File Gateway for Hybrid Cloud Storage Architectures

Overview and Best Practices for the File Gateway Configuration of the AWS Storage Gateway Service

March 2019

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS's current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS's products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. AWS's responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

Organizations are looking for ways to reduce their physical data center footprints, particularly for storage arrays used as secondary file backup or on-demand workloads. However, providing data services that bridge private data centers and the cloud comes with a unique set of challenges. Traditional data center storage services rely on low-latency network attached storage (NAS) and storage area network (SAN) protocols to access storage locally. Cloud-native applications are generally optimized for API access to data in scalable and durable cloud object storage, such as Amazon Simple Storage Service (Amazon S3). This paper outlines the basic architecture and best practices for building hybrid cloud storage environments using the AWS Storage Gateway in a file gateway configuration to address key use cases, such as cloud tiering, hybrid cloud backup, distribution, and cloud processing of data generated by on-premises applications.

Introduction

Organizations are looking for ways to reduce their physical data center infrastructure. A great way to start is by moving secondary or tertiary workloads, such as long-term file retention and backup and recovery operations, to the cloud. In addition, organizations want to take advantage of the elasticity of cloud architectures and features to access and use their data in new on-demand ways that a traditional data center infrastructure can't support.

AWS Storage Gateway has multiple gateway types, including a file gateway that provides low-latency Network File System (NFS) and Server Message Block (SMB) access to Amazon Simple Storage Service (Amazon S3) objects from on-premises applications. At the same time, customers can access that data from any Amazon S3 API-enabled application. Configuring AWS Storage Gateway as a file gateway enables hybrid cloud storage architectures in use cases such as archiving, on-demand bursting of workloads, and backup to the AWS Cloud.

Individual files that are written to Amazon S3 using the file gateway are stored as independent objects. This provides high durability, low-cost, flexible storage with virtually infinite capacity. Files are stored as objects in Amazon S3 in their original format without any proprietary modification. This means that data is readily available to data analytics and machine learning applications and services that natively integrate with Amazon S3 buckets, such as Amazon EMR, Amazon Athena, or Amazon Transcribe. It also allows for storage management through native Amazon S3 features, such as lifecycle policies, analytics, and cross-region replication (CRR).

A file gateway communicates efficiently between private data centers and AWS. Traditional NAS protocols (SMB and NFS) are translated to object storage API calls. This makes file gateway an ideal component for organizations looking for tiered storage of file or backup data with low-latency local access and durable storage in the cloud.

File Gateway Architecture

A file gateway provides a simple solution for presenting one or more Amazon S3 buckets and their objects as a mountable NFS or SMB file share to one or more clients on-premises.

The file gateway is deployed as a virtual machine in VMware ESXi or Microsoft Hyper-V environments on-premises, or in an Amazon Elastic Compute Cloud (Amazon EC2) instance in AWS. File gateway can also be deployed in data center and remote office locations on a Storage Gateway hardware appliance. When deployed, file gateway provides a seamless connection between on-premises NFS (v3.0 or v4.1) or SMB (v1 or v2) clients—typically applications—and Amazon S3 buckets hosted in a given AWS Region. The file gateway employs a local read/write cache to provide a low-latency

access to data for file share clients in the same local area network (LAN) as the file gateway.

A *bucket share* consists of a file share hosted from a file gateway across a single Amazon S3 bucket. The file gateway virtual machine appliance currently supports up to 10 bucket shares.

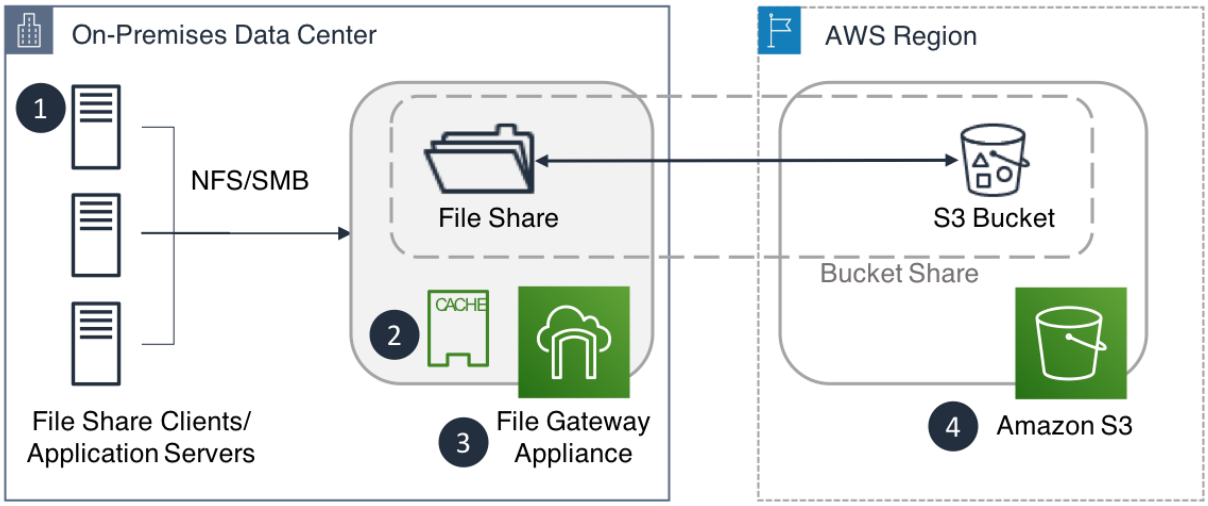


Figure 1: Basic file gateway architecture

Here are the components of the file gateway architecture shown in Figure 1:

1. Clients, access objects as files using an NFS or SMB file share exported through an AWS Storage Gateway in the file gateway configuration
2. Expandable read/write cache for the file gateway
3. File gateway virtual appliance
4. Amazon S3, which provides persistent object storage for all files that are written using the file gateway

File to Object Mapping

After deploying, activating, and configuring the file gateway, one or more bucket shares can be presented to clients that support NFS v3 or v4.1 protocols, or mapped to a share via SMB v1 or v2 protocols on the local LAN. Each share (or mount point) on the gateway is paired to a single bucket, and the contents of the bucket are available as files and folders in the share.

Writing an individual file to a share on the file gateway creates an identically named object in the associated bucket. All newly created objects are written to Amazon S3 Standard, Amazon S3 Standard – Infrequent Access (S3 Standard – IA), or Amazon S3

One Zone – Infrequent Access (S3 One Zone – IA) storage classes, depending on the configuration of the share.

The Amazon S3 key name of a newly created object is identical to the full path of the file that is written to the mount point in AWS Storage Gateway.

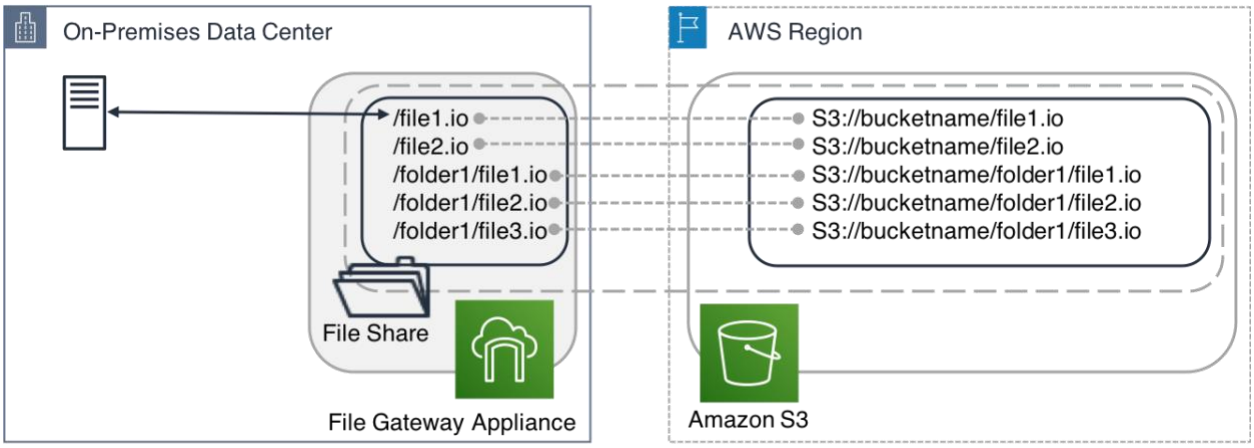


Figure 2: Files stored over NFS on the file gateway mapping to Amazon S3 objects

One difference between storing data in Amazon S3 versus a traditional file system is the way in which granular permissions and metadata are implemented and stored. Access to files stored directly in Amazon S3 is secured by policies stored in Amazon S3 and AWS Identity and Access Management (IAM). All other attributes, such as storage class and creation date, are stored in a given object's metadata. When a file is accessed over NFS or SMB, the file permissions, folder permissions, and attributes are stored in the file system.

To reliably persist file permissions and attributes, the file gateway stores this information as part of Amazon S3 object metadata. If the permissions are changed on a file over NFS or SMB, the gateway modifies the metadata of the associated objects that are stored in Amazon S3 to reflect the changes. Custom default UNIX permissions are defined for all existing S3 objects within a bucket when a share is created from the AWS Management Console or using the file gateway API. This feature lets you create NFS or SMB enabled shares from buckets with existing content without having to manually assign permissions after you create the share.

The following is an example of a file that is stored in a share bucket and is listed from a Linux-based client that is mounting the share bucket over NFS. The example shows that the file "file1.txt" has a modification date and standard UNIX file permissions.

```
[e2-user@host]$ ls -l /media/filegateway1/
total 1
-rw-rw-r-- 1 ec2-user ec2-user 36 Mar 15 22:49 file1.txt
[e2-user@host]$
```

The following example shows the output from the head-object on Amazon S3. It shows the same file from the perspective of the object that is stored in Amazon S3. Note that the permissions and time stamp in the previous example are stored durably as metadata for the object.

```
[e2-user@host]$ aws s3api head-object --bucket filegateway1 --key
file1.txt
{
    "AcceptRanges": "bytes",
    "ContentType": "application/octet-stream",
    "LastModified": "Wed, 15 Mar 2017 22:49:02 GMT",
    "ContentLength": 36,
    "VersionId": "93XCzHcBUHBSg2yP.8yKMHzxUumhovEC",
    "ETag": "\"0a7fb5dbb1ae1f6a13c6b4e4dcf54977-1\"",
    "ServerSideEncryption": "AES256",
    "Metadata": {
        "file-group": "500",
        "user-agent-id": "sgw-7619FB1F",
        "file-owner": "500",
        "aws-sgw":
"57c3c3e92a7781f868cb10020b33aa6b2859d58c868190661bcceae87f7b96f1",
        "file-mtime": "1489618141421",
        "file-ctime": "1489618141421",
        "user-agent": "aws-storage-gateway",
        "file-permissions": "0664"
    }
}
[e2-user@host]$
```

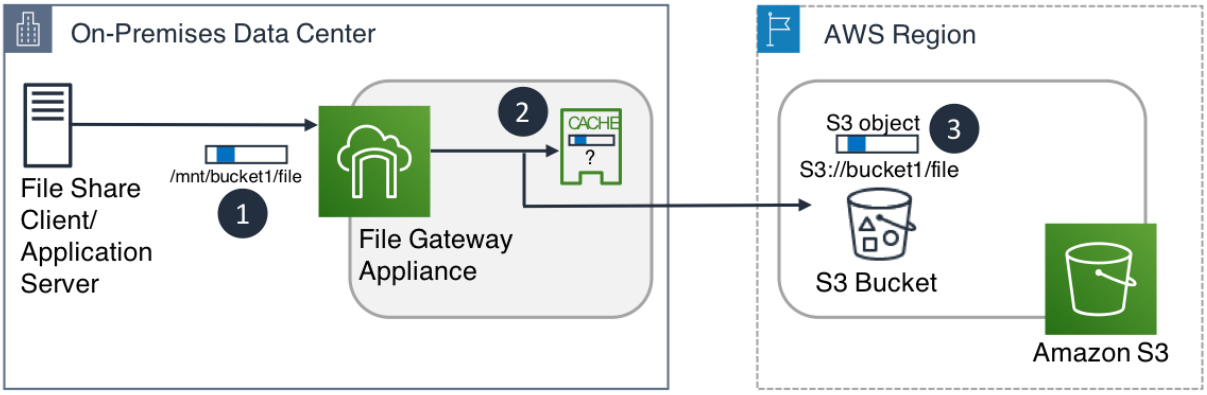
Read/Write Operations and Local Cache

As part of a file gateway deployment, dedicated local storage is allocated to provide a read/write cache for all hosted share buckets. The read/write cache greatly improves response times for on-premises file (NFS/SMB) operations. The local cache holds both recently written and recently read content and does not proactively evict data while the cache disk has free space. However, when the cache is full, AWS Storage Gateway evicts data based on a least recently used (LRU) algorithm. Recently accessed data is available for reads, and write operations are not impeded.

Read Operations (Read-Through Cache)

When an NFS client performs a read request, the file gateway first checks the local cache for the requested data. If the data is not in the cache, the gateway retrieves the

data from Amazon S3 using Range `GET` requests to minimize data transferred over the Internet while repopulating the read cache on behalf of the client.



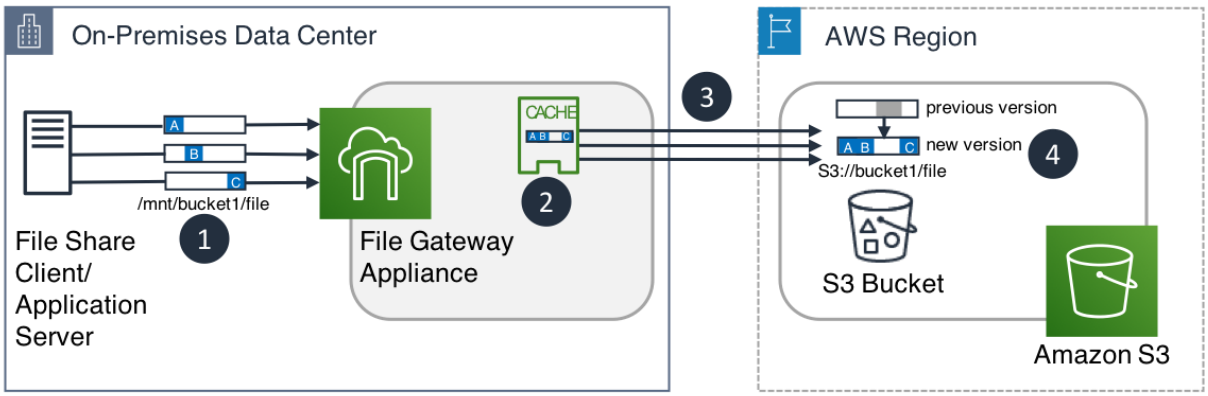
1. The NFS/SMB client performs a read request on part of a given file.
2. The file gateway first checks to see if required bytes are cached locally.
3. In the event the bytes are not in the local cache, the file gateway performs a byte range `GET` on the associated S3 object.

Figure 3: File gateway read operations

Write Operations (Write-Back Cache)

When a file is written to the file gateway over NFS/SMB, the gateway first commits the write to the local cache. At this point, the write success is acknowledged to the local NFS/SMB client, taking full advantage of the low latency of the local area network. After the write cache is populated, the file is transferred to the associated Amazon S3 bucket asynchronously to increase local performance of Internet transfers.

When an existing file is modified, the file gateway transfers only the newly written bytes to the associated Amazon S3 bucket. This uses Amazon S3 API calls to construct a new object from a previous version in combination with the newly uploaded bytes. This reduces the amount of data required to be transferred when clients modify existing files within the file gateway.



1. File share client performs many parallel writes to a given file.
2. File gateway appliance acknowledges writes synchronously, aggregates writes locally.
3. File gateway appliance uses S3 multi-part upload to send new writes (bytes) to S3.
4. New object is constructed in S3 from a combination of new uploads and byte ranges from the previous version of an object.

Figure 4: File gateway write operations

Choosing the Right Cache Resources

When configuring a file gateway VM on a host machine, you can allocate disks for the local cache. Selecting a cache size that can sufficiently hold the active working set (e.g. a Database backup file) provides optimal performance for file share clients. Additionally, splitting the cache across multiple disks maximizes throughput by parallelizing access to storage, resulting in faster reads and writes. When available for your on-premises gateway, we also recommend using SSD or ephemeral disks, which can provide write and read (cache hits) throughputs of up to 500MB/s.

Security and Access Controls Within a Local Area Network

When you create a mount point (share) on a deployed gateway, you select a single Amazon S3 bucket to be the persistent object storage for files and associated metadata. Default UNIX permissions are defined as part of the configuration of the mount point. These permissions are applied to all existing objects in the Amazon S3 bucket. This process ensures that clients that access the mount point adhere to file and directory-level security for existing content.

In addition, an entire mount point and its associated Amazon S3 content can be protected on the LAN by limiting mount access to individual hosts or a range of hosts.

For NFS file shares, this limitation is defined by using a Classless Inter-Domain Routing (CIDR) block or individual IP addresses. For SMB file shares, you can control access using Active Directory (AD) domains, or authenticated guest access. You can further limit access to selected AD users and groups, allowing only specified users (or users in the specified groups) to map the file share as a drive on their Microsoft Windows machines.

Monitoring Cache and Traffic

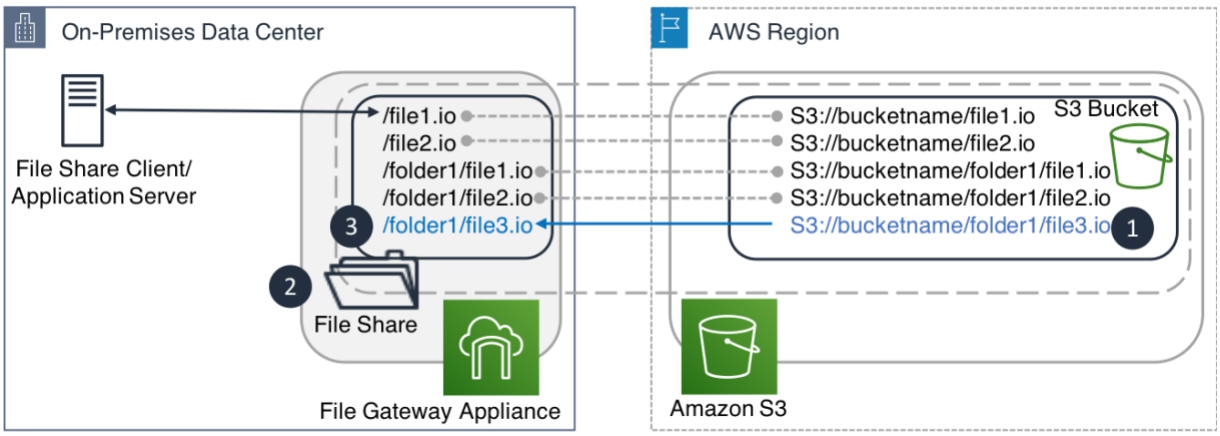
As workloads or architectures evolve, the cache and Internet requirements that are associated with a given file gateway deployment can change over time. To give visibility into resource use, the file gateway provides statistical information in the form of Amazon CloudWatch metrics. The metrics cover cache consumption, cache hits/misses, data transfer, and read/write metrics. For more information, see Monitoring Your File Share.

File Gateway Bucket Inventory

To reduce both latency and the number of Amazon S3 operations when performing list operations, the file gateway stores a local bucket inventory that contains a record of all recently listed objects. The bucket inventory is populated on-demand as the file share clients list parts of the file share for the first time. The file gateway updates inventory records only when the gateway itself modifies, deletes, or creates new objects on behalf of the clients. The file gateway cannot detect changes to objects in an NFS or SMB file share's bucket by a secondary gateway that is associated with the same Amazon S3 bucket or by any other Amazon S3 API call outside of the file gateway.

When Amazon S3 objects have to be modified outside of the file share and recognized by the file gateway (such as changes made by Amazon EMR or other AWS services), the bucket inventory must be refreshed using either the RefreshCache API call or `RefreshCache` AWS Command Line Interface (CLI) command. `RefreshCache` can be manually invoked, automated using a CloudWatch Event, or triggered through the use of the NotifyWhenUploaded API call once the files have been written to the file share using a secondary gateway. A CloudWatch notification named `Storage Gateway Upload Notification Event` is triggered once the files written by the secondary gateway have been uploaded to S3. The target of this event could be a Lambda function invoking `RefreshCache` to inform the primary gateway of this change.

`RefreshCache` re-inventories the existing records in a file gateway's bucket inventory. This communicates changes of known objects to the file share clients that access a given share.



1. Object created by secondary gateway or external source.
2. RefreshCache API called on file gateway appliance share.
3. Foreign object is reflected in file gateway bucket inventory and accessible by clients.

Figure 5: RefreshCache API called to re-inventory Amazon S3 bucket

Bucket Shares with Multiple Contributors

When deploying more complex architectures, such as when more than one file gateway share is associated with a single Amazon S3 bucket, or in scenarios where a single bucket is modified by one or more file gateways in conjunction with other Amazon S3-enabled applications, note that file gateway does not support object locking or file coherency across file gateways.

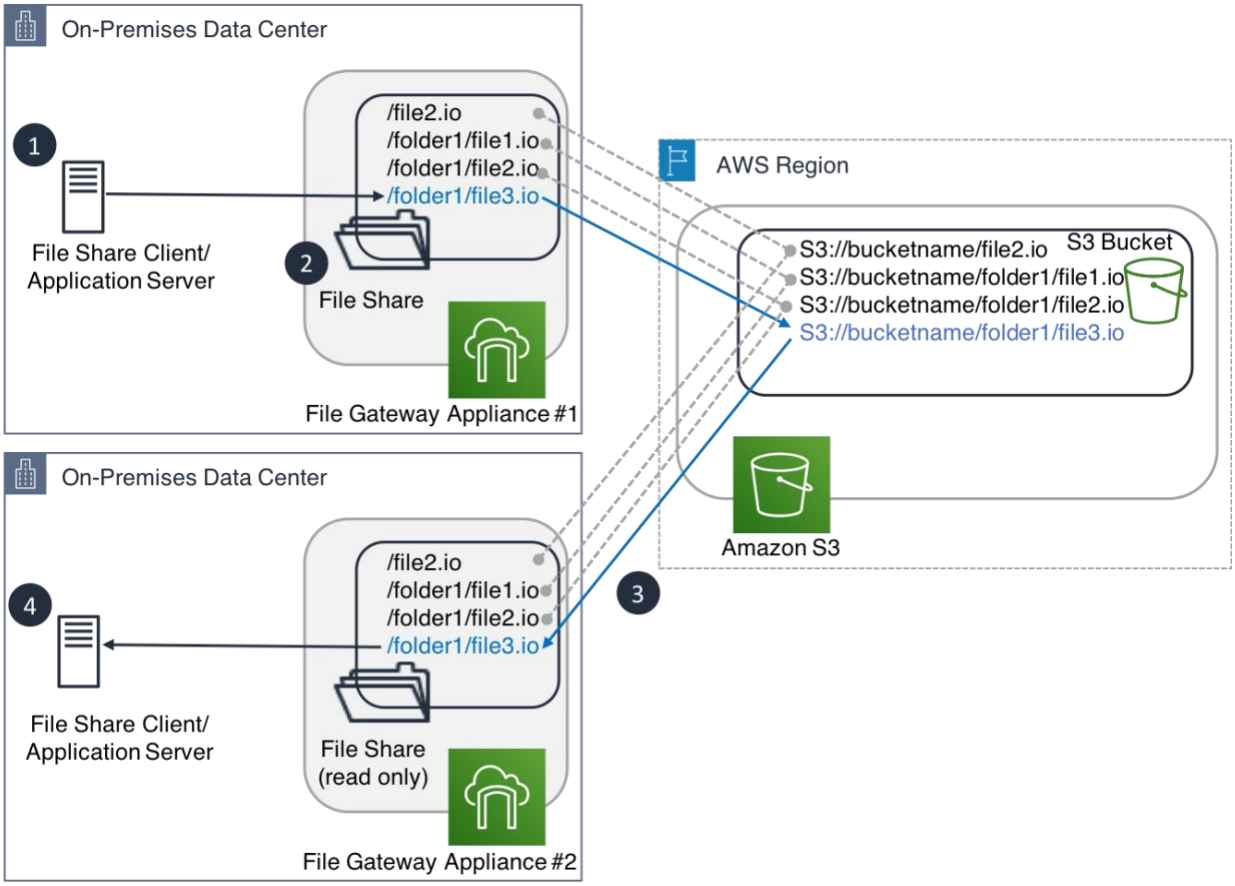
Since file gateways cannot detect other file gateways, be cautious when designing and deploying solutions that use more than one file gateway share with the same Amazon S3 bucket. File gateways associated with the same Amazon S3 bucket detect new changes to the content in the bucket only in the following circumstances:

1. A file gateway recognizes changes it makes to the associated Amazon S3 bucket and can notify other gateways and applications by invoking the NotifyWhenUploaded API after it is done writing files to the share.
2. A file gateway recognizes changes made to objects by other file gateways when the affected objects are located in folders (or prefixes) that have not been queried by that particular file gateway.
3. A file gateway recognizes changes in an associated Amazon S3 bucket (bucket share) made by other contributors after the RefreshCache API is executed.

We recommend that you use the read-only mount option on a file gateway share when you deploy multiple gateways that have a common Amazon S3 bucket. Designing architectures with only one writer and many readers is the simplest way to avoid write conflicts. If multiple writers are required, the clients accessing each gateway must be

tightly controlled to ensure that they don't write to the same objects in the shared Amazon S3 bucket.

When multiple file gateways are accessing the same objects in the same Amazon S3 bucket, make sure to call the RefreshCache API on file gateway shares that have to recognize changes made by other file gateways. To further optimize this operation and reduce the time it takes to run, you can invoke the RefreshCache API on specific folders (recursively or not) in your share.



1. Client creates a new file and file gateway #1 uploads object to S3.
2. Customer invokes NotifyWhenUploaded API on file share of file gateway #1.
3. CloudWatch Event (generated upon completion of Step 1) initiates the RefreshCache API call to initiate a re-inventory on file gateway #2.
4. File gateway #2 presents newly created objects to clients.

Figure 6: RefreshCache API makes objects created by file gateway #1 visible to file gateway #2

Amazon S3 and the File Gateway

The file gateway uses Amazon S3 buckets to provide storage for each mount point (share) that is created on an individual gateway. When you use Amazon S3 buckets, mount points provide limitless capacity, 99.999999999% durability on objects stored, and a consumption-based pricing model.

Costs for data stored in Amazon S3 via AWS Storage Gateway are based on the region where the gateway is located and the storage class. A given mount point writes data directly to Amazon S3 Standard, Amazon S3 Standard – IA, or Amazon S3 One Zone – IA storage, depending on the initial configuration selected when creating the mount point. All of these storage classes provide equal durability. However, Amazon S3 Standard – IA and Amazon S3 One Zone – IA have a different pricing model and lower availability (i.e., 99.9% compared with 99.99%), which makes them good solutions for less frequently accessed objects. The pricing for Amazon S3 Standard – IA and Amazon S3 One Zone – IA is ideal for objects that exist for more than 30 days and are larger than 128 KB per object.

For details about price differences for Amazon S3 storage classes, see the Amazon S3 Pricing page.

Using Amazon S3 Object Lifecycle Management for Cost Optimization

Amazon S3 offers many storage classes. Today, AWS Storage Gateway file gateway supports S3 Standard, S3 Standard – Infrequent Access, and S3 One Zone – IA natively. Amazon S3 lifecycle policies automate the management of data across storage tiers. It's also possible to expire objects based on the object's age.

To transition data between storage classes, lifecycle policies are applied to an entire Amazon S3 bucket, which reflects a single mount point on a storage gateway. Lifecycle policies can also be applied to a specific prefix that reflects a folder within a hosted mount point on a file gateway. The lifecycle policy transition condition is based on the creation date or, optionally, on the object tag key value pair. For more information about tagging, see Object Tagging in the *Amazon S3 Developer Guide*.

As an example, a lifecycle policy in its simplest implementation moves all objects in a given Amazon S3 bucket from Amazon S3 Standard to Amazon S3 Standard – IA, and finally to Amazon S3 Glacier as the data ages. This means that files created by the file gateway are stored as objects in Amazon S3 buckets and can then be automatically transitioned to more economical storage classes as the content ages.

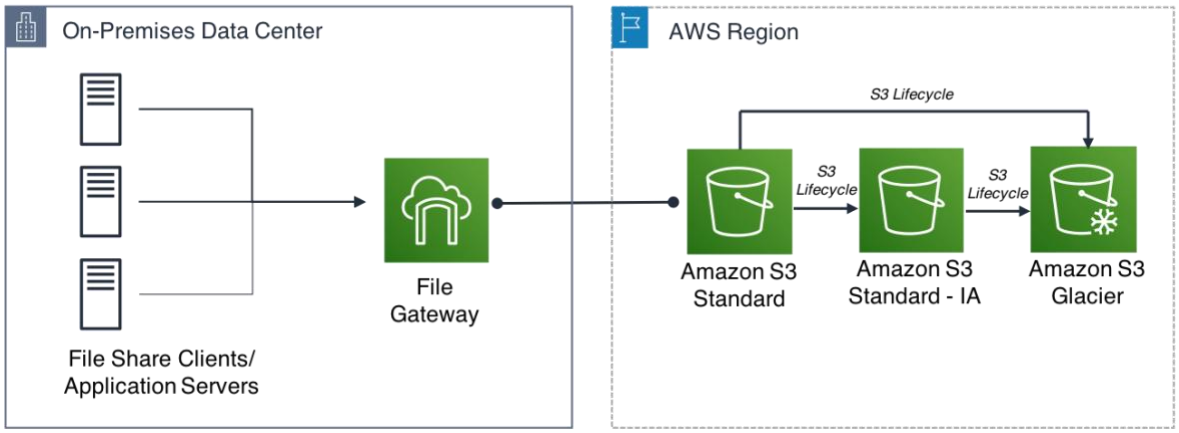


Figure 7: Example of file gateway storing files as objects in Amazon S3 Standard and transitioning to Amazon S3 Standard – IA and Amazon S3 Glacier

If you use file gateway to store data in S3 Standard-IA or S3 One Zone-IA or access data from any of the infrequent storage classes, see Using Storage Classes in the *AWS Storage Gateway User Guide* to learn how the gateway mediates between NFS/SMB (file based) uploads to update or access the object.

Transitioning Objects to Amazon S3 Glacier

Files migrated using lifecycle policies are immediately available for NFS file read/write operations. Objects transitioned to Amazon S3 Glacier are visible when NFS files are listed on the file gateway. However, they are not readable unless restored to an S3 storage class using an API or the Amazon S3 console.

If you try to read files that are stored as objects in Amazon S3 Glacier, you encounter a read I/O error on the client that tries the read operation. For this reason, we recommend using lifecycle to transition files to Amazon S3 Glacier objects only for file content that does not require immediate access from an NFS/SMB client in an AWS Storage Gateway environment.

Amazon S3 Object Replication Across AWS Regions

Amazon S3 cross-region replication (CRR) can be combined with a file gateway architecture to store objects in two Amazon S3 buckets across two separate AWS Regions. CRR is used for a variety of use cases, such as protection against human error, protection against malicious destruction, or to minimize latency to clients in a remote AWS Region. Adding CRR to the file gateway architecture is just one example of how native Amazon S3 tools and features can be used in conjunction with the file gateway.

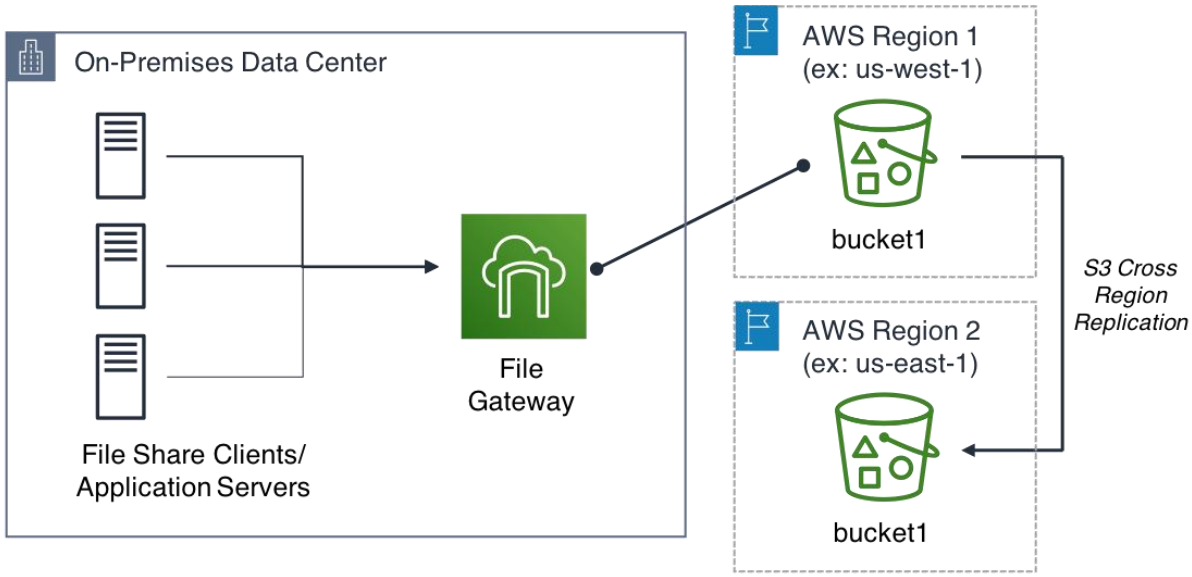


Figure 8: File gateway in a private data center with CRR to duplicate objects across AWS Regions

Using Amazon S3 Object Versioning

You can use file gateway with Amazon S3 Object Versioning to store multiple versions of files as they are modified. If you require access to a previous version of the object using the gateway, it first must be restored to the previous version in S3. You must also use the RefreshCache operation for the gateway to be notified of this restore. See Object Versioning Might Affect What You See in Your File System in the *AWS Storage Gateway User Guide* to learn more about using Amazon S3 versioned buckets for your file share.

Using the File Gateway for Write Once Read Many (WORM) Data

You can also use file gateway to store and access data in environments with regulatory requirements that require use of WORM storage. In this case, select a bucket with S3 Object Lock enabled as the storage for the file share. If there are file modifications or renames through the file share clients, the file gateway creates a new version of the object without affecting prior versions so the original locked version remains unchanged. See also Using the file gateway with Amazon S3 Object Lock in the *AWS Storage Gateway User Guide*.

File Gateway Use Cases

The following scenarios demonstrate how a file gateway can be used in both cloud tiering and backup architectures.

Cloud Tiering

In on-premises environments where storage resources are reaching capacity, migrating colder data to the file gateway can extend the life span of existing storage on-premises and reduce the need to use capital expenditures on additional storage hardware and data center resources. When adding the file gateway to an existing storage environment, on-premises applications can take advantage of Amazon S3 storage durability, consumption-based pricing, and virtual infinite scale, while ensuring low-latency access to recently accessed data over NFS or SMB.

Data can be tiered using either native host OS tools or third-party tools that integrate with standard file protocols such as NFS or SMB.

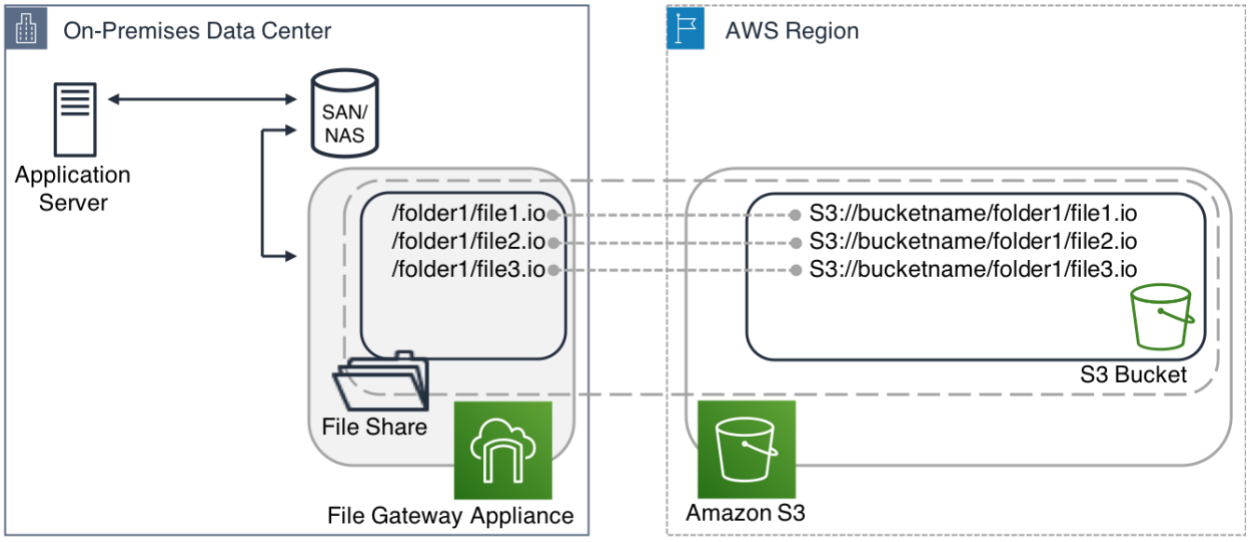


Figure 9: File gateway in a private data center providing Amazon S3 Standard or Amazon S3 Standard – IA as a complement to existing storage deployments

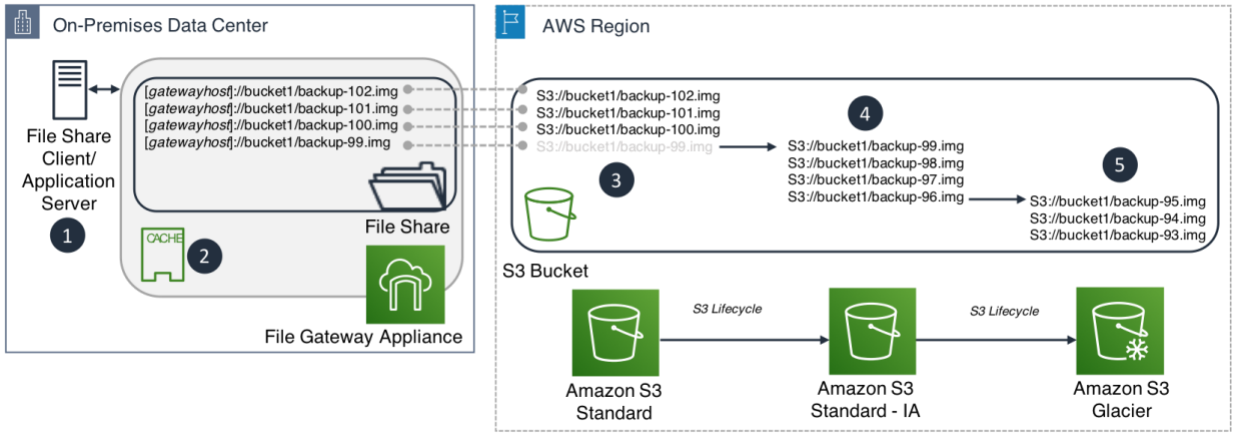
Hybrid Cloud Backup

The file gateway provides a low-latency NFS/SMB interface that creates Amazon S3 objects of up to 5 TiB in size, stored in a supported AWS Region. This makes it an ideal hybrid target for backup solutions that can use NFS or SMB. By using a mixture of Amazon S3 storage classes, data is stored on low-cost, highly durable cloud storage and automatically tiered to progressively lower-cost storage as the likelihood of restoration diminishes. Figure 10 shows an example architecture that assumes backups must retained for one year. After 30 days, the likelihood of restoration becomes infrequent, and after 60 days it becomes extremely rare.

In this solution, you use Amazon S3 Standard as the initial location for backups for the first 30 days. The backup software or scripts write backups to the file share, preferably in the form of multi-megabyte or larger size files. Larger files offer better cost

optimization in the end-to-end solution, including colder storage costs, and lifecycle transition costs because fewer transitions are required.

After another 30 days, the backups are transitioned to Amazon S3 Glacier. Here, they are held until a full year has passed since they were first created, at which point they are deleted.



1. Client writes backups to file gateway over NFS or SMB.
2. File gateway cache, sized greater than expected backup.
3. Initial backups stored in S3 Standard.
4. Backups are transitioned to S3 Standard-IA after 30 days.
5. Backups are transitioned to S3 Glacier after 60 days.

Figure 10: Example of file gateway storing files as objects in Amazon S3 Standard and transitioning to Amazon S3 Standard - IA and Amazon S3 Glacier

When sizing the file gateway cache in this type of solution, understand the backup process itself. One approach is to size the cache to be large enough to contain a complete full backup, which allows restores from that backup to come directly from the cache—much more quickly than over a wide-area network (WAN) link.

If the backup solution uses software that consolidates backup files by reading existing backups before writing ongoing backups, factor this configuration into the sizing of cache also. This is because reading from the local cache during these types of operations reduces cost and increases overall performance of ongoing backup operations.

For both cases specified above, you can use AWS DataSync to transfer data to the cloud from an on-premises data store. From there, the access to the data can be retained using a file gateway.

Conclusion

The file gateway configuration of AWS Storage Gateway provides a simple way to bridge data between private data centers and Amazon S3 storage. The file gateway can enable hybrid architectures for cloud migration, cloud tiering, and hybrid cloud backup.

The file gateway's ability to provide a translation layer between the standard file storage protocols and Amazon S3 APIs without obfuscation makes it ideal for architectures in which data must remain in its native format and be available both on-premises and in the AWS Cloud.

For more information about the AWS Storage Gateway service, see AWS Storage Gateway.

Contributors

The following individuals and organizations contributed to this document:

- Peter Levett, Solutions Architect, AWS
- David Green, Solutions Architect, AWS
- Smitha Sriram, Senior Product Manager, AWS
- Chris Rogers, Business Development Manager, AWS

Further Reading

For additional information, see the following:

- AWS Storage Services Overview Whitepaper
- AWS Whitepapers Web page
- AWS Storage Gateway Documentation
- AWS Documentation Web page

Document Revisions

Date	Description
March 2019	Updated for S3 One Zone-IA storage class.

Date	Description
April 2017	Initial document creation