FlashCloud – Architecture Overview

Introduction

The Avere FXT Edge Filer with FlashCloud enables cloud storage platforms to be destinations for storage (core filers) over NFS and SMB. As of version 4.6.2.3, the FXT supports Amazon Web Services (AWS), Google Cloud Platform (GCP), Amplidata/HGST, Cleversafe/IBM, and SwiftStack as cloud storage targets. For a complete list of supported core filers, please check with Avere Global Support or your account team.

Each object storage vendor has different nuances in the way it handles object additions, removal, consistency and security. FlashCloud has been built to provide object consistency and security regardless of platform. This document examines how the FXT with FlashCloud manages objects in cloud storage environments.

Object Creation

The Amazon S3-compatible API has become the de facto standard for cloud object storage. The FXT uses vendor S3 APIs via REST to manage storage on supported cloud storage targets. Common commands include PUT, GET, and LIST.

Charges are incurred from public cloud vendors for issuing commands as well as the amount of storage consumed and retrieved from cloud environments. Each cloud vendor has their own charging structure for what are termed “egress charges”, which means retrieval from the cloud storage. One way FlashCloud helps to minimize public cloud costs is by segmenting data into smaller objects.

Object Sizes

FlashCloud divides objects greater than 1MB into chunks of 1MB. For example, a 8.5MB file would be broken into eight 1MB objects and one .5MB object. This allows updates to only affect specific objects.

If a user was working on a 25MB spreadsheet, for example, and only needed to add data to the end of the file, FlashCloud would only need to update a portion of the file. Instead of removing a 25MB file from the cloud and rewriting a new 25MB object, FlashCloud will only change the blocks that were affected by the user’s changes. This results in only changing one or two 1MB objects instead of one large 25MB object. The benefit of this segmentation is even greater with file sizes into gigabytes or even terabytes.

Object Naming

FlashCloud users don't want object names mirroring file names. For example, they would not want payroll.db to be segmented into payroll.db.1, payroll.db.2, etc. FlashCloud objects are named using pseudo-random hexadecimal characters (0-9, A-F). This way, those same payroll objects might have names that look more like 61FA175A_31433C59042702003FCA5A00000F671B63AFD1BAB55.00000000010.FFFFFFFD.0001 or 8E82AB19_31433C590404010037CA5A00000B7450257A5ACD933.00000000009.FFFFFFFD.0001.

Snapshots

The process of creating a new version of objects instead of replacing them, is what makes cloud snapshots possible. The FXT cluster is able to schedule snapshots for cloud core filers. When it comes time for a snapshot, the system copies the pointers corresponding to that date and time. All the objects needed for a snapshot are protected from deletion.

Object Consistency

Most cloud vendors provide "eventual consistency" for objects written to the cloud. This means that it is possible to write a file, immediately try to read that file, and it may or may not be there and it may or may not contain the changes that were just committed. "Eventual consistency" means that after some amount of time, the objects will reflect what was committed.

For most enterprise customers, "eventual consistency" is not sufficient. A file might be updated from one remote office and another other remote office might not get all of its changes. To accommodate for this consistency, FlashCloud maintains a small database to keep track of the latest version of each object. The FXT's cache will retain the content of each newly-created object while also keeping track of the latest version of each recently-created object. This guarantees that each read request will produce the latest, most accurate version of the requested data.

Compression

The FlashCloud setup process allows administrators the option of using compression for all cloud objects, but is not required. Compression saves space in the cloud but requires memory and CPU cycles.

FlashCloud offers 2 compression options. The default option is to use LZ4 compression which has the best performance ratio between compression/decompression and speed. Administrators may also opt for LZ4HC which offers over 30% more compression but may take 10 times longer to decompress. Compression options may be changed at any time. New objects will use the new compression option. This includes the ability to use no compression.

Encryption

Objects stored in a public cloud need to be kept confidential and their integrity must be protected. Objects are encrypted by the FXT before being sent to cloud storage. FlashCloud protects users data by encrypting it in two states. Data is encrypted as it is transferred over the wire (in-transit encryption) and it is encrypted as it sits in the cloud (at-rest encryption).

In-transit Encryption

Before data leaves the FXT cluster, a secure session is established. The FXT cluster will then transmit its data via HTTPS to secure the transmission and keep the contents of that transmission secret.

At-rest Encryption

One concern of storage administrators is the security of the data where it is stored. Whether on a local disk or in the cloud, most storage administrators prefer to have data stored in an encrypted format.

All cloud objects are written using 256-bit AES (Advanced Encryption Standard) in CBC (Cipher Block Chaining) mode. Each object stored in the cloud is encrypted using a randomly generated Data Encryption Key (DEK). The DEK is encrypted with a Key Encrypting Key (KEK) which is known as Key Wrapping (RFC 3394). The wrapped DEK is stored along with the object. During decryption, the DEK is first unwrapped using the KEK, and then the object payload is decrypted using the DEK.

After encryption, a Hashed Message Authentication Code (HMAC) using SHA-512 is calculated over the object, and the resulting code is appended to the object. For HMAC calculation, a separate randomly generated key is used, which is also wrapped with the KEK and stored with the object.

Key Management

Simple Key Management
Each object in the cloud has a unique DEK and HMAC-Key, both of which are wrapped with the KEK and stored with the object. The KEK can be rotated by generating a new one. Historic and current KEKs are kept in a key database for each cloud core filer.

The key database, also known as the master key, can be downloaded by the cluster administrator, but it must first be encrypted with a user-supplied passphrase. The AES key is used to encrypt the key database. The encrypted key database is downloaded at the time of KEK creation. It is the cluster administrator’s responsibility to securely backup the key database and provide the passphrase for recovery. Per security best practices, it is recommended that keys be rotated on a periodic basis.

KMIP
Some customers prefer to automate key management with an Enterprise Key Manager (EKM). The standard way to communicate with an EKM, is the Key Management Interoperability Protocol (KMIP).

FlashCloud provides KMIP interoperability. Encrypted TLS sessions are established with KMIP so keys may be transmitted securely to the EKM. The EKM then manages all encryption keys without the need to download and save files on a storage administrator’s local machine.