Google Cloud Platform

Dedicated Server Gaming Solution

Introduction

As the number of people who play games continues to increase, a massive amount of computing resources will be required to power the compelling gaming experiences. This paper presents a highly scalable and reliable gaming implementation that leverages Google App Engine and Google Compute Engine for real-time player interactions. Core game elements, such as game matchmaking and player customization, are powered by App Engine while Compute Engine is utilized for running dedicated game servers [1] and common game engines.

[1] Dedicated game servers refers to the custom binaries responsible for handling real time player interaction.

Key points covered in this solution are:

- Scaling to serve hundreds to millions of players.
- Utilizing the Google Cloud Platform to build a fully featured game experience.
- Leveraging App Engine for front-end interactions and maintaining game state in the datastore.
- Orchestrating and autoscaling Compute Engine dedicated game servers with App Engine.
- Gaining business insights by analyzing massive user and game datasets.

Online gaming has grown from just a few people running game servers in their garages to millions of players enjoying a seamless online experience with matchmaking, in-game stores, and friend lists. These common game components have resulted in the development of sophisticated distributed systems which rival high performance computing and large scale web implementations. The fierce competition to develop and deliver the next "blockbuster" game or viral social sensation requires game developers to carefully manage their resources in order to focus on critical game components. Overprovisioning a cloud platform or focusing on unnecessary complexities results in significantly less human and financial resources to focus on gameplay or graphic assets. By utilizing Google Cloud Platform, game developers can focus on creating unique game experiences while taking advantage of Google's extensive experience in developing distributed systems.

This solution leverages the scalability and reliability of App Engine to match players with game sessions running on fully managed Compute Engine game servers. App Engine is an extensible platform that can be used to provide required features such as user profiles, game matchmaking, in-game store, social communities, and mobile engagement. Ideally, App Engine can be utilized for to powering all aspects of the online game, but developers often require access to virtual machines for running common game engines and software development kits (SDKs). Many aspects of this solution can also be used for a pure App Engine implementation, but the primary focus of this solution is the use case that requires dedicated game servers on Compute Engine.

The products used in this solution are:

Google App Engine

- Powers the main graphical user interface to provide game and user settings
- Provides Matchmaking and server browsing
- Distributes load to Compute Engine instances
- Maintains clusters to handle player gameplay load

Google Compute Engine

- Runs custom game servers

Google BigQuery

- Analyzes massive game and user data sets

Google Cloud Storage

- Stores game server binaries
- Distributes game client binaries and game assets
- Stores backup logs to process and ingest into BigQuery

Scenario

Dora has a spare moment and wants to play a few rounds of her favorite online game, Giant Robot Smash 5000. She opens her laptop, launches the game, and signs into her custom profile.

Before starting a competitive multiplayer match, Dora notices there are new missile launchers available for purchase in the game's store. After a few quick clicks, she purchases new equipment through Google Wallet and configures her favorite giant walking robot.

Now Dora is ready to start a new game and save the galaxy (again) and she joins an alliance with a few of her friends who are currently online. The group requests to join a server running their favorite game mode and within a few seconds they are sitting in their mech's cockpits, ready to accept any challenge.

Overview of the Solution

The reference architecture diagram, shown in Figure 1 below, provides a high-level overview of how Compute Engine and App Engine integrate to create a scalable and reliable online gaming solution.

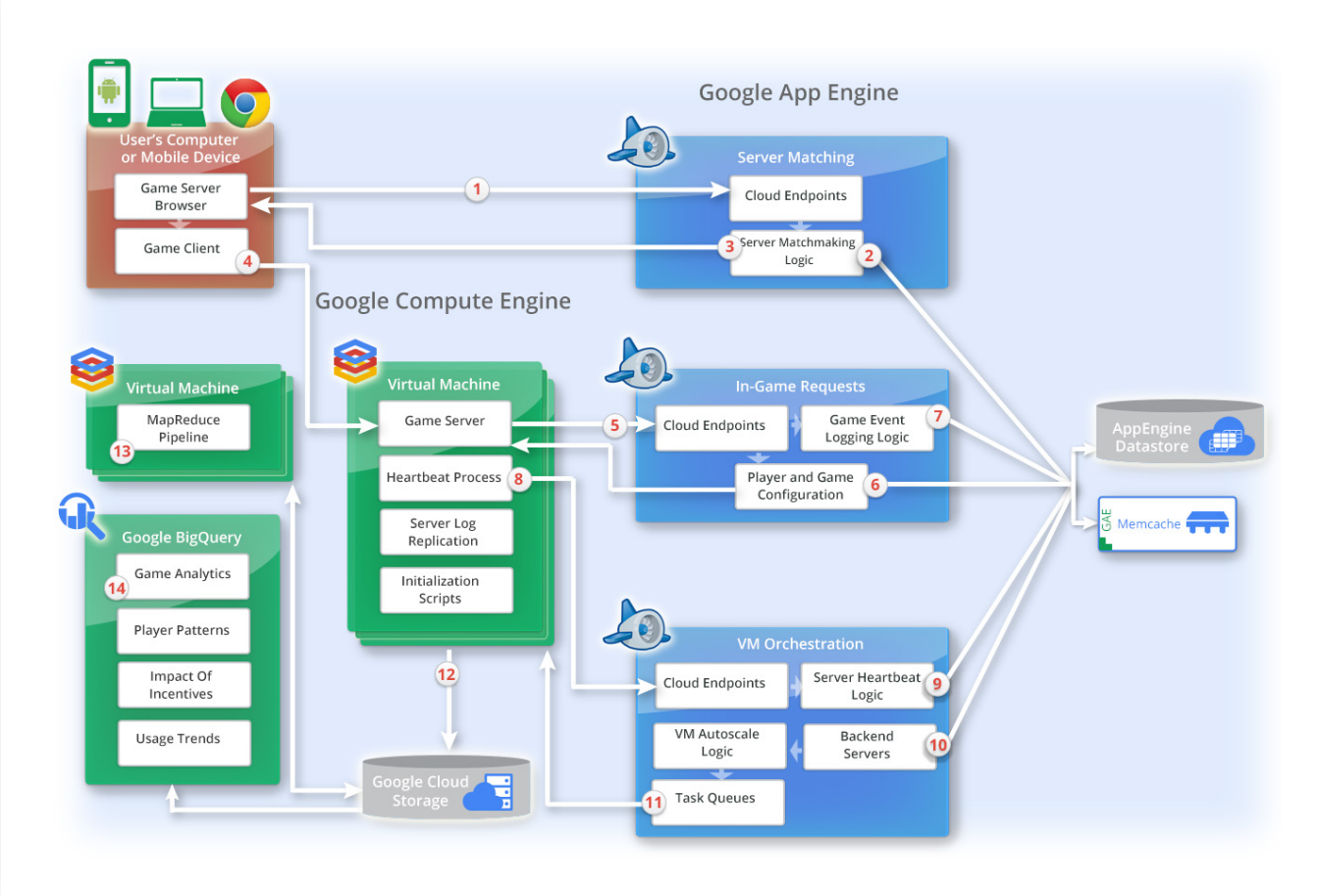

Figure 1. Reference architecture diagram for an online gaming solution

Key Components of the Proposed Online Gaming Solution

1. Game Server Selection
2. Player's Game Client Connecting to Dedicated Game Server
3. In-Game Requests and Google Compute Engine Instance Health Checks
4. Autoscale Game Servers
5. Store Logs for Analysis and MapReduce
6. Google BigQuery Analysis of Massive User and Game Datasets

A user starts playing a match of their favorite game by loading the local application or navigating to the game's website. If it is their first time playing, all client binaries and game assets can be download from Cloud Storage. Although game clients for mobile devices and personal computers will vary, the core

game features can be provided for all devices. Core features include updating user profiles, managing player configurations, and checking friends' achievements. App Engine can be utilized for all these devices by directly serving websites or providing a RESTful API for accessing all the required information.

The following section describes each of the key components of the proposed gaming solution (Fig. 1) in more detail:

1. Game Server Selection

Allowing players to join a game server and interact with other players is one of the most important components of the main interface. Matchmaking is an integral part of this gaming solution because it matches players with people in the same region and game modes. Depending on the search, performance, and scalability requirements, this solution can also be extended to include a full featured server browser and search capability by leveraging Google Cloud SQL, Search API, or Datastore.

2. Player's Game Client Connecting to Dedicated Game Server

Once the player selects a server to join and the game client receives the dedicated server's IP address, the player's game client establishes a connection to the dedicated server running on Compute Engine and loads in-game assets. The Compute Engine game servers are responsible for handling all player interactions through low latency client server communication. Information about designing a multiplayer game server is beyond the scope of this paper. When designing multiplayer game servers, it is recommended to leverage existing game servers and software development kits.

3. In-Game Requests and Compute Engine Instance Health Checks

When a dedicated game server is running on Compute Engine, it may need to send in-game requests to App Engine. If a player has purchased items from a store or created custom game configurations, App Engine can serve as the source of truth for this information. Additionally, the dedicated game server can communicate back to App Engine to update player scores, statistics, and experience.

After a game match has completed, players can either remain on a game server for a new round or be redirected towards server matchmaking. Player's scores, match statistics, and ingame store recommendations can be displayed between matches. If a dedicated game server terminates unexpectedly, the game client must handle this event and redirect players towards server matchmaking for a new session.

4. Autoscale Game Servers

Autoscaling is one of the first background tasks that does not significantly affect gameplay, but is critical to building a scalable, fully featured game. This step indicates the dedicated game server autoscaling logic implemented by a developer in App Engine. As the number of players increases, virtual machine orchestration logic creates new dedicated servers to handle the increased load. Similarly, if the number of players decreases during the day, unused dedicated servers can be terminated to eliminate unnecessary expense.

5. Store Logs for Analysis and MapReduce

Google Cloud Storage is recommended for storing files, such as server logs and output from MapReduce pipelines. The dedicated game servers on Compute Engine produce a significant amount of valuable data for understanding player behavior and troubleshooting software bugs. In order to store this data long-term, files should be regularly uploaded to Google Cloud Storage from Compute Engine Instances using a background process. If MapReduce pipelines are required for transforming and aggregating data,

the relevant files can be downloaded from Google Cloud Storage and processed on additional Compute Engine instances. Output from the MapReduce jobs can be stored in Google Cloud Storage where it can be used as input for additional pipelines, ingested into Google BigQuery, or compiled into reports.

6. Google BigQuery Analysis of Massive User and Game Datasets

Integrated into this solution is Google BigQuery, an ad-hoc query tool for analyzing massive datasets in real-time. When dedicated game servers are hosting millions of active players, billions of rows of useful data can be produced. Whether it is raw game logs or MapReduce output, the data can be ingested into Google BigQuery from Google Cloud Storage with a predefined schema. After ingestion has finished, SQL-like queries complete within seconds and can be used to obtain valuable information such as user engagement and the impact of game

Implementation Details

The following section provides implementation details for distributing player load and providing core game functionality required for creating a full-featured game experience. The primary focus of this solution is on the core scenario of distributing game servers to handle real time player interactions. This solution can be expanded to provide additional features as a full store and social community model, but they are outside of the scope of this paper.

The following architecture diagram (Fig. 2) is an overview which describe how a scalable, dedicated server gaming solution can be implemented.

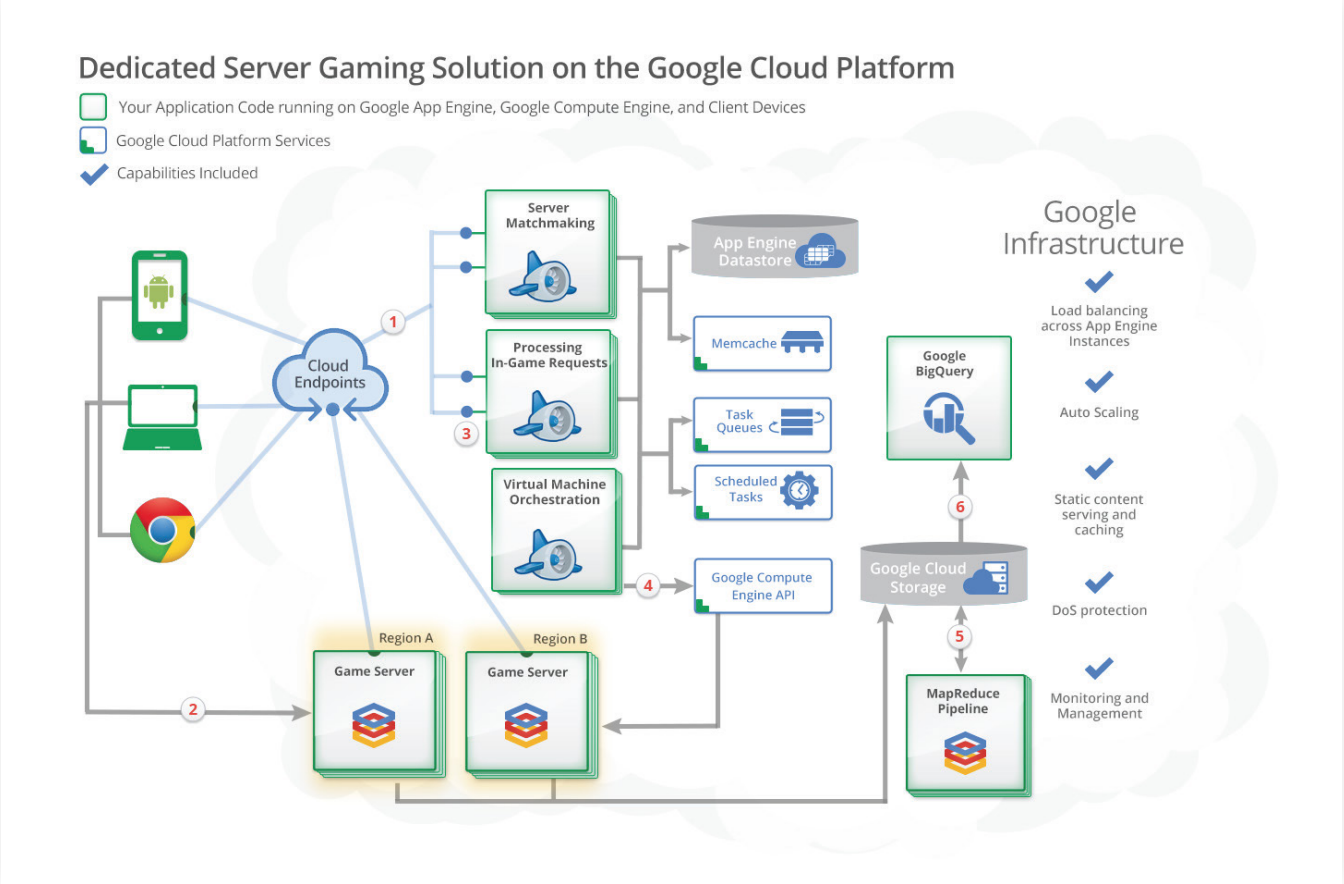

Figure 2. Implementation Details for an dedicated server gaming solution

Walkthrough and Implementation Details

1. Game Server Selection Request

Players use the game server browser to request a list of recommended game servers based on matchmaking criteria. This request is submitted through Cloud Endpoints which provide an authenticated RESTful API powered by App Engine.

2. Game Server Matchmaking Logic

Server matchmaking logic provides users with a list of recommended servers. Depending on the scale and frequency of server matchmaking requests, there are different techniques to implement this solution. One approach is to use App Engine background tasks to maintain a list of recommended servers in each datacenter and store the list in Memcache for quick retrieval. The logic for recommending servers depends on the type of game. Some games, for example, are lowest load to minimize latency, while other games need to put people in servers so there are a good number of people to play against. Although Memcache provides a high performance, distributed memory object caching system, the recommendations must also be stored in App Engine Datastore in order to handle Memcache evictions. The recommendation background tasks can be scheduled to run every minute by the App Engine Cron service. It is important for the background tasks to maintain a list of recommended servers for each region since players usually want to connect to the lowest latency server. Other server selection techniques may round-robin through available servers or provide a reasonably sized list to the client so it can identify the lowest latency servers. More complex solutions can include tasks that maintain player counts, load, latency, and states for all servers. Solutions can also be designed to query dynamically for each request.

3. Game Server Matchmaking Results Returned

The results from game server matchmaking are returned to the client where the player either selects from a list or the client automatically determines the ideal server.

4. Game Client Connects to Dedicated Game Server

The game client attempts to connect to the IP address of the selected dedicated game server. If the connection fails or the server is full, the client can attempt to connect to other servers provided or direct the player back to server matchmaking.

5. Game Server In-Game Requests

After players establish a connection to the dedicated game server, the server is responsible for handling all events from the player and providing information about other players currently in the match. App Engine is utilized to maintain a consistent game experience across all dedicated servers by handing important events and providing player information. For example, if the player has a custom configuration, a request to App Engine will provide information about the configuration and allow the player to access all purchased items. As players gain experience and important in-game events occur, details can be sent to App Engine in order to maintain a complete view of all players. An authenticated request to Cloud Endpoints and the provided RESTful API is an easy way to connect the game servers to App Engine.

6. Request Player Configuration

When games allow a player to purchase items or create a custom character configuration, the information must be maintained in a reliable and scalable database. The App Engine Datastore is designed to scale for web applications that serve millions of users and the underlying Google Megastore

technology is leveraged across Google. The App Engine Datastore is recommended for storing all player information because it seamlessly scales as a game grows from hundred to millions of players. Memcache can also be leveraged to store results from frequent App Engine Datastore queries in order to improve performance. Since Memcache is a finite resource, smart usage is encouraged. If complex SQL queries are required or MySQL must be utilized for other reasons, Google Cloud SQL provides a fully managed and highly available relational database-as-a-service. Although it is tightly integrated with App Engine, Google Cloud SQL is not designed to scale infinitely and load testing is highly recommended in order to understand real-world database performance.

7. Store Important In-Game Events

Handling and storing important events, such as players gaining experience after in-game actions, is a critical part of creating an addictive and engaging game. Similar to requests for player configuration, these requests are handled by App Engine and key information can be stored in the App Engine Datastore. The major difference between these two types of requests is that in-game events can occur at a higher frequency for all active players. For example, a player's configuration may only be obtained at the start of the match whereas in-game events can happen every time a player's character gains experience. Although the App Engine Datastore can scale to handle thousands of events from millions of users, entity groups, NoSQL and eventual consistency must be understood in order to eliminate potential scalability concerns. Detailed technical discussions about these topics are located in the App Engine developer documentation.

8. Server Heartbeat Process

A critical component of maintaining a healthy cluster of dedicated game servers is continual tracking of each server's statistics and health. Once again, Cloud Endpoints are leveraged to provide an authenticated RESTful API where a process running on each Compute Engine instance can provide utilization statistics. Hardware related information, such as CPU and RAM utilization, can be provided along with game specific information, such as average player latency and number of players active on the server.

9. Store Server Health and Statistics

The Heartbeat process running on each Compute Engine instance can provide a large amount of valuable information. Server heartbeat logic is required to parse and store relevant data. Information directly relevant for autoscaling, such as the number of players active on servers and average latency, should be cached in Memcache for quick retrieval by the autoscaling backend processes. Any important values should also be stored in the App Engine Datastore to protect against Memcache eviction. If this information is also relevant for analytics and maintaining server history, it is recommended to store all historical values in a separate table which is utilized independently of autoscaling.

10. Autoscale Dedicated Game Servers and Maintain a Healthy Cluster

Although there are many approaches to autoscaling Compute Engine resources with respect to player load, the common component involves running a scheduled task every minute with App Engine Cron service. The ideal number of virtual machines can be calculated by a predetermined schedule or by analyzing the number of available positions in game servers or player latency. The other important input to autoscaling is determining the currently active healthy machine count by pulling recent heartbeat process data from Memcache or the App Engine Datastore. The difference between the ideal and current number of game servers can be utilized to create or delete instances. Additionally, any unhealthy servers should be configured to eliminate them from server selection and delete the instances once there are no players on the server. If game servers need to be migrated between different Compute

Engine zones, the autoscaling logic can be used to create instances in the new zone while terminating vacant instances in the old zone. This is a very high-level overview of autoscaling game servers and it is strongly recommended to carefully implement the scaling algorithms. Focus on avoiding issues such as overshoot and noisy responses. Compute Engine servers are billed hourly, so in order to reduce unnecessary costs from unused Compute Engine instances, avoid the constant creation and deletion of instances.

11. Create and Delete Dedicated Game Servers

Once a game server must be deleted or created, a task is added to an App Engine task queue. A separate background task is responsible for pulling server maintenance tasks and executing Compute Engine API calls. Additional backends can be added if the number of required API calls increases beyond the limits of a single backend. If there are few Compute Engine API calls, the server maintenance can be handled by a scheduled tasks to reduce App Engine resource utilization. It is recommended to include a timestamp with every server maintenance task in order to create alerts if a backlog develops in the system. Push queues can be used as an alternative to pull queues and it is recommend to run load tests in order to understand how each autoscaling system responds under heavy utilization. Although Google does not provide a load testing service, common open source technology can be run on Compute Engine or third party services can be utilized for extensive load testing.

12. Store Logs in Google Cloud Storage

From in-game server logs recording every player's actions and movements to end-game statistics, many log files are generated on each game server, from in-game server logs recording every player's actions and movements to end-game statistics. These files can be copied to Google Cloud Storage using a background process running at regular intervals. If critical data is stored in the files, they should be stored on persistent disk in order to prevent data loss if an instance terminates before the copy process has completed. Otherwise, storing files on ephemeral disk provides a lower cost alternative, but the disk will be deleted immediately after an instance terminates. Regardless of disk choice, it is always recommended to have an automated copy process for maintaining all logs and statistics in Google Cloud Storage.

13. Transform and Process Log Files

After collecting a large amount of raw log data from servers, the log files need to be cleaned, augmented with additional data, and aggregated to different levels. MapReduce or Extract, Transform, and Load tools can be leveraged to create data that can either be used for user facing features, such as store item recommendations, or ingested into Google BigQuery for analysis.

14. Reporting and Analytics

Google BigQuery is an important part of a gaming solution as it allows ad-hoc analysis of massive datasets containing user and game information. For example, it can be used to determine the impact of gameplay incentives, such as store sales or double experience weekends, on user retention and engagement. Additionally, Google BigQuery maintains consistent performance as data scales to terabytes and billions of rows.

Sample App

A sample application which demonstrates the high level concepts of this solution is available and it can be utilized as working reference. The core functionality of the sample application is as follows:

- Client queries App Engine for an IP address of a dedicated game server
- Client starts a new game by connecting to a game server running on Compute Engine
- Administrators can create and delete game servers from App Engine Administration UI
- Compute Engine instances report load levels to App Engine periodically
- Administrators can view load levels of all game server Compute Engine instances
- App Engine automatically adds new instances to the cluster if the cluster exceeds a max load threshold

Get the source on Github »

Conclusion

This solution demonstrates how developers can scale their online game to support millions of players while providing a full-featured gaming experience. By leveraging multiple components of the Google Cloud Platform, developers gain the scalability and reliability of App Engine while running industry standard game servers on Compute Engine. This allows game developers to quickly launch, iterate, and scale while simultaneously focusing on delivering a great game.