y%m%d.%T` Fail to mount with SMB2.1 !!!!!
                else
                    echo !!!!! `date +%Y%m%d.%T` $smbPath mounted on ${mntPath} using SMB2.1 !!!!!
                fi
            else
                echo !!!!! `date +%Y%m%d.%T` $smbPath mounted on ${mntPath} using SMB3.0 !!!!!
            fi
        done
        fi
done
```

HDInsight[21]: Here we show the filter we use to protect only HDInsights type blob storage. Then the logic of a blob storage is used (not shown in the figure).

[20] File storage full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/prebck-filestorage.sh
[21] HDInsights full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/prebck-hdinsigth.sh



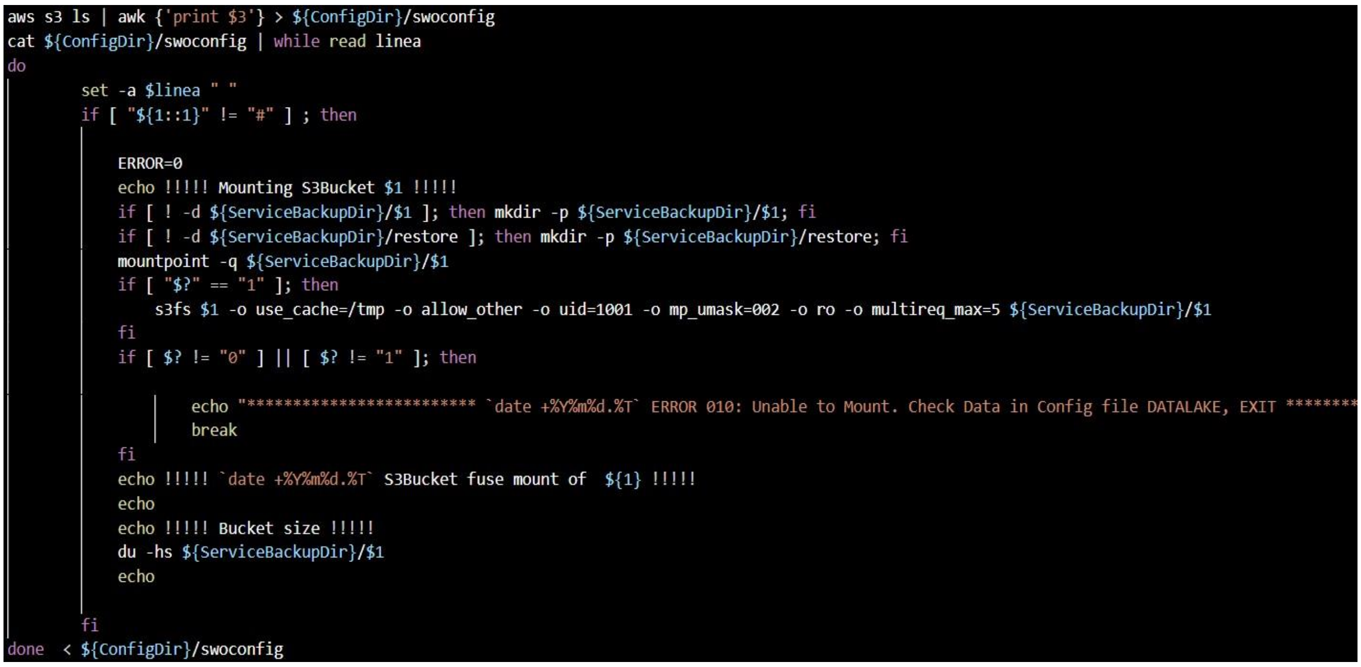
```
cat ${ConfigDir}/swoconfig | while read linea
do
        set -a $linea " "
        if [ "${1::1}" != "#" ] ; then
              ERROR=0
              storageaccounts=$(az hdinsight show --name $1 --resource-group $RESOURCEGROUP | jq '.properties.storageProfile.storageaccounts[].reso
              for storageaccount in ${storageaccounts[@]}; do
                pass="$(az storage account keys list --account-name $storageaccount --query "[].{value:value}" --output tsv | head -1)"
                if [ -z ${pass} ]; then echo "ERROR: Unable to get key from storage account. EXIT "; exit 1; fi
                containers="$(az hdinsight show --name $1 --resource-group $RESOURCEGROUP | jq '.properties.storageProfile.storageaccounts[].fileSy
                for container in ${containers[@]}; do
                        echo !!!!! Mounting container $container of storage account $storageaccount !!!!!
                        echo "accountName ${storageaccount}" > ${ConfigDir}/${container}
                        echo "accountKey ${pass}" >> ${ConfigDir}/${container}
                        if [ $USERSERVICEPRINCIPAL = "YES" ]; then
                                echo "authType SPN" >> ${ConfigDir}/${container}
                                echo "servicePrincipalClientId $SERVICEPRINCIPALCLIENTID" >> ${ConfigDir}/${container}
                                echo "servicePrincipalClientSecret $SERVICEPRINCIPALCLIENTSECRET" >> ${ConfigDir}/${container}
                                echo "servicePrincipalTenantId $TENANID" >> ${ConfigDir}/${container}
                        fi
                        echo "containerName $container" >> ${ConfigDir}/${container}
                        if [ ! -d ${ServiceBackupDir}/$storageaccount/${container} ]; then mkdir -p ${ServiceBackupDir}/$storageaccount/${container
                        if [ ! -d ${ServiceBackupDir}/$storageaccount/restore ]; then mkdir -p ${ServiceBackupDir}/$storageaccount/restore; fi
                           blobfuse ${ServiceBackupDir}/$storageaccount/${container} --tmp-path=/tmp/blobfusetmp.$storageaccount.${container} -o at
                        if [ "$?" != "0" ] ; then echo "ERROR: Unable to Mount. Check Data in Config file. EXIT "; exit 1; fi
                        echo !!!!! `date +%Y%m%d.%T` $task blog fuse mount of container ${container} of account $storageaccount !!!!!
                        echo
                        echo !!!!! Container size !!!!!
                        df -h ${ServiceBackupDir}/$storageaccount/${container}
                        echo
                  done
                done
        fi
done  < ${ConfigDir}/swoconfig
```

NetAppStorage[22]

This is a long script, please see in repo. https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/prebck-netappstorage.sh

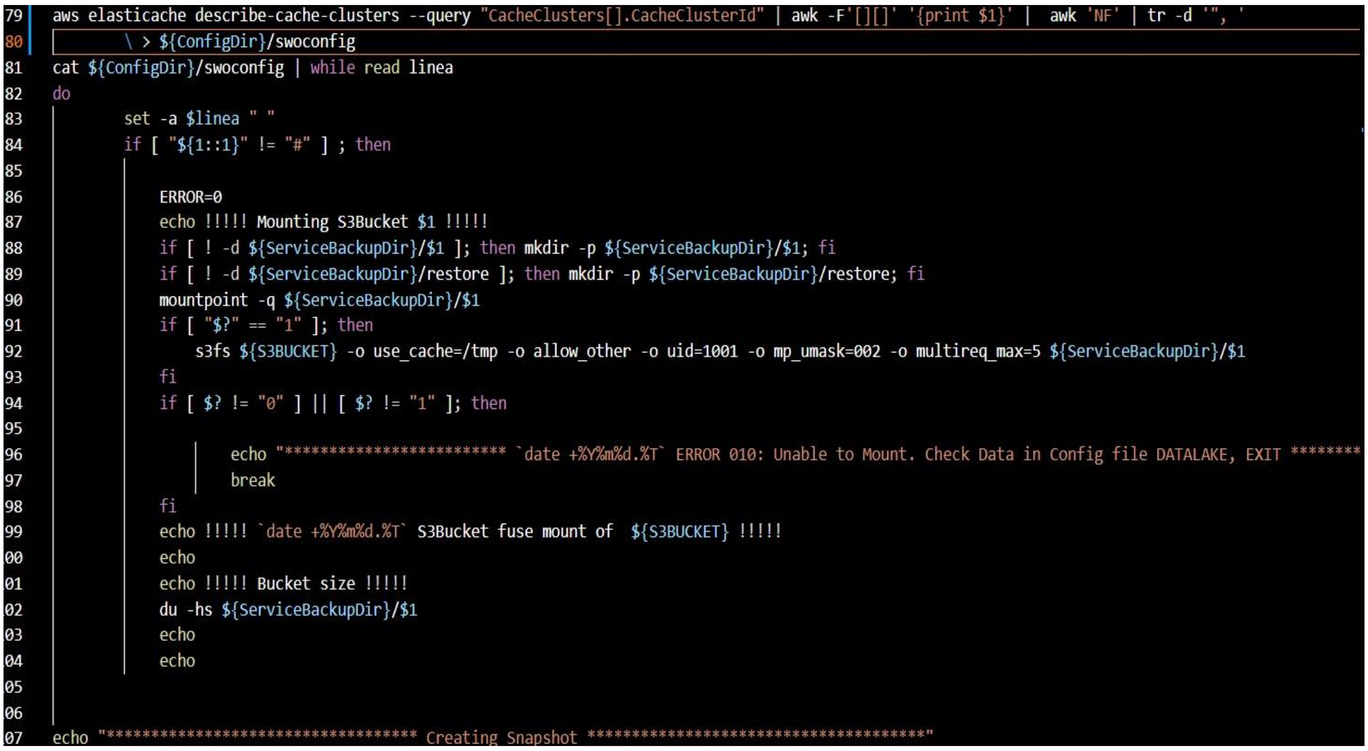
S3[23]**:** In this code snippet we can see how to use s3fs to mount an s3 storage, the s3 resources were previously discovered with AWS CLI commands.

[22] NetApp storage full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/azure/CLI/prebck-netappstorage.sh
[23] S3 full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/aws/CLI/backup-s3.sh



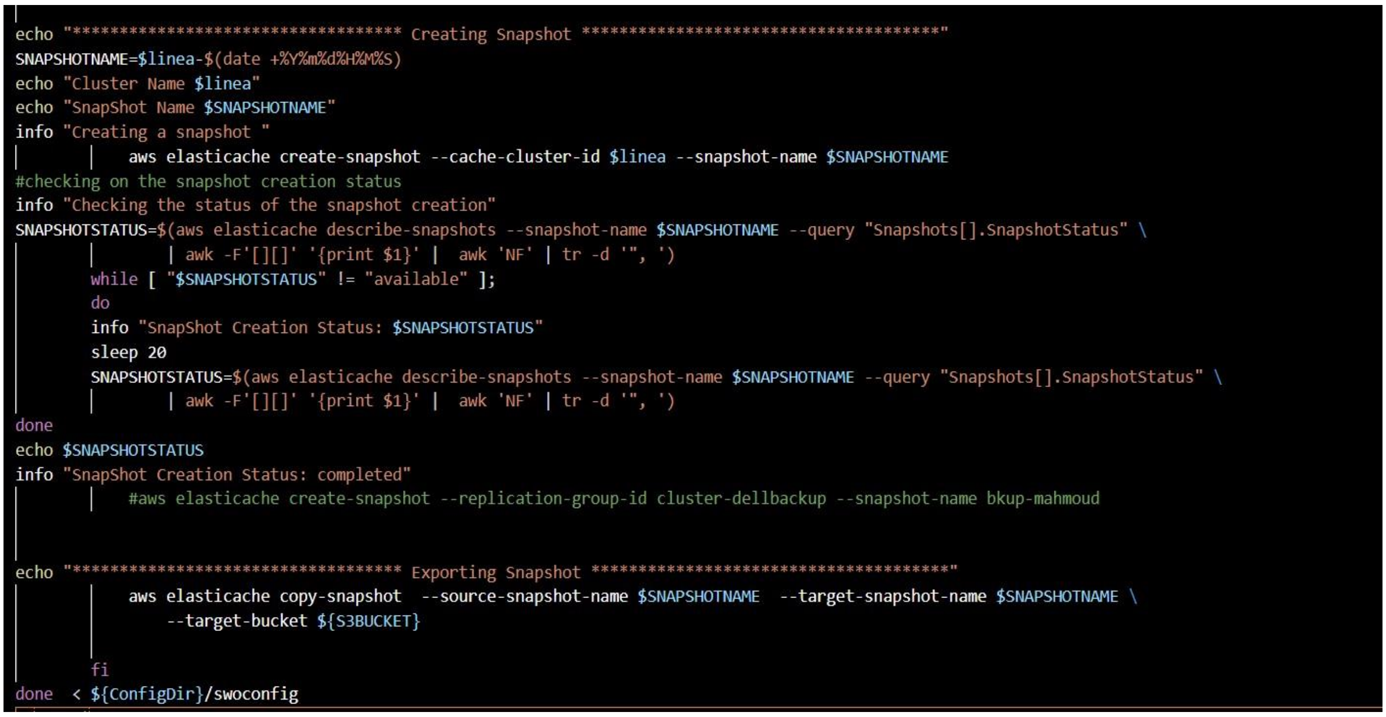
```
aws s3 ls | awk {'print $3'} > ${ConfigDir}/swoconfig
cat ${ConfigDir}/swoconfig | while read linea
do
        set -a $linea " "
        if [ "${1::1}" != "#" ] ; then

            ERROR=0
            echo !!!!! Mounting S3Bucket $1 !!!!!
            if [ ! -d ${ServiceBackupDir}/$1 ]; then mkdir -p ${ServiceBackupDir}/$1; fi
            if [ ! -d ${ServiceBackupDir}/restore ]; then mkdir -p ${ServiceBackupDir}/restore; fi
            mountpoint -q ${ServiceBackupDir}/$1
            if [ "$?" == "1" ]; then
                s3fs $1 -o use_cache=/tmp -o allow_other -o uid=1001 -o mp_umask=002 -o ro -o multireq_max=5 ${ServiceBackupDir}/$1
            fi
            if [ $? != "0" ] || [ $? != "1" ]; then

                echo "************************* `date +%Y%m%d.%T` ERROR 010: Unable to Mount. Check Data in Config file DATALAKE, EXIT ********
                break
            fi
            echo !!!!! `date +%Y%m%d.%T` S3Bucket fuse mount of  ${1} !!!!!
            echo
            echo !!!!! Bucket size !!!!!
            du -hs ${ServiceBackupDir}/$1
            echo

        fi
done  < ${ConfigDir}/swoconfig
```

AWS Redis[24]**:** We use s3 to mount a snapshot of the redis cache.



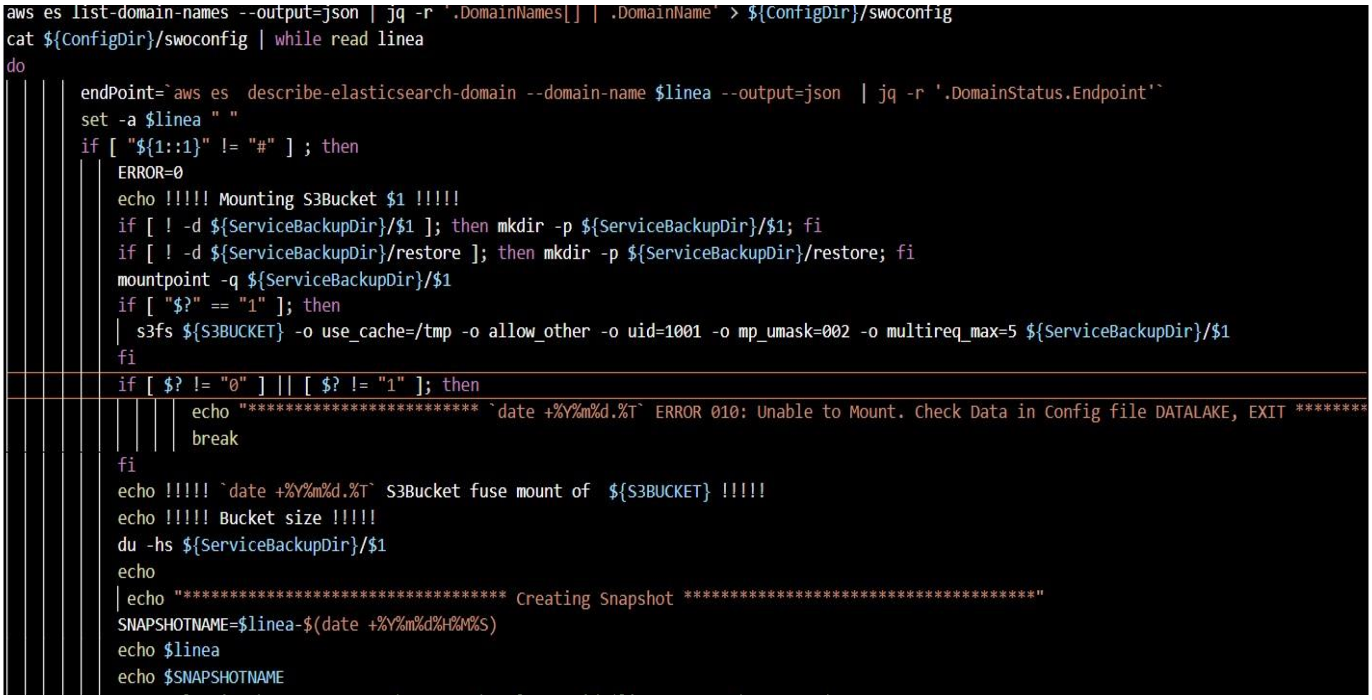
```
aws elasticache describe-cache-clusters --query "CacheClusters[].CacheClusterId" | awk -F'[][]' '{print $1}' |  awk 'NF' | tr -d '", '
        \ > ${ConfigDir}/swoconfig
cat ${ConfigDir}/swoconfig | while read linea
do
        set -a $linea " "
        if [ "${1::1}" != "#" ] ; then

            ERROR=0
            echo !!!!! Mounting S3Bucket $1 !!!!!
            if [ ! -d ${ServiceBackupDir}/$1 ]; then mkdir -p ${ServiceBackupDir}/$1; fi
            if [ ! -d ${ServiceBackupDir}/restore ]; then mkdir -p ${ServiceBackupDir}/restore; fi
            mountpoint -q ${ServiceBackupDir}/$1
            if [ "$?" == "1" ]; then
                s3fs ${S3BUCKET} -o use_cache=/tmp -o allow_other -o uid=1001 -o mp_umask=002 -o multireq_max=5 ${ServiceBackupDir}/$1
            fi
            if [ $? != "0" ] || [ $? != "1" ]; then

                echo "************************* `date +%Y%m%d.%T` ERROR 010: Unable to Mount. Check Data in Config file DATALAKE, EXIT ********
                break
            fi
            echo !!!!! `date +%Y%m%d.%T` S3Bucket fuse mount of  ${S3BUCKET} !!!!!
            echo
            echo !!!!! Bucket size !!!!!
            du -hs ${ServiceBackupDir}/$1
            echo
            echo


echo "************************************ Creating Snapshot *************************************"
```

[24] AWS Redis full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/aws/CLI/backup-AWS-redis.sh



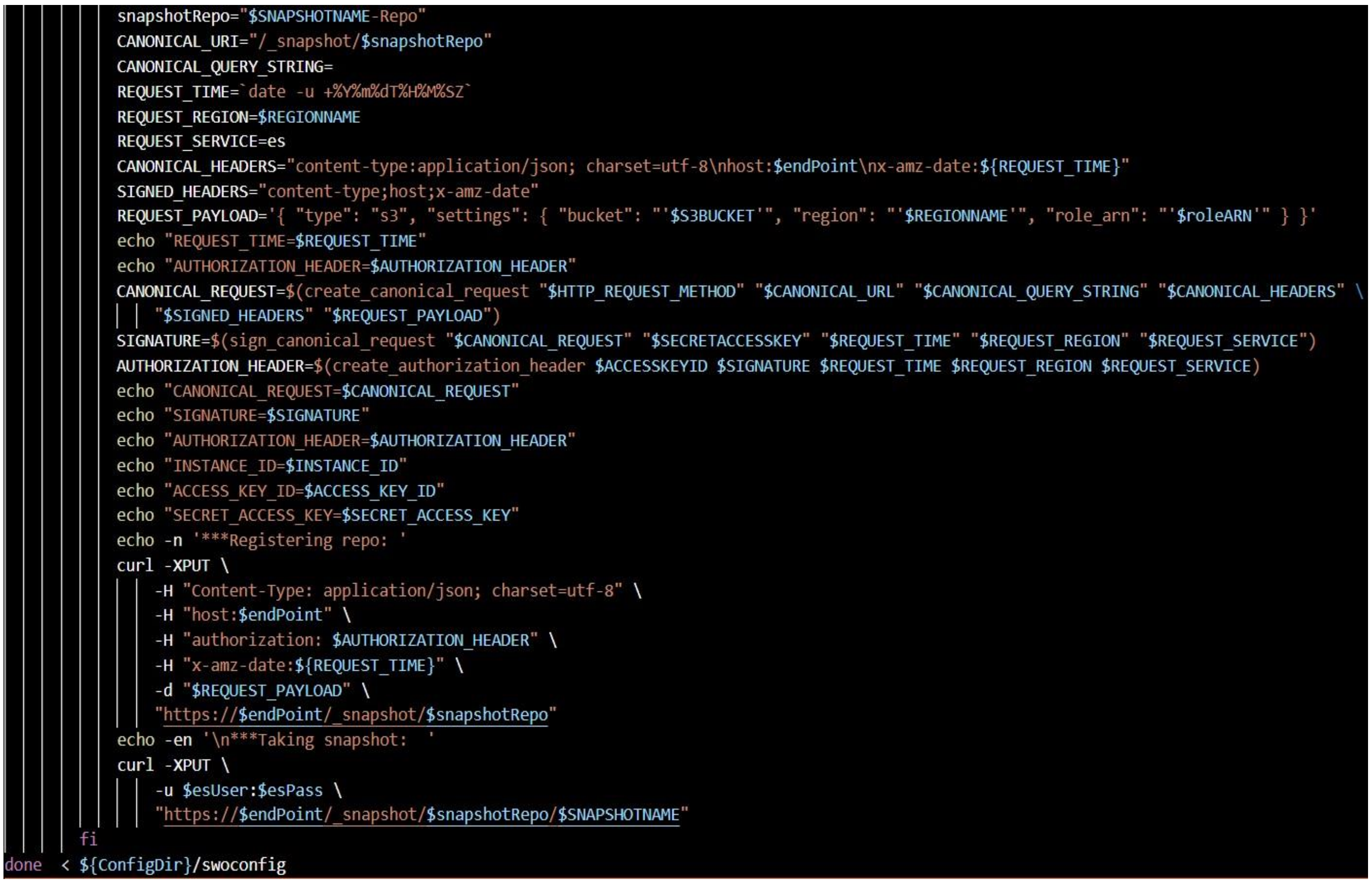
```
echo "************************************ Creating Snapshot ***************************************"
SNAPSHOTNAME=$linea-$(date +%Y%m%d%H%M%S)
echo "Cluster Name $linea"
echo "SnapShot Name $SNAPSHOTNAME"
info "Creating a snapshot "
            aws elasticache create-snapshot --cache-cluster-id $linea --snapshot-name $SNAPSHOTNAME
#checking on the snapshot creation status
info "Checking the status of the snapshot creation"
SNAPSHOTSTATUS=$(aws elasticache describe-snapshots --snapshot-name $SNAPSHOTNAME --query "Snapshots[].SnapshotStatus" \
                | awk -F'[][]' '{print $1}' |  awk 'NF' | tr -d '", ')
        while [ "$SNAPSHOTSTATUS" != "available" ];
        do
        info "SnapShot Creation Status: $SNAPSHOTSTATUS"
        sleep 20
        SNAPSHOTSTATUS=$(aws elasticache describe-snapshots --snapshot-name $SNAPSHOTNAME --query "Snapshots[].SnapshotStatus" \
                | awk -F'[][]' '{print $1}' |  awk 'NF' | tr -d '", ')
done
echo $SNAPSHOTSTATUS
info "SnapShot Creation Status: completed"
            #aws elasticache create-snapshot --replication-group-id cluster-dellbackup --snapshot-name bkup-mahmoud


echo "************************************ Exporting Snapshot ***************************************"
            aws elasticache copy-snapshot  --source-snapshot-name $SNAPSHOTNAME  --target-snapshot-name $SNAPSHOTNAME \
                --target-bucket ${S3BUCKET}

        fi
done  < ${ConfigDir}/swoconfig
```

AWS Elasticsearch[25]: We use s3 to mount a snapshot of the elasticsearch.



```
aws es list-domain-names --output=json | jq -r '.DomainNames[] | .DomainName' > ${ConfigDir}/swoconfig
cat ${ConfigDir}/swoconfig | while read linea
do
        endPoint=`aws es  describe-elasticsearch-domain --domain-name $linea --output=json  | jq -r '.DomainStatus.Endpoint'`
        set -a $linea " "
        if [ "${1::1}" != "#" ] ; then
            ERROR=0
            echo !!!!! Mounting S3Bucket $1 !!!!!
            if [ ! -d ${ServiceBackupDir}/$1 ]; then mkdir -p ${ServiceBackupDir}/$1; fi
            if [ ! -d ${ServiceBackupDir}/restore ]; then mkdir -p ${ServiceBackupDir}/restore; fi
            mountpoint -q ${ServiceBackupDir}/$1
            if [ "$?" == "1" ]; then
              s3fs ${S3BUCKET} -o use_cache=/tmp -o allow_other -o uid=1001 -o mp_umask=002 -o multireq_max=5 ${ServiceBackupDir}/$1
            fi
            if [ $? != "0" ] || [ $? != "1" ]; then
                    echo "************************* `date +%Y%m%d.%T` ERROR 010: Unable to Mount. Check Data in Config file DATALAKE, EXIT ********
                    break
            fi
            echo !!!!! `date +%Y%m%d.%T` S3Bucket fuse mount of  ${S3BUCKET} !!!!!
            echo !!!!! Bucket size !!!!!
            du -hs ${ServiceBackupDir}/$1
            echo
              echo "************************************ Creating Snapshot ***************************************"
            SNAPSHOTNAME=$linea-$(date +%Y%m%d%H%M%S)
            echo $linea
            echo $SNAPSHOTNAME
```

[25] Elasticsearch full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/sharing-competition/src/avamar/aws/CLI/backup-AWS-ES.sh



```
            snapshotRepo="$SNAPSHOTNAME-Repo"
            CANONICAL_URI="/_snapshot/$snapshotRepo"
            CANONICAL_QUERY_STRING=
            REQUEST_TIME=`date -u +%Y%m%dT%H%M%SZ`
            REQUEST_REGION=$REGIONNAME
            REQUEST_SERVICE=es
            CANONICAL_HEADERS="content-type:application/json; charset=utf-8\nhost:$endPoint\nx-amz-date:${REQUEST_TIME}"
            SIGNED_HEADERS="content-type;host;x-amz-date"
            REQUEST_PAYLOAD='{ "type": "s3", "settings": { "bucket": "'$S3BUCKET'", "region": "'$REGIONNAME'", "role_arn": "'$roleARN'" } }'
            echo "REQUEST_TIME=$REQUEST_TIME"
            echo "AUTHORIZATION_HEADER=$AUTHORIZATION_HEADER"
            CANONICAL_REQUEST=$(create_canonical_request "$HTTP_REQUEST_METHOD" "$CANONICAL_URL" "$CANONICAL_QUERY_STRING" "$CANONICAL_HEADERS" \
                "$SIGNED_HEADERS" "$REQUEST_PAYLOAD")
            SIGNATURE=$(sign_canonical_request "$CANONICAL_REQUEST" "$SECRETACCESSKEY" "$REQUEST_TIME" "$REQUEST_REGION" "$REQUEST_SERVICE")
            AUTHORIZATION_HEADER=$(create_authorization_header $ACCESSKEYID $SIGNATURE $REQUEST_TIME $REQUEST_REGION $REQUEST_SERVICE)
            echo "CANONICAL_REQUEST=$CANONICAL_REQUEST"
            echo "SIGNATURE=$SIGNATURE"
            echo "AUTHORIZATION_HEADER=$AUTHORIZATION_HEADER"
            echo "INSTANCE_ID=$INSTANCE_ID"
            echo "ACCESS_KEY_ID=$ACCESS_KEY_ID"
            echo "SECRET_ACCESS_KEY=$SECRET_ACCESS_KEY"
            echo -n '***Registering repo: '
            curl -XPUT \
                -H "Content-Type: application/json; charset=utf-8" \
                -H "host:$endPoint" \
                -H "authorization: $AUTHORIZATION_HEADER" \
                -H "x-amz-date:${REQUEST_TIME}" \
                -d "$REQUEST_PAYLOAD" \
                "https://$endPoint/_snapshot/$snapshotRepo"
            echo -en '\n***Taking snapshot:  '
            curl -XPUT \
                -u $esUser:$esPass \
                "https://$endPoint/_snapshot/$snapshotRepo/$SNAPSHOTNAME"
        fi
done  < ${ConfigDir}/swoconfig
```

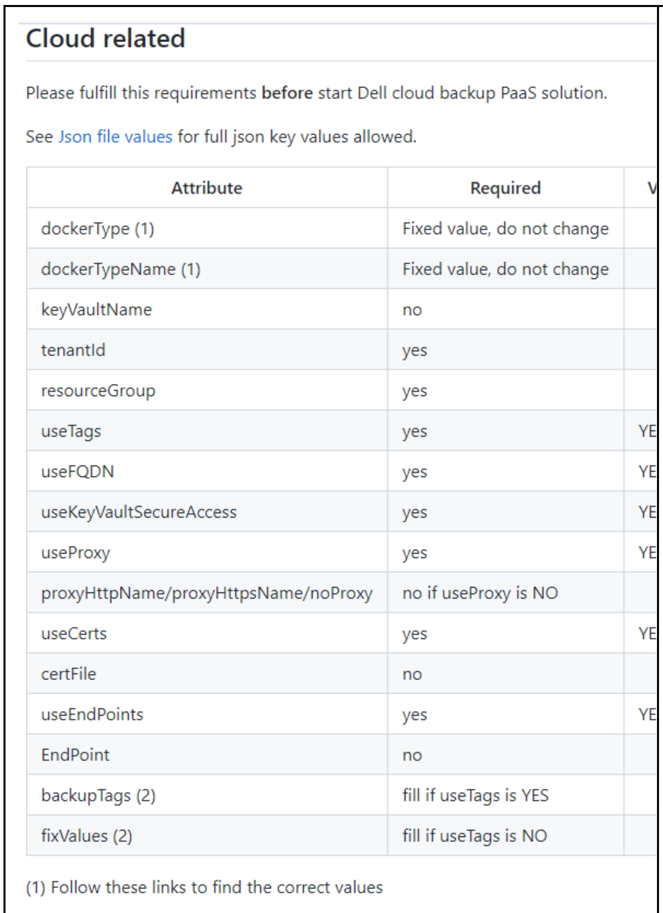
Appendix E: Json file config guideline

The json files (example dps-setup.(dockerType).json) contain the keys to be configured to deploy a new commercial container. Please see these guidelines to fulfill this file type.

- cloudProvider `Azure/AWS`
- dockerType `azsql/blobstorage/cosmosql/cvision/databriks/datafactory/eventhub/filestorage/keyvault/mongodb/atlas/netappstorage/postgresql/redis`
- dockerTypeName `AZSQL/BS/CSQL/CV/DB/DF/EH/FS/KV/MG/MGA/NS/PG/RD`
- keyVaultName `Azure Key Vault name`
- tenantId `Tenantid`
- resourceGroup `resurce group name or all to all RGs`
- useTags `tags or default values using fixValues`
- useFQDN `FQDN or IP through nslookup`
- useKeyVaultSecureAccess `Keyvault access using curl (YES) or az cli (NO)`
- useProxy `YES if docker file needs proxy ENV variables otherwise NO`
- proxyHttpName and proxyHttpsName `Proxies FQDN and port values`
- noProxy `No proxy for FQDNs (comma separated)`
- changeDefaultsubscription `YES to change from default subscription`
- subscriptionID `Subuscrition ID`
- useCerts `YES if certificate is needed otherwise NO`
- cers `Certificate name or * to include all src/packages/DockerEmbebed/certificates/`
- useServicePrincipal `YES for SPN otherwise NO`
- servicePrincipalClientId `Service principal client id`
- servicePrincipalClientSecret `Service principal client password`
- useEndPoints `YES if endpoints are used otherwise NO`
- EndPoint `Endpoint FQDN or IP`
- useAvamar `Use avamar to store backup data`
- avamarClientPort `Avamar client port, FROM 28003`
- avamarServerName and datadomainServerName `Avamar and Data Domain FQDN`
- avamarDomain `Avamar docker domain, eg. clients`
- avamarVersion `Avamar version`
- createSchedule | createRetention `Unused`
- createDataset | createGroup `Complete to create dataset and group on avamar`
- mountType `ddboostfs`
- RootBackupDir `DDBoostFS or local mount point on container`
- storageUnit `Data Domain Storage Unit used to hold data`
- ddboostuser `ddboost user used to connect this container to DD`
- containerName `FQDN of container used to register this client on Avamar. Add forward and reverse DNS records to DNS Server Values="$"cloudProvider"-docker-$"dockerType-$Fix_value_Incremental"`
- resourceType `Azure resource type to be discover`
- backupTags \ Type `Type of tag, values: AZSQL/BS/CSQL/CV/DB/DF/EH/FS/KV/MG/MGA/NS/PG/RD`
- backupTags \ Value `Value of type tag`
- fixValues \ Type `Type of tag, values: AZSQL/BS/CSQL/CV/DB/DF/EH/FS/KV/MG/MGA/NS/PG/RD`
- fixValues \ Type `Hardcoded value`

Use case-specific keys

Cloud, Avamar, Data Domain and Container requirements



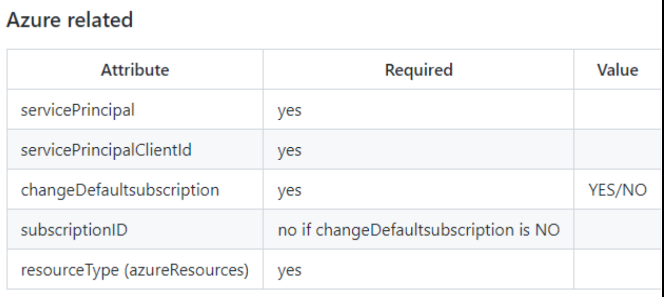
Cloud related

Please fulfill this requirements **before** start Dell cloud backup PaaS solution.

See Json file values for full json key values allowed.

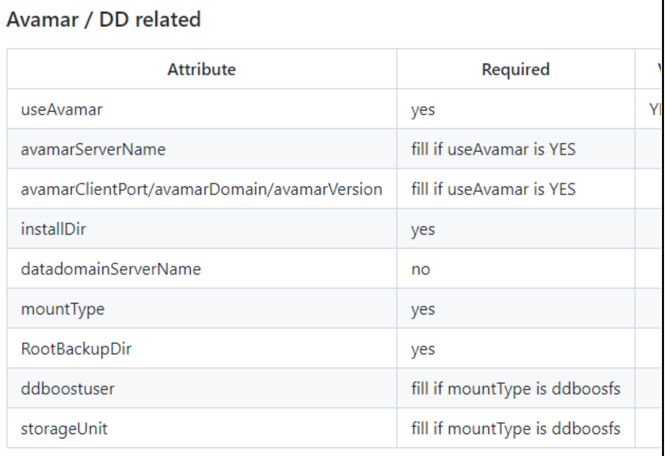
Attribute	Required	V
dockerType (1)	Fixed value, do not change	
dockerTypeName (1)	Fixed value, do not change	
keyVaultName	no	
tenantId	yes	
resourceGroup	yes	
useTags	yes	YE
useFQDN	yes	YE
useKeyVaultSecureAccess	yes	YE
useProxy	yes	YE
proxyHttpName/proxyHttpsName/noProxy	no if useProxy is NO	
useCerts	yes	YE
certFile	no	
useEndPoints	yes	YE
EndPoint	no	
backupTags (2)	fill if useTags is YES	
fixValues (2)	fill if useTags is NO	

(1) Follow these links to find the correct values



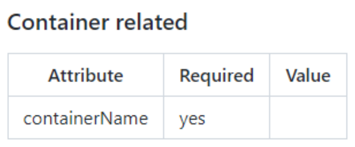
Azure related

Attribute	Required	Value
servicePrincipal	yes	
servicePrincipalClientId	yes	
changeDefaultsubscription	yes	YES/NO
subscriptionID	no if changeDefaultsubscription is NO	
resourceType (azureResources)	yes	



Avamar / DD related

Attribute	Required	V
useAvamar	yes	Y
avamarServerName	fill if useAvamar is YES	
avamarClientPort/avamarDomain/avamarVersion	fill if useAvamar is YES	
installDir	yes	
datadomainServerName	no	
mountType	yes	
RootBackupDir	yes	
ddboostuser	fill if mountType is ddboosfs	
storageUnit	fill if mountType is ddboosfs	



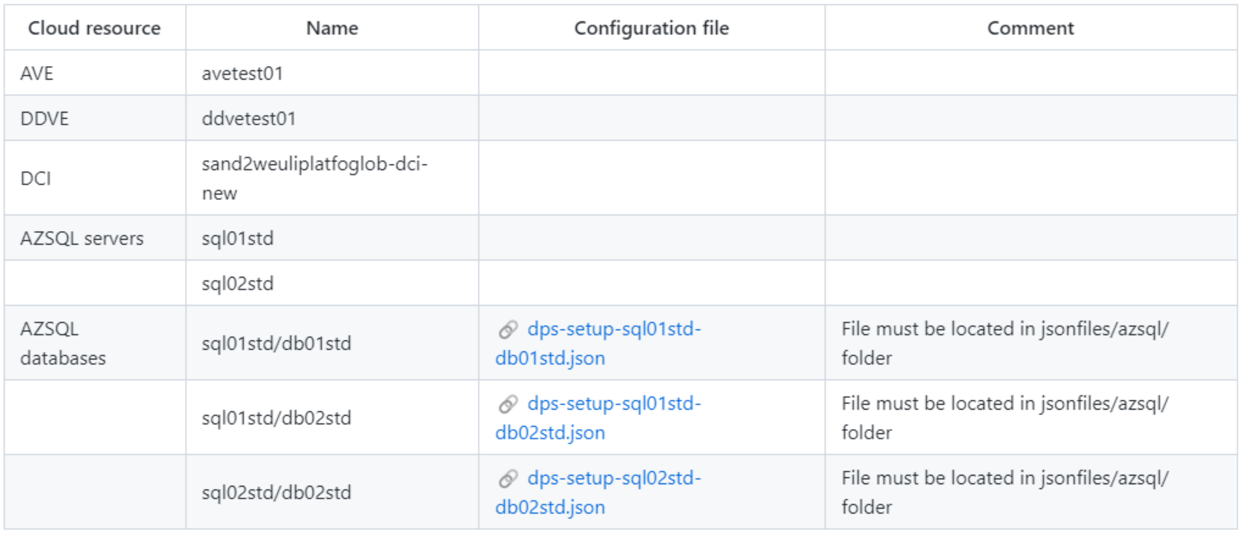
Container related

Attribute	Required	Value
containerName	yes	

Please see this link for use case-specific requirements: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/tree/main/docs/UseCaseRequirements

Appendix F: Ephemeral containers implementation example

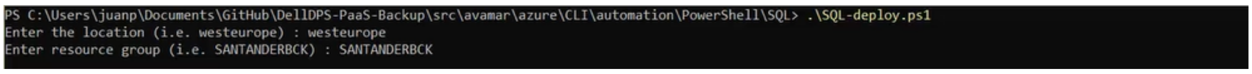
This is the configuration used for the demo.



Cloud resource	Name	Configuration file	Comment
AVE	avetest01		
DDVE	ddvetest01		
DCI	sand2weuliplatfoglob-dci-new		
AZSQL servers	sql01std		
	sql02std		
AZSQL databases	sql01std/db01std	dps-setup-sql01std-db01std.json	File must be located in jsonfiles/azsql/ folder
	sql01std/db02std	dps-setup-sql01std-db02std.json	File must be located in jsonfiles/azsql/ folder
	sql02std/db02std	dps-setup-sql02std-db02std.json	File must be located in jsonfiles/azsql/ folder

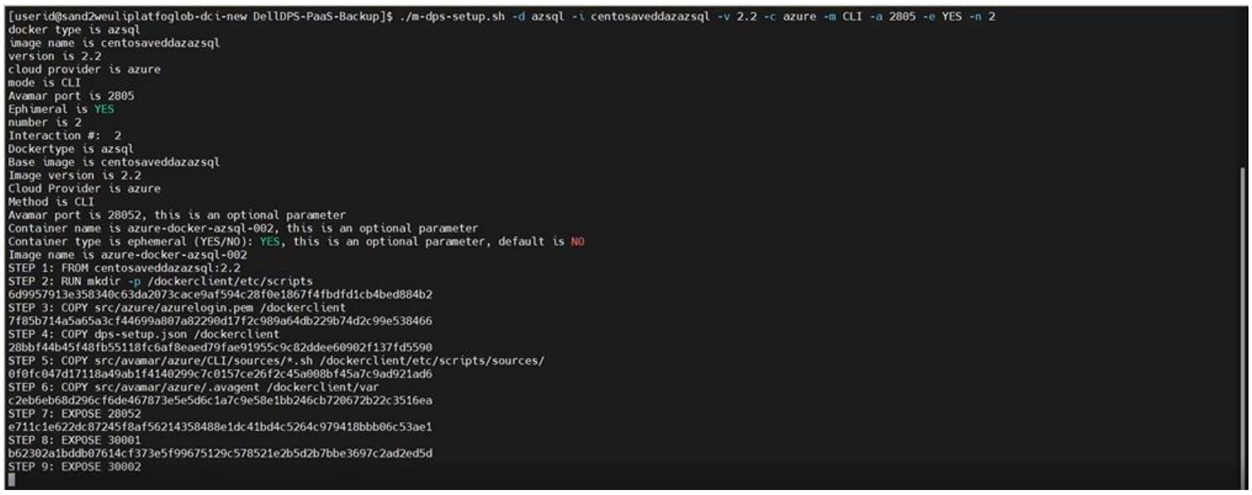
Full demo: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/main/docs/EphemeralContainers.md

Azure SQL resource creation: This video shows the Azure SQL servers and databases creation using a PowerShell script[26]. This demo backs up two databases on server sql01std and one database on server sql02std.



```
PS C:\Users\juanp\Documents\GitHub\DellDPS-PaaS-Backup\src\avamar\azure\CLI\automation\PowerShell\SQL> .\SQL-deploy.ps1
Enter the location (i.e. westeurope) : westeurope
Enter resource group (i.e. SANTANDERBCK) : SANTANDERBCK
```

Docker container creation: We use the ***m-dps-setup script***[27] to create them:

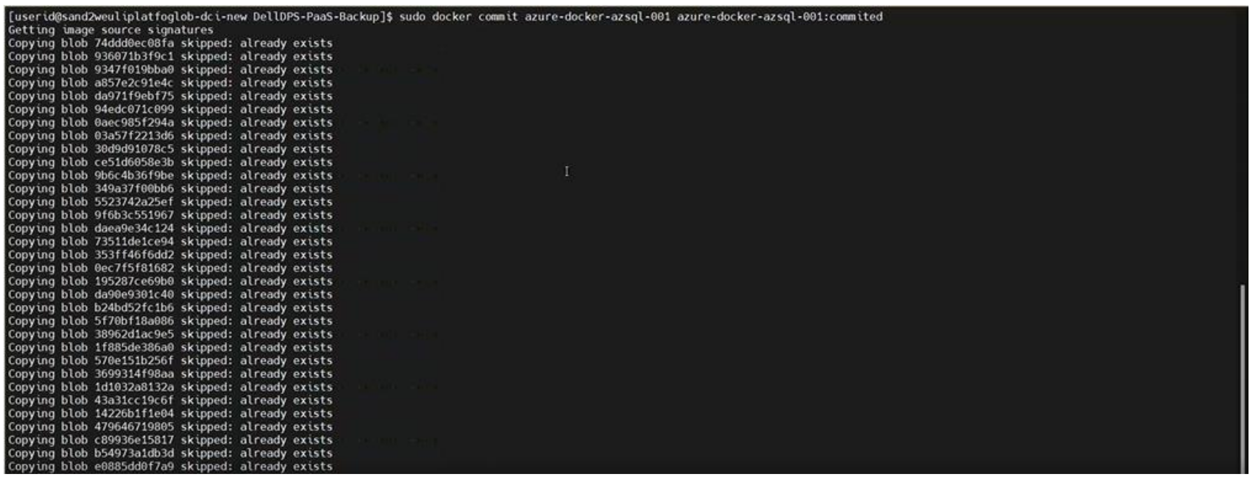


```
[userid@sand2weuliplatfoglob-dci-new DellDPS-PaaS-Backup]$ ./m-dps-setup.sh -d azsql -i centosaveddazazsql -v 2.2 -c azure -m CLI -a 2805 -e YES -n 2
docker type is azsql
image name is centosaveddazazsql
version is 2.2
cloud provider is azure
mode is CLI
Avamar port is 2805
Ephimeral is YES
number is 2
Interaction #:  2
Dockertype is azsql
Base image is centosaveddazazsql
Image version is 2.2
Cloud Provider is azure
Method is CLI
Avamar port is 28052, this is an optional parameter
Container name is azure-docker-azsql-002, this is an optional parameter
Container type is ephemeral (YES/NO): YES, this is an optional parameter, default is NO
Image name is azure-docker-azsql-002
STEP 1: FROM centosaveddazazsql:2.2
STEP 2: RUN mkdir -p /dockerclient/etc/scripts
6d9957913e358340c63da2073cace9af594c28f0e1867f4fbdfd1cb4bed884b2
STEP 3: COPY src/azure/azurelogin.pem /dockerclient
7f85b714a5a65a3cf44699a807a82290d17f2c989a64db229b74d2c99e538466
STEP 4: COPY dps-setup.json /dockerclient
28bbf44b45f48fb55118fc6af8eaed79fae91955c9c82ddee60902f137fd5590
STEP 5: COPY src/avamar/azure/CLI/sources/*.sh /dockerclient/etc/scripts/sources/
0f0fc047d17118a49ab1f4140299c7c0157ce26f2c45a008bf45a7c9ad921ad6
STEP 6: COPY src/avamar/azure/.avagent /dockerclient/var
c2eb6eb68d296cf6de467873e5e5d6c1a7c9e58e1bb246cb720672b22c3516ea
STEP 7: EXPOSE 28052
e711c1e622dc87245f8af56214358488e1dc41bd4c5264c979418bbb06c53ae1
STEP 8: EXPOSE 30001
b62302a1bddb07614cf373e5f99675129c578521e2b5d2b7bbe3697c2ad2ed5d
STEP 9: EXPOSE 30002
```

Commit Docker container as committed images: The recently created containers have been

[26] AZ SQL provisioning full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/main/src/avamar/azure/CLI/automation/PowerShell/SQL/SQL-deploy.ps1
[27] m-dps-setup.sh mete script full code: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/main/m-dps-setup.sh

registered as Avamar and Data Domain clients. We are going to save this configuration as "committed" images to be used in ephemeral containers mode.



```
[userid@sand2weuliplatfoglob-dci-new DellDPS-PaaS-Backup]$ sudo docker commit azure-docker-azsql-001 azure-docker-azsql-001:commited
Getting image source signatures
Copying blob 74ddd0ec08fa skipped: already exists
Copying blob 936071b3f9c1 skipped: already exists
Copying blob 9347f019bba0 skipped: already exists
Copying blob a857e2c91e4c skipped: already exists
Copying blob da971f9ebf75 skipped: already exists
Copying blob 94edc071c099 skipped: already exists
Copying blob 0aec985f294a skipped: already exists
Copying blob 03a57f2213d6 skipped: already exists
Copying blob 30d9d91078c5 skipped: already exists
Copying blob ce51d6058e3b skipped: already exists
Copying blob 9b6c4b36f9be skipped: already exists
Copying blob 349a37f00bb6 skipped: already exists
Copying blob 5523742a25ef skipped: already exists
Copying blob 9f6b3c551967 skipped: already exists
Copying blob daea9e34c124 skipped: already exists
Copying blob 73511de1ce94 skipped: already exists
Copying blob 353ff46f6dd2 skipped: already exists
Copying blob 0ec7f5f81682 skipped: already exists
Copying blob 195287ce69b0 skipped: already exists
Copying blob da90e9301c40 skipped: already exists
Copying blob b24bd52fc1b6 skipped: already exists
Copying blob 5f70bf18a086 skipped: already exists
Copying blob 38962d1ac9e5 skipped: already exists
Copying blob 1f885de386a0 skipped: already exists
Copying blob 570e151b256f skipped: already exists
Copying blob 3699314f98aa skipped: already exists
Copying blob 1d1032a8132a skipped: already exists
Copying blob 43a31cc19c6f skipped: already exists
Copying blob 14226b1f1e04 skipped: already exists
Copying blob 479646719805 skipped: already exists
Copying blob c89936e15817 skipped: already exists
Copying blob b54973a1db3d skipped: already exists
Copying blob e0885dd0f7a9 skipped: already exists
```

Avamar policy creation: We create a dataset to back up a Linux File System plug-in whose Pre-Script will be upcontainers.sh azsql, the source data will be /usr/local/avamar/var/backups-azsql.sh. The policy will use the previously created dataset with only one member that will be the DCI.

Avamar client registration: Note that there are two new clients in Avamar and they correspond to the two recently created containers. The m-dps-setup did this task.

Avamar policy execution: The policy starts the main thread on the DCI's Avamar client, this launches child threads that run in the ephemeral containers (#2) and process the JSON configuration files (#3)

Avamar backup records: This shows the Avamar backup records.

Appendix G: Requirements when use DDboostFS

a) Create DD Boost user (Data Domain side - sysadmin access or similar is required)

```
- user add <DDBoost user> role user
- user password aging show
- user password aging set <DDBoost user> max-days-between-change 99999
```

b) Create storage unit (Data Domain side - sysadmin access or similar is required)

```
- ddboost storage-unit create <storage-unit name> user <DDboost user>
```

References

1. Ephemeral container demo page: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup/blob/main/docs/EphemeralContainers.md

2. Code repository: https://github.com/ps-iberia-public-cloud-backup/DellDPS-PaaS-Backup. This is a private repo, please request an invite to pablo.calvo@dell.com.