Content Delivery for Games

Optimizing the delivery of game assets to players

November 2020

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 current AWS 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 products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS 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

Most games whether played on mobile, PC, or a console device require various assets such as textures, images, models, video, and audio delivered to the game client to enable game play. This whitepaper describes the requirements, design choices, and provides some best practices and reference architectures for implementing, operating, and delivering assets for games using one or more Content Delivery Networks (CDN).

Introduction

Today, gamers want to play their favorite game anytime, anywhere, and on any device. With rising popularity of free-to-play (F2P) games, players often switch games if they have played through the content or if the game is not engaging enough. Due to this high propensity to player churn, game developers and publishers are forced to keep up with a constant demand for fresh content as often as possible, in order to retain more players, and monetize effectively to sustain their business.

Depending on the type of game, and target device platform: console, mobile, or PC, content assets for games may vary and include a diverse mix of textures, images, models, videos, audios, and more. The size of these assets can also vary from hundreds of megabytes for mobile games to tens of gigabytes for PC or console games. Due to these reasons, CDNs are often used to efficiently cache such assets at the edge close to the players and deliver these as needed.

In this paper, we will cover the various requirements for setting up CDN for games to deliver game assets, and some typical challenges and design choices to consider. We will outline the following aspects of such a solution:

1. Efficient assets layout in the origin to optimize cost and performance.
2. Content updates and serving multiple assets versions to clients.
3. Uploading content and integration with build pipelines.
4. Spreading load between multiple CDNs to optimize performance, cost, and improve availability.
5. Collecting download performance metrics from clients and CDNs, real-time performance monitoring, and batch reporting.

Use cases

The primary use case for CDN in games is caching and serving static assets of various types and sizes efficiently to game clients running on customer devices. However, there are other use cases such as media streaming, dynamic content acceleration, ad serving, and much more which are out of scope of this document.

Requirements

The most important requirements for static game assets distribution include the following:

- **Sufficient download capacity to handle workload spikes**: CDN capacity directly influences game success because player experience includes pre- and post-installation downloads. An ineffective content delivery strategy could lead to unsuccessful downloads and churn before players even start to play the game.
- **Cost efficiency**: For popular games, the content delivery can make up a significant part of IT infrastructure spend. For this reason, it is important to develop a cost-efficient architecture to improve game economics.
- **Consistent download performance**: Players may originate from around the world and will expect a similar experience regardless of where they live or how far they are from the origin of the download assets.
- **Compliance with data localization requirements**: It is important to consider how your game download strategy will address localized data compliance regulations in specific parts of the world.
- **Content versioning**: Ability to serve different versions of the assets at the same time to support older clients or for A/B testing.

For many of the reasons stated previously, most game distribution platforms like Steam, PlayStation Network, Xbox Live, and mobile stores for iOS and Android distribute content through their own CDN solutions. However, many game publishers still rely on external CDNs because unified solutions that work across various games distribution platforms simplifies content management and provide more control to the publisher.

Design choices

When it comes to implementing a content distribution solution, there are several design considerations which we will cover in this section along with the possible options and recommendations.

Bundle assets with a game or serve separately

Mobile game developers often publish a relatively small game in application stores and expect the game to download all required assets from a CDN after the game first launches. For example, a typical mobile game size in an app store can be around 100 MB, but the game may download several hundred megabytes of additional assets

afterwards. This approach can reduce the waiting time for players and thus reduces the chance of player churn before they even start to play. Players may also be more likely to download games from application stores if they are smaller in size.

PC and console games often use a similar approach and it is also common for PC games to have a chain of installers. For example, a player may first download an installer or a *download center* application, and then the installer will download and install a game. Once the game starts, it may then download required assets as needed.

It is generally a good idea to keep the size of the first application that players download reasonably low, and download remaining assets later in background where possible.

Content layout in the origin

Game developers usually serve the following types of content:

- **Game assets**: Format may vary, usually some archive. This is by far the largest part of download volume.
- **Manifest files** (usually JSON): Provides a list of content to download with URLs for the game client to use.
- **Configuration files**: Optional files which may define game settings. Files might need to be updated independently from a game application or asset revisions and are used to configure features in the game or other runtime configurations.

Instead of serving every asset independently, game developers usually pack assets into compressed resource packs to improve download performance. Additionally, you can split these archives into chunks and re-assemble chunks back on the client. You need to implement logic to break assets into pieces and reassemble them back on the client in this case.

It is important to choose an optimal resource pack or chunk size. Larger packs or chunks improve bandwidth. However, small chunks provide their own benefits which includes providing more efficient partial content updates because there is less data to re-upload and re-download. This also works well with rsync-like protocols and enables resuming downloads easier. You can also download several smaller chunks in parallel to improve performance.

A good starting point is to pack necessary assets into multiple archives, and target 10-100 megabytes file size for optimal balance between download performance and manageability. However, an important exception to this approach is browser or Facebook Messenger games. These platforms cannot download a big content pack in advance because there is no way to store it on the device. Instead, players effectively

play the game nearly entirely from CDN. In this case, small separate files should be served to client and compression on CDN can improve performance in this case.

Updates and versioning

Game developers regularly release new game and content versions and it is important to identify a strategy for updating assets that are being served through CDN before their TTL expires. You can either upload a new content pack with the same name and invalidate it from CDN caches, or you can store every new asset version under a unique filename which is usually recommended.

Uploading new assets with the same file names and invalidating caches has the following disadvantages:

- CDN providers usually charge extra for cache invalidations.
- It is hard to roll forward and back between different asset versions.
- It is harder to analyze CDN access logs when all versions share the same filename.
- It is impossible to serve multiple asset versions at the same time. This is almost always required for games, for example, to provide corresponding asset versions to outdated clients.

These reasons are common examples why it is usually recommended to store all assets versions in the origin under different filenames and maintain version history. You may include a version identifier either in your file name, use separate directories, or implement a separate mapping service to track assets version and filenames.

> **Note:** There might be some cases where CDN cache invalidation makes sense. For example, if you are storing game configuration values in files stored in a content delivery network, then it may be important that updates are deployed as a global game behavior change. In this case make sure you selectively invalidate only required files, and don’t use wildcards when not necessary.

Download flow

In the use case of a download flow, game clients need to retrieve a list of assets to download along with their URLs before they can begin downloading the actual assets. One approach for this is to operate a dedicated *dispatcher* service through an API or from a simple manifest file that can be downloaded from a well-known location.

A dispatcher service is usually recommended over a manifest file for the following reasons:

- You can dynamically change CDN behavior without implementing changes to the client. For example, you can adjust traffic split between multiple CDN providers as a hot configurable change that is detected nearly immediately to influence traffic management behavior.
- You can tailor a response based on the information provided by a client, such as, version, client ID, IP address, or geo-location attributes such as the internet service provider (ISP) and Autonomous System Number (ASN) of the client to route traffic for specific networks to a pre-configured set of CDN providers. This also enables advanced functionality, such as serving localized content to players from different geo locations or A/B testing (serving modified assets to a small percentage of users).

When using multiple CDN providers, it is important to implement some failover or CDN selection logic on the client. The client can either receive a list of multiple URLs for every asset (primary and backup ones), or receive URLs for primary locations only and can request backup ones, if needed. Using the first approach means slightly more metadata traffic but is more flexible and performant for the client.

It is usually more difficult and time-consuming to update clients than backend services, so it is recommended not to implement overly complex failover or load-balancing scenarios in the client. Instead, it is recommended to keep the client simple, and implement most traffic management logic on the server side (backend).

Single or multiple CDNs

Consider using multiple CDNs simultaneously for the following reasons:

- **High availability**: Games need to operate even if a certain CDN experiences performance issues.
- **Handling traffic spikes**: Multiple CDN setups allow *spill over* traffic to secondary CDNs if the primary one cannot handle workload spikes well.
- **Compliance with data localization requirements**: In certain parts of the world you may be able to address regulatory requirements more effectively than with other CDN providers, for example, in China.
- **Improve performance for users in specific regions**: Different CDNs can have different geographical coverage or ISP peering connectivity that can more effectively serve your players.

- **Cost considerations**: Certain CDN providers may be able to provide more cost-effective rates than other providers in certain geographies.

For the reasons above, it is recommended to setup at least one backup CDN for high availability reasons. Consider setting up multiple if there is a specific business need or other considerations need to be addressed.

Traffic management

There are different ways to route traffic between different CDN providers. You can implement either a simple failover mechanism or a more sophisticated solution with weighted distribution and advanced logic. A game client can failover to a backup CDN only if a download from the primary one failed. Alternatively, you can setup weights for every CDN and decide which one to use based on those along with geo location and other settings.

The balancing logic can be implemented in a game client or in a dispatcher service. In the previous case the game client receives a full list of URLs for every content pack it needs, and decides which ones to use. In the latter case, the dispatcher service selects a preferred URL for the client and vends it to the client.

It is usually recommended to implement traffic management and failover logic in a dispatcher service for the following reasons:

- If changes to the logic is required, it is easier to update a dispatcher service transparently to the client than to push an update down to the client. Changes to a client must be aligned to its release cycles, and, more importantly, there is usually no way to force players to update their applications in a non-disruptive way. It might take weeks or months until all players install the updated version.
- Ensuring that all clients have similar code in every game can be redundant and requires extra effort.
- It can be difficult to coordinate real-time changes in traffic split between CDNs based on metrics within hundreds of thousands or even millions of clients. It is far easier to orchestrate this server-side.

Normally customers try to route most of their traffic through their primary CDN to optimize contracts and get better pricing terms due to usage volume, and leverage other CDNs only when needed. For example, players from specific geographic locations may experience better performance from specific CDN providers, or you a CDN provider may experience issues that are outside of your control to remediate.

Traffic distribution logic and CDN weights can be adjusted either manually, after a regular review, or automatically in near real-time. Manual decisions can be based on historical usage trends, billing and performance reports for different CDNs, or expert knowledge about upcoming events. You can also integrate your CDN monitoring solution with a dispatcher service to automatically change CDN behavior in a near-real time.

Recommendations:

- Implement central point of control over traffic split in a centralized dispatcher service.
- Setup CDN logs and metrics collection to understand usage and performance trends.
- Review usage and performance data regularly to decide if any changes are needed. Consider automatic solution if there is a need for more timely reaction.

Uploading content

If you use multiple CDNs for high availability, you can also consider storing content in separate origins. Amazon Simple Storage Service (Amazon S3) is designed for 99.99% availability (with S3 Standard storage class), automatically replicates data across 3 AZs, and is independent from CDN, so it can be used as a single origin for multiple CDNs. However, some customers implement different origins for different CDNs, and upload content to them in parallel for reasons such as increased high availability or cost reduction. For example, Amazon CloudFront does not charge origin fetch costs when an AWS origin is used such as Amazon S3, which provides a cost benefit for using S3 as an origin for CloudFront Distributions. However, other CDN providers may offer similar cost reductions for storing origin content within their cloud storage solutions.

Typically, customers implement a proxy service that provides a common upload interface content development teams and abstracts specific CDN implementation details from the user. The proxy service is responsible for uploading content to different origins and validating uploads success. Optionally, the proxy service can provide new URLs to a *dispatcher* service and usually provides APIs to integrate with CI/CD pipelines. Depending on business requirements, an upload service may also provide a user interface (UI) to manually upload content.

Logging and reporting

When implementing a CDN solution for your game, it is important that you have a mechanism for regularly analyzing usage behavior and performance so that you can

continually improve the player experience and reduce costs where appropriate. Reports about CDN performance, usage and price are important for the following reasons:

- Understanding historical usage and performance data to make decisions about usage commitments and traffic split between different CDN providers
- Understanding CDN cost per game to identify opportunities for cost reduction
- Identify issues with game content distribution and troubleshoot as needed

Data sources for reporting can include CDN access logs, CDN metrics and service logs, and customer data from game clients. It is recommended to collect this data from all your CDN providers in a single location such as an Amazon S3 bucket, and build a unified reporting solution. Many CDN providers (including Amazon CloudFront) can deliver access logs to S3. For an example implementation of a reporting solution, see Monitoring and Reporting.

You can also implement a custom client-side telemetry collection SDK to collect download metrics from your game clients. Client-side metrics are often used in addition to CDN logs because access logs provided by most CDNs don't reflect a full download experience; they only show the experience as perceived by the CDN. For example, CDN access logs do not provide full information about overall download experience for a specific client (including retries after unsuccessful attempts) and about failed downloads because they only represent the state of the request from the perspective of the CDN server. Standard metrics collected from clients include download URLs, download time, download status (successful/unsuccessful), and the number of retries. You can choose to send all events from clients or sample them to optimize cost and bandwidth.

Monitoring

In addition to a logging and reporting solution you can build a near-real-time monitoring solution that provides the following capabilities:

- Early identification of potential churn because of unsatisfactory download experience.
- Uncovering availability or performance issues to timely react on them.
- Early detection of anomalies like unexpected workload spikes, unusual users' distribution between geo locations, or costs increase.
- Automated adjustment of traffic distribution and *dispatcher* service behavior based on metrics.

Data sources to analyze in near real-time include metrics from your game clients, Amazon CloudFront real-time access logs, and CDN metrics. Client events and CloudFront real-time access logs can be ingested into Amazon Kinesis Data Streams and analyzed in real-time with Amazon Kinesis Data Analytics. The following reference architecture for a monitoring and reporting solution shows an example implementation.

Example Reference Architectures

This section shows example architectures that can be used to help customers perform the following actions:

- Analyze CDN performance and efficiency
- Deliver content to players through multiple CDNs
- Upload new versions of game assets to CDN origins

Monitoring and reporting solution for CDN

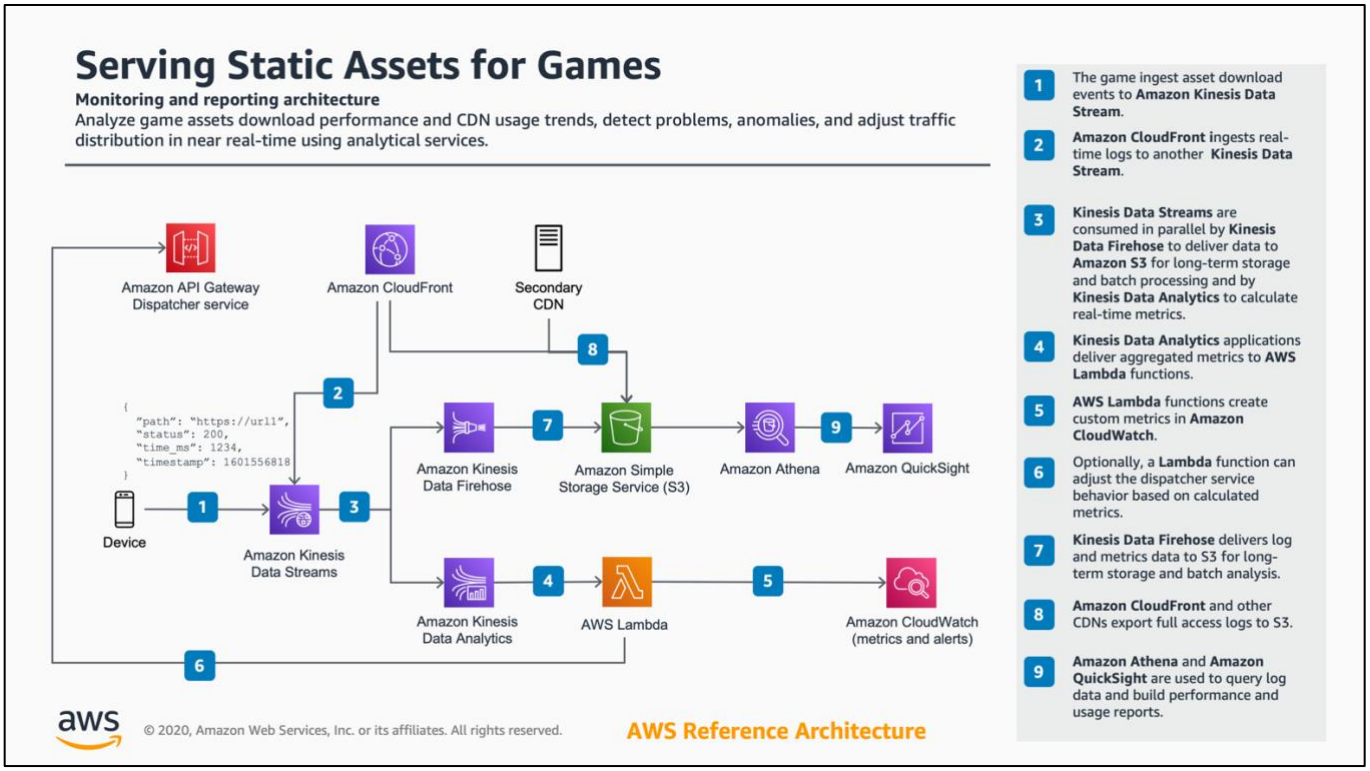

Figure 1 – Monitoring and Reporting example architecture

Asset download flow from CDN to customers

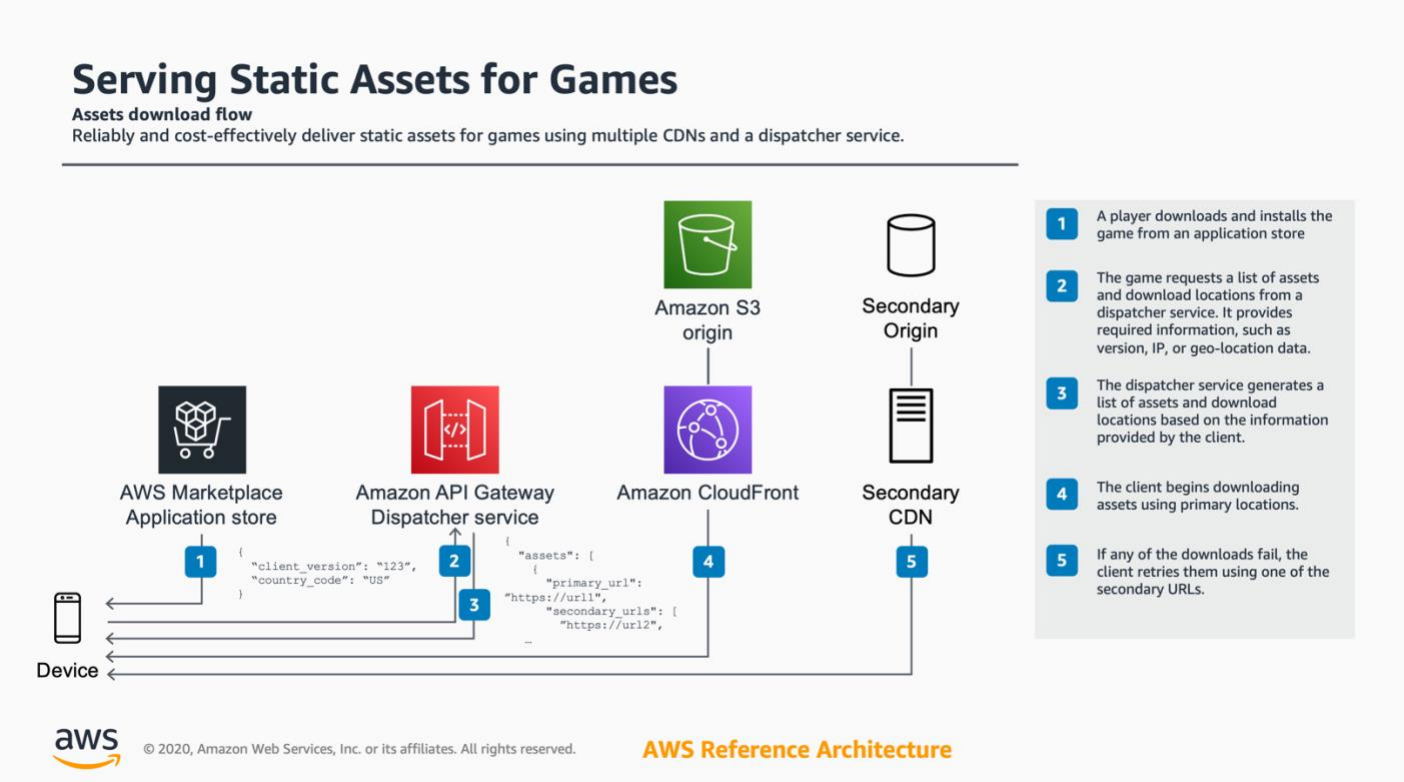

Figure 2 – Asset download flow from CDN to Customers example architecture

Asset upload flow to CDN

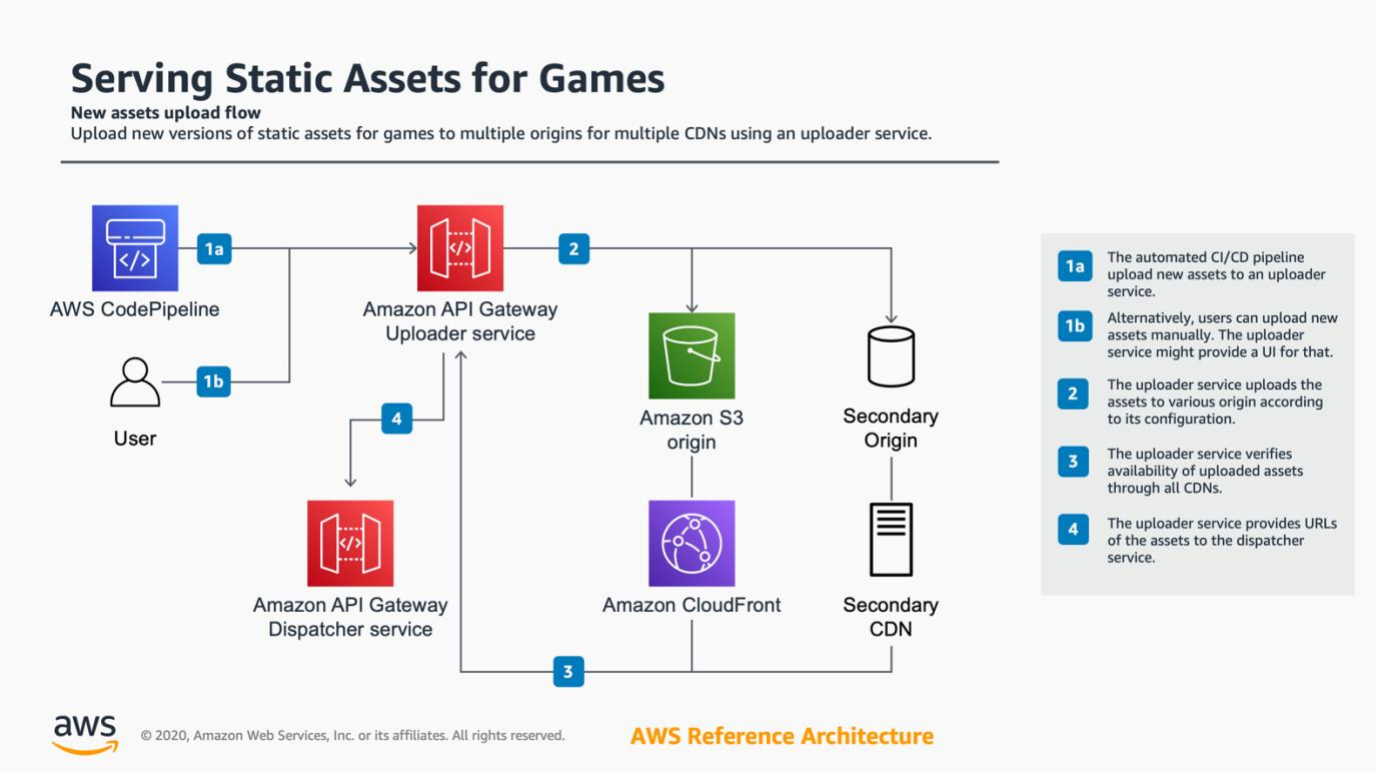

Figure 3 – Asset Upload Flow to CDN example architecture

CDN Evaluation Criteria

Game developers usually pay attention to the following factors when they evaluate or choose a CDN.

Performance and error-rate

The main performance metric that is used is available bandwidth to the client.

Latency is usually not as important as bandwidth. It is often fine to serve, for example, clients in Europe from the US, as long as the bandwidth is fine. One exception here is browser or Facebook-messenger games, where games request assets as they run, and players effectively play from CDN. In those games, lower latency improves user experience.

Another important requirement is low error rate. An unsuccessful download means retries and attempts to download from backup CDN, if there is any. This increases waiting time for players and increases chances of churn.

Cost

This is an important factor, since a CDN is a significant part of IT infrastructure spend. In addition to price per GB, you should consider the following:

- Is per-request pricing and number of requests included in a pricing agreement? If not, how is this pricing going to influence your CDN costs?
- Is it possible to establish usage commitments without overcharge fees? Due to the nature of the business, CDN usage is often unpredictable.
- How is the pricing different per geographical region? Does this align with the regions that you expect your players to play the game from? Is there an option of a flat rate? Is specific traffic split between regions required by CDN pricing terms?
- Consider the implications of origin fetch based on where you are storing your origin content. As mentioned earlier in this document, certain CDN providers may offer free origin fetches if content is stored in their cloud storage solutions (i.e. Amazon S3).

Regional Coverage

Customers often have to pay attention to regional specifics and even use multiple CDNs for reasons that include the following:

- **Regulations**: For example, a separate CDN with origin and points of presence located within the country would be typically required to serve Mainland China users.
- **Optimize performance in countries with significant user base**: Local points of presence and good peering with last mile providers are important to provide a good user experience for users in countries with sub-optimal network connectivity options.

Functionality

Even though serving static game assets is not the most sophisticated use-case for CDN, some *advanced* CDN functionality might come in handy.

A CDN solution should provide support management through APIs to integrate with upload tools, CI/CD pipelines and infrastructure-as-code solutions such as AWS CloudFormation and Terraform. It should also provide monitoring capabilities, and be able to deliver access logs to Amazon S3 buckets for centralized analysis. Access logs data can be used together with performance data from clients to build reports on CDN usage and performance. Some CDNs also support real-time logs and metrics streaming.

Compute on the edge functionality (like Amazon CloudFront Lambda@Edge) can help in some cases. For example, it can be used to generate unique installer names for every user for PC games to track pre-launch user experience, or to generate customized content depending on client or user-specific attributes in the request.

Ability to handle spikes

When a new game is released or an existing game releases new content, the excitement from players often leads to sharp traffic increases within CDN providers. Even though customers may often implement gradual rollout mechanisms to lessen the traffic spike, popular games still commonly experience traffic volume increases that can negatively impact CDN performance if not managed effectively. It is important to test a CDN's ability to handle traffic spikes in advance and coordinate with CDN providers to provide early notice of planned large-scale download usage. A multi-CDN setup allows

spill over traffic to secondary CDN providers if the primary provider doesn't provide sufficient performance or capacity.

Security

Typically, a user's authentication and authorization to download content is not a significant concern, since the majority of content distributed by a game is likely public content anyway. However, you might want to consider restricting access to betas or previews. For example, Amazon CloudFront provides the ability to serve private content with signed URLs and signed cookies to improve security. Additionally, CloudFront also provides Origin Access Identity, a feature that enables developers to restrict access to their Amazon S3 content only to their CloudFront Distribution, thereby forcing content to be downloaded through the CDN provider which further enhances security posture.

Other security features to consider include DDoS protection, such as that offered by AWS Shield, web application firewalls such as AWS WAF, and ensuring that correct permissions are setup for S3 bucket configuration. Additionally, Amazon S3 bucket versioning can provide a way to avoid unwanted object deletion.

Conclusion

The ability to deliver game assets in an efficient manner is a key determinant for acquiring new players, and this can vary from game to game. Before deciding on the content distribution strategy for your game, consider the design choices applicable for your game, review the various reference architectures, and the evaluation criteria mentioned in this document and determine the best fit for your game and players.

Contributors

Contributors to this document include:

- Eugene Krasikov, Sr. Solutions Architect, Amazon Web Services
- Kyle Somers, Sr. Solutions Architect, Amazon Web Services
- Nari Gopala, Worldwide Tech. Leader, Amazon Web Services

Further Reading

For additional information, see:

- Amazon CloudFront Developer Guide for real-time logs
- Dynamic Media Content for Games Reference Architecture
- CloudFront Real-time Monitoring Code Sample

Document Revisions

Date	Description
November 2020	First publication