Upgrading to brainCloud from PlayFab

Groups, Friends and Messages

Table of Contents

Introduction

In this guide, we explore how some of PlayFab's social features, such as friends, groups, and chat, compare to brainCloud's and how to replicate your current implementation on brainCloud.

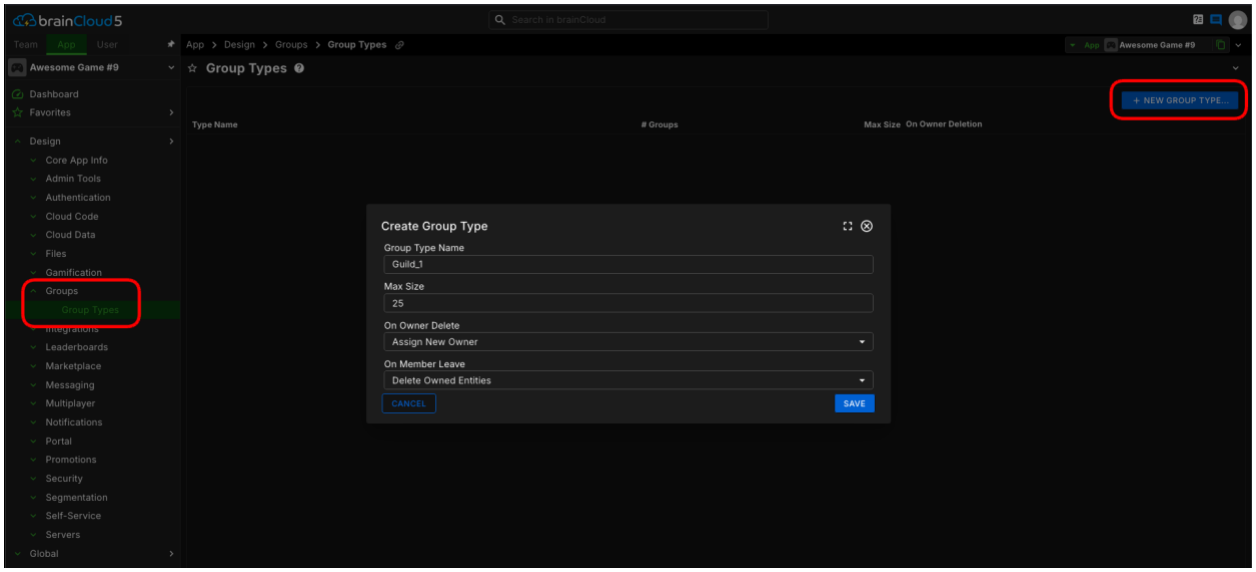
Groups

Both PlayFab and brainCloud offer a "Groups" feature, allowing players to form teams, clans, or guilds. The functionality is comparable between the platforms, so we won't delve too deeply. For a quick overview, skip to the Groups API Comparison table below.

Configuring & Creating Groups

A key difference between PlayFab and brainCloud is that you must create a "group type" before players can form new groups. This template sets the basic settings and distinguishes different group kinds, such as guilds, chat groups, families, or regional social groups.

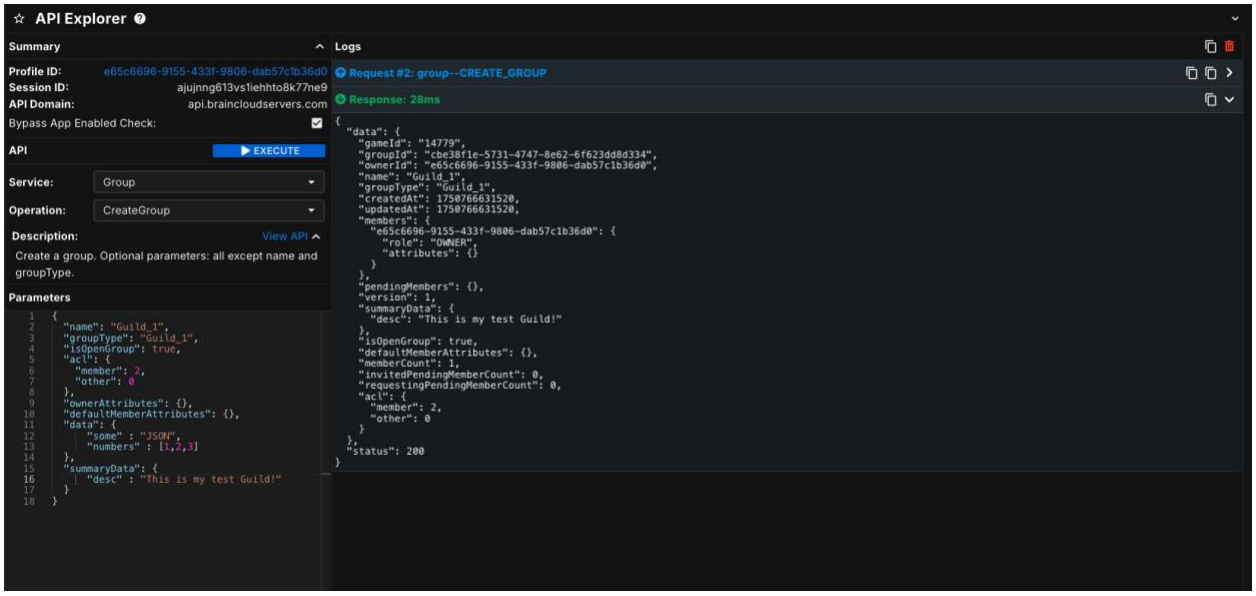
To create a new group type, navigate to **Design → Groups → Group Types** and click the New Group Type button at the top right of the dashboard.



We need to give the group type a name and a max membership size. The other options allow you to decide what happens to the primary admin permission when the group creator or owner leaves (assign a new owner or disband the group) and what happens to member data when a member leaves. You can find more information about group types here.

Creating Groups

Players can create new Groups using the CreateGroup request. This request primarily requires the group type and group name, but additional options are available that may ease your migration effort. Let's examine this request in the API Explorer. You can find more information about the API Explorer in the Cloud Scripts Document.

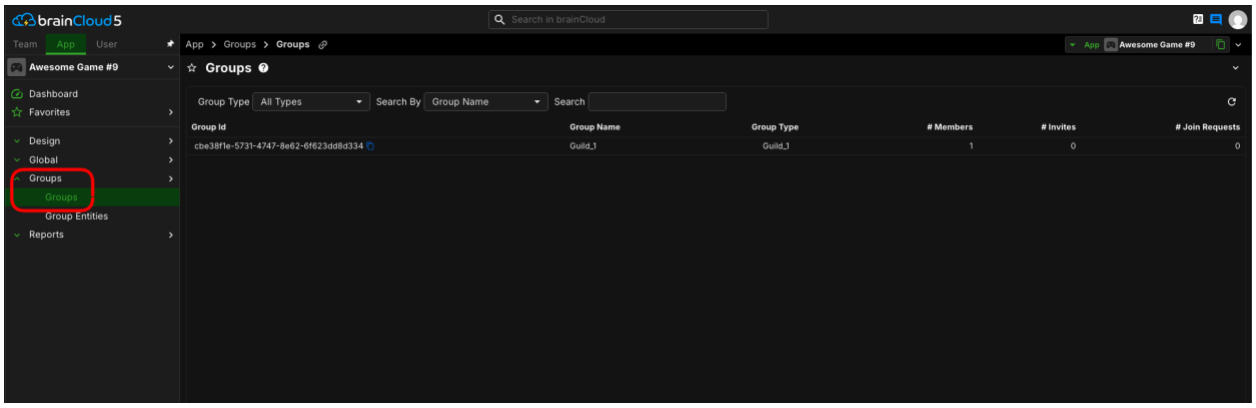


You can see additional parameters above. The “data” and “summaryData” fields can replace PlayFab’s Group Objects, which are JSON objects for storing key/value pairs in groups. You can set the group to private or public using the **isOpenGroup** parameter, ensuring new members use the invite/approval system instead of joining automatically. The access-control-list (acl) parameter sets the update permissions policy; members have full update permissions, while “other” (guests) have none.

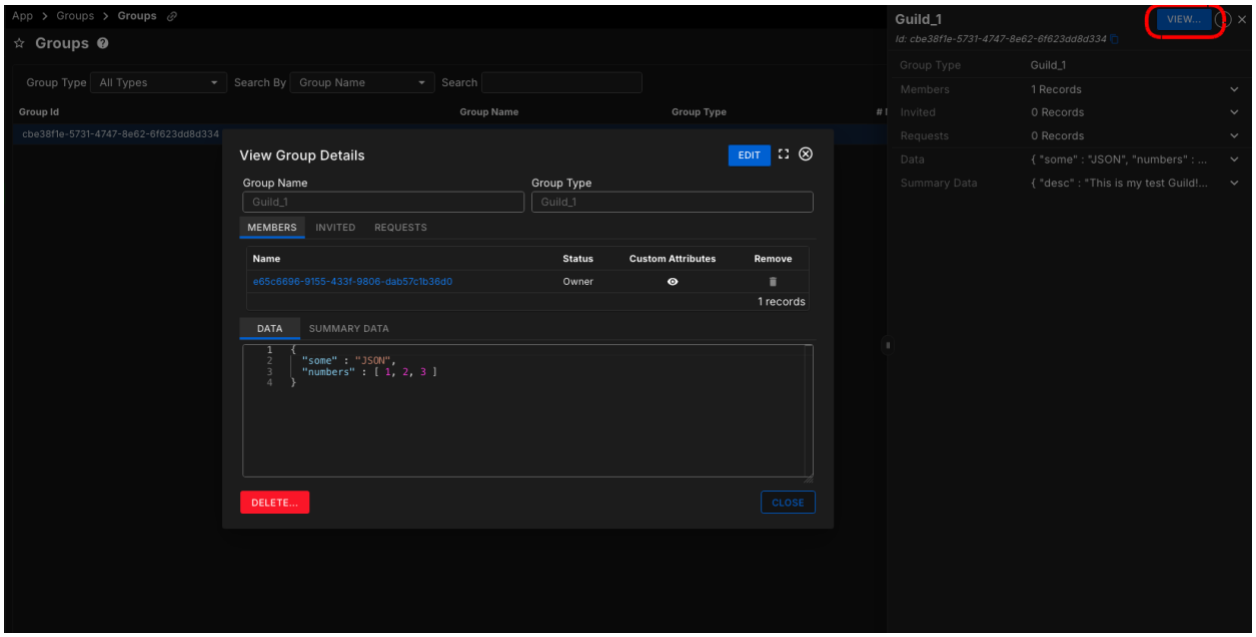
In the response above, just like with PlayFab, the new group’s ID is returned, which will be used to reference this group in future requests.

Groups Dashboard

As with PlayFab, admins and developers can view all groups and their information from the brainCloud portal. This dashboard is located in the **Groups → Groups** menu.



You can click on any group to view or edit its information, review invites and requests, and approve them.



Group Files

brainCloud allows binary files to be associated with groups, accessible only by members. There is a separate client API for group files that handles updating, deleting, and listing them.

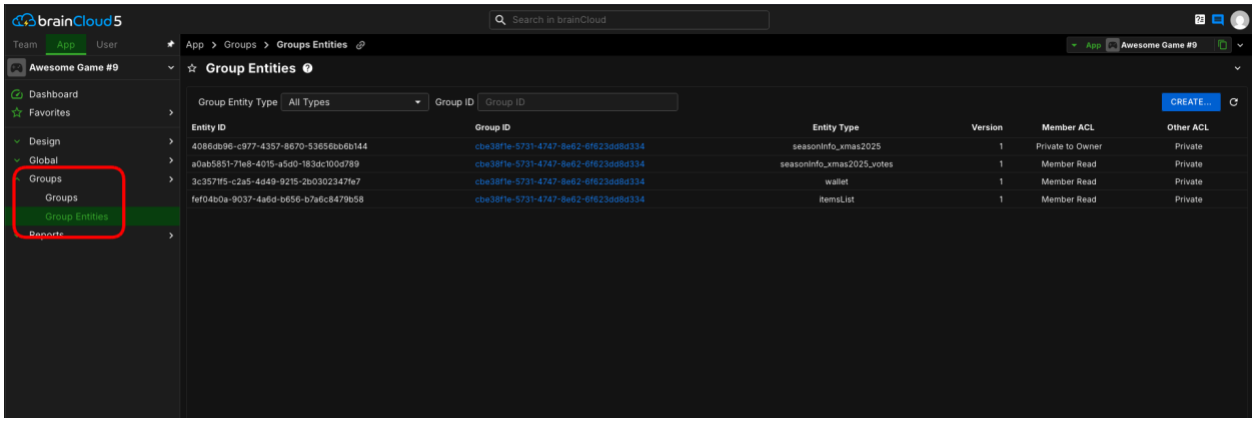
It's important to note that you can't upload group files directly. Instead, players upload user files using the UploadFile request and then move them into a group with the MoveUserToGroupFile MoveUserToGroupFile request.

You can learn more about brainCloud's CDN and File features in the Title Content, UGC & Files document.

Group Entities

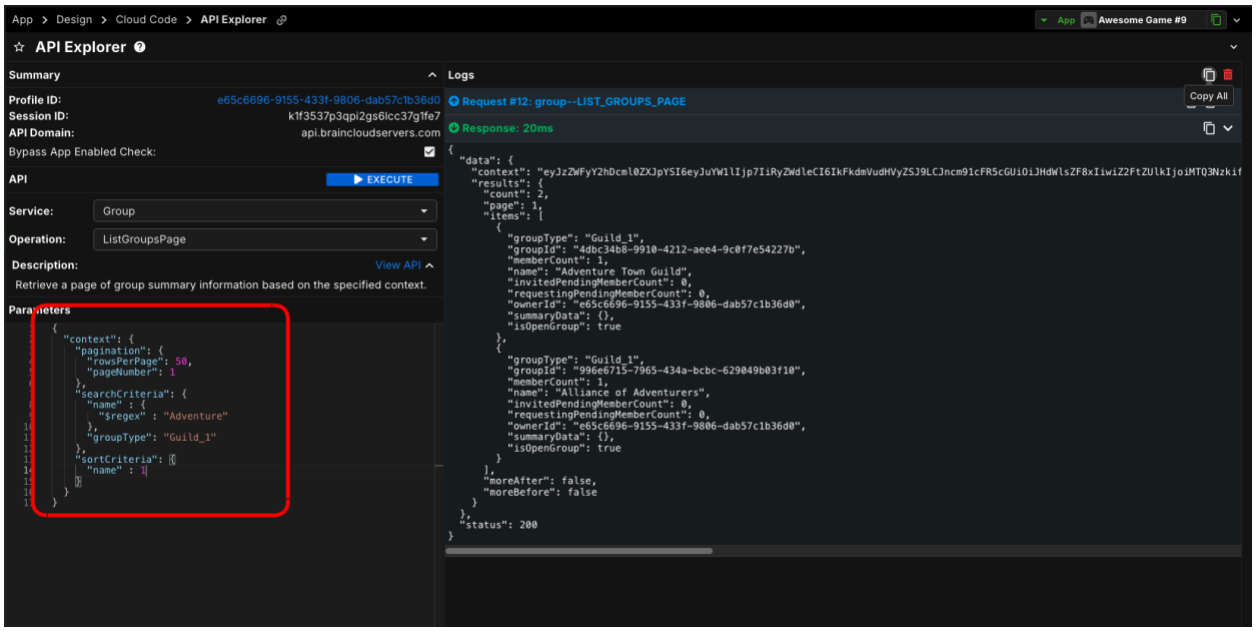
While groups have **data** and **summaryData** JSON fields for storing key/value pairs, brainCloud also allows more complex interactions using Group Entities. Groups can have multiple entities with their own access policies, which can be used to store group stats (IncrementGroupEntityData), group votes (using custom cloud code), group progression, etc.

You can view the Group Entities for a given group from the **Groups → Group Entities** menu.



Group Search & Listing

Unlike PlayFab, brainCloud allows you to set groups as public or private using the SetGroupOpen request. This feature also enables public groups to be listed and searched based on complex criteria. The default request for searching groups is ListGroupsPage. The example below uses this request to search for groups of type "Guild_1" with a partial name search, sorted alphabetically.



Groups API Comparison

As mentioned earlier, brainCloud includes much of the same functionality as PlayFab for Groups. Below is a table outlining these APIs and their corresponding brainCloud requests.

PlayFab	brainCloud
CreateGroup	CreateGroup
GetGroup	ReadGroup
DeleteGroup	DeleteGroup
AddMembers	AddGroupMember
RemoveMembers	RemoveGroupMember
ApplyToGroup	JoinGroup

InviteToGroup	InviteGroupMember
ListMembership	GetMyGroups ListGroupsWithMember
ListGroupMembers	ReadGroup ReadGroupMembers
ChangeMemberRole	UpdateGroupMember
AcceptGroupApplication	AcceptGroupJoinRequest
DeclineGroupApplication	RejectGroupJoinRequest
AcceptGroupInvitation	AcceptGroupInvitation
DeclineGroupInvitation	RejectGroupInvitation

brainCloud also offers additional functionality and APIs for Groups and Group Files not listed above. If your game includes custom cloud scripts needing migration, explore those extra APIs to see if they meet your needs. Check out our automation guide Cloud Script Document for more information.

Friends

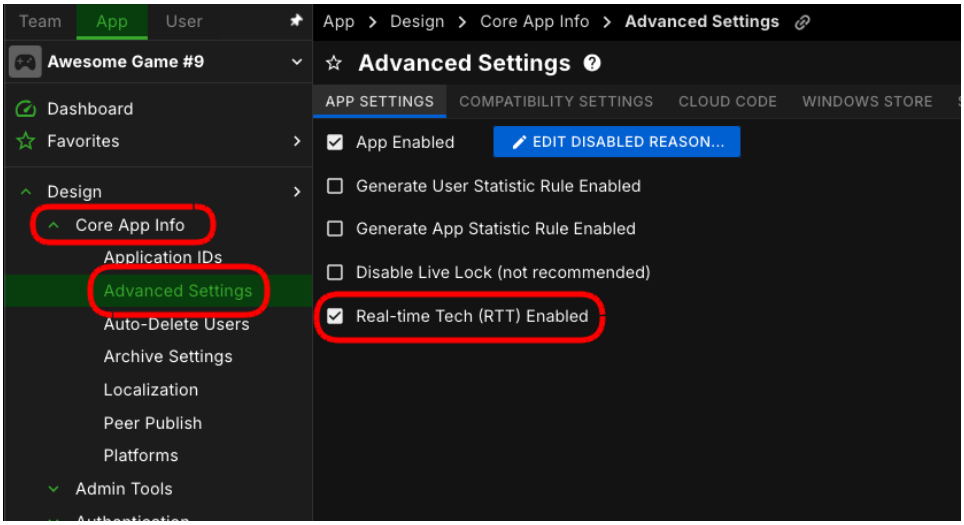
Friends work roughly the same in brainCloud as they do in PlayFab. You can add friends, remove friends and list friends (by platform if required) as you can with PlayFab, but also check the online status and get a profile summary for specific friends.

brainCloud provides an extensive set of APIs for searching for friends from third-party platforms, by name or partial name. This offers players and developers more flexibility in finding friends to add without needing extra custom cloud code.

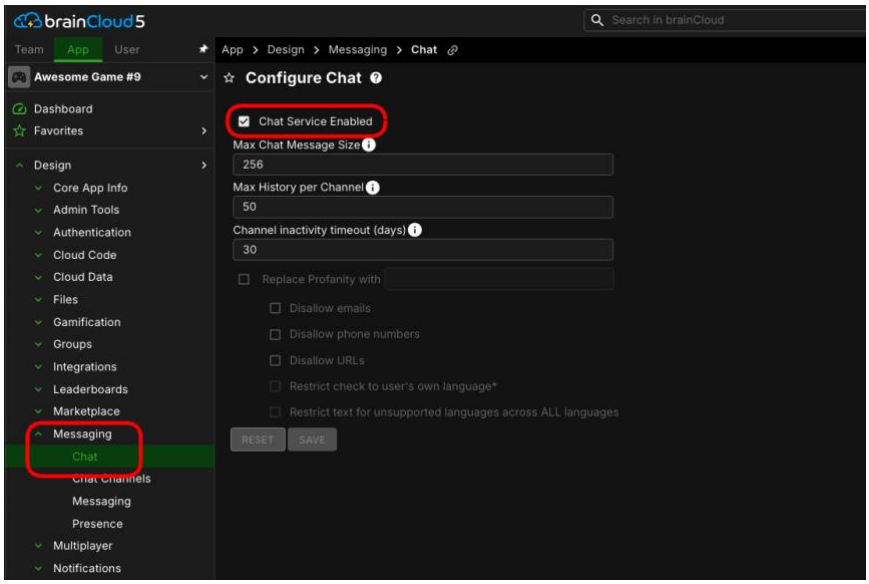
Chat Messages

PlayFab's chat service operates through their Party API, offering real-time communication via the PlayFab Party SDK. Similarly, brainCloud offers this service in their SDK based on brainCloud RTT. This section explains how to set up RTT and connect players to a chat channel for asynchronous communication.

To start, enable RTT on your app for the brainCloud Chat feature to function. Navigate to **Design → Core App Infos → Advanced Settings** and select "Real-time Tech (RTT) Enabled."



You can also configure the chat service in the **Messaging → Chat** menu. Here, you'll find settings for the chat feature, including options for profanity filtering if using WebPurify. There is a guide for integrating WebPurfity with brainCloud here but we will take a brief look at how we might implement a custom profanity filter later in this guide. For now though, you can leave all these settings as their default.



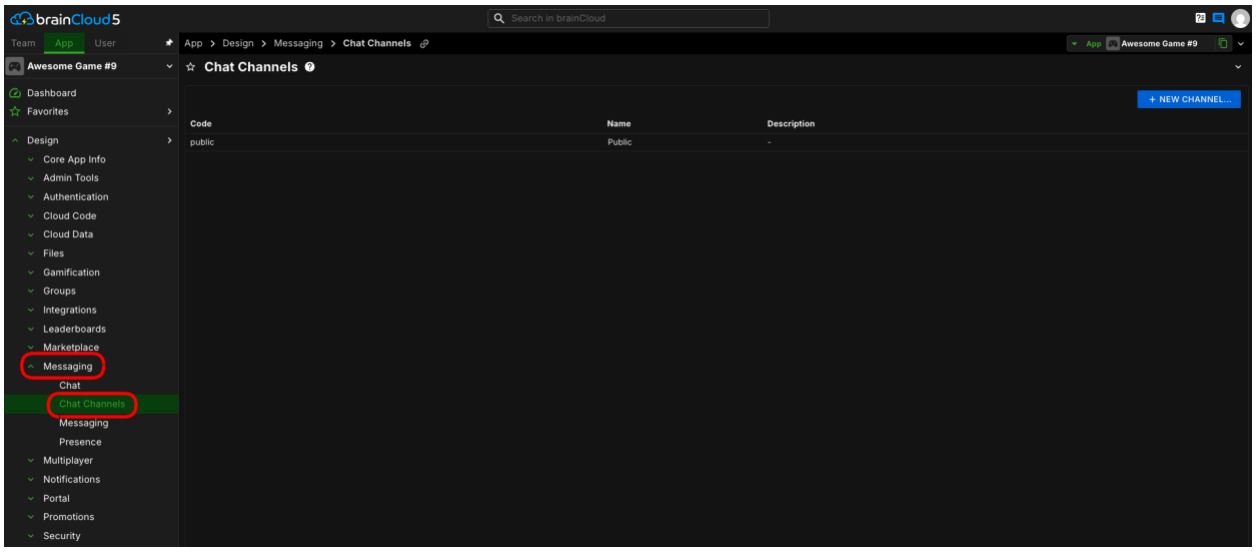
Chat Channels

Next, you may need to set up your chat channels. Chat channels are similar to Chat Control objects in PlayFab's Party API. Using a channel ID, players can join a channel to send and receive messages.

Channel IDs follow the format "appId:channel-type:channel-code" and are automatically generated per channel in most cases. There are three types of channels for client communication:

Global Channels

Global Channels are open for any player to connect to. brainCloud provides a default "public" channel, viewable in the **Messaging → Chat Channels** menu.



While you can create custom channels from this dashboard, remember that all these channels are public and can be joined by any player with the code.

The channel ID for the default public channel would be “<app-id>:gl:public”.

Group Channels

brainCloud creates a chat channel for each group players create. This is useful for private chat groups outside the systems using the Groups feature like teams, guilds, or clans.

The channel ID for a group channel would be “<app-id>:gr:<group-id>”.

Dynamic Channels

Dynamic channels can be created by players or developers on the fly and given unique channel codes, which need to be shared directly with players for them to connect. These channels are ideal for lobbies, matchmaking groups, or other non-public channels. However, they aren't needed for private messages between players, as there is already a Messaging feature for that, covered later in this guide. Dynamic channels can only be created using custom cloud code.

Dynamic channels are ephemeral and expire after 30 days of inactivity.

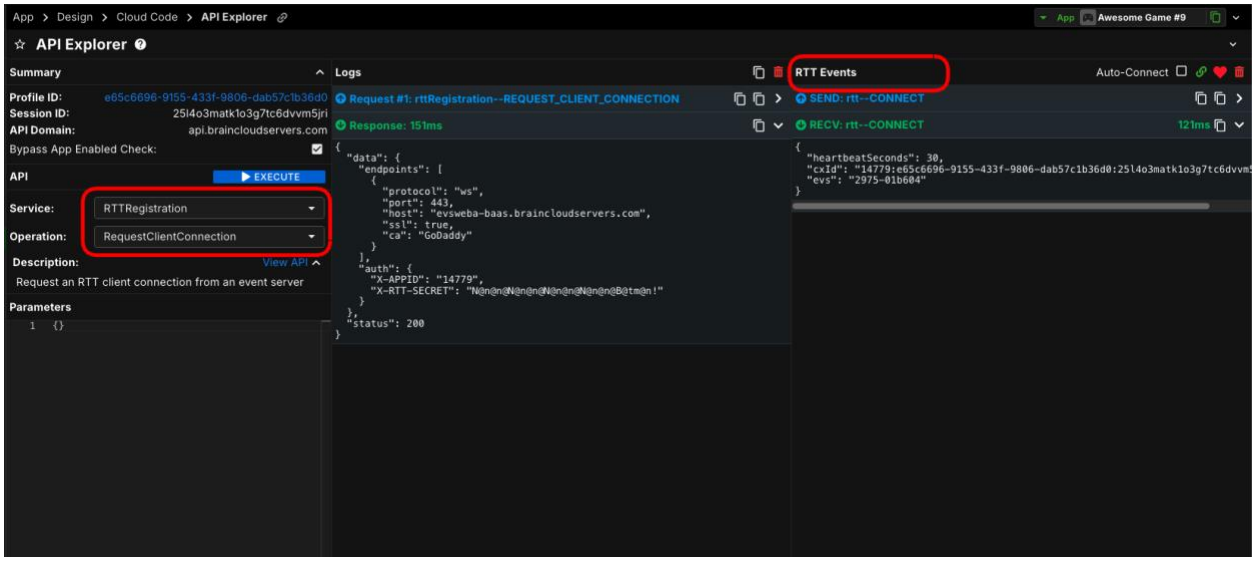
The channel ID for a group channel would be “<app-id>:dy:<channel-code>”.

Connecting To Channels

Connecting to channels is simple once you know the channel type and code. You can use the ChannelConnect request for this.

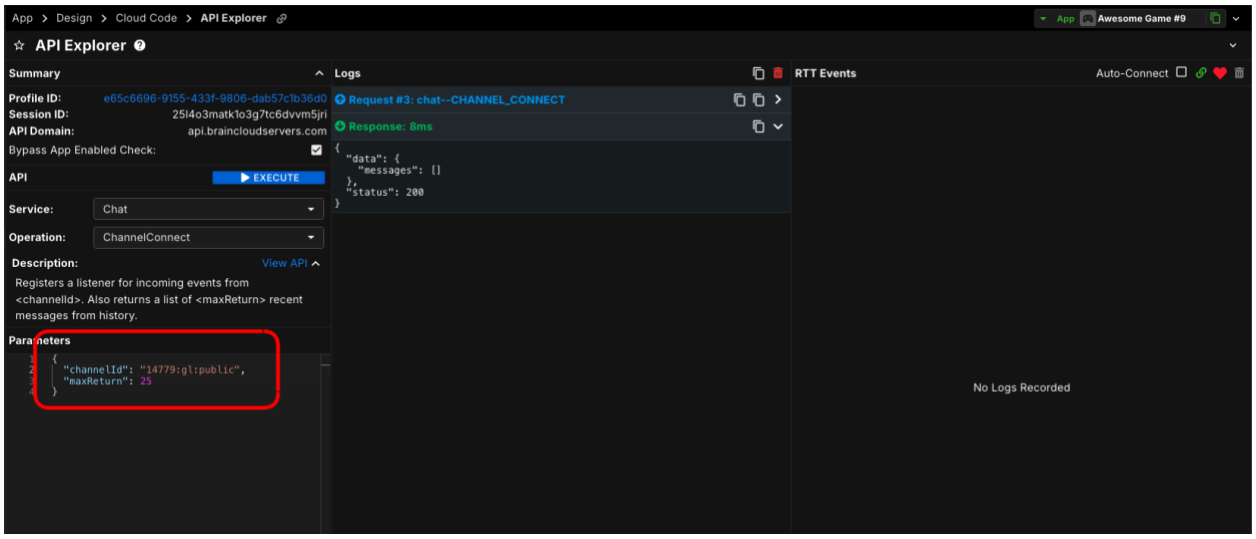
While client examples are available for download to test chat functionality for your migration, you can also test it through the brainCloud portal API Explorer. You'll need two windows open to see messages from each player you're testing with.

First, request an RTT client connection. This opens a panel on the left-hand side of the dashboard when you execute the request.



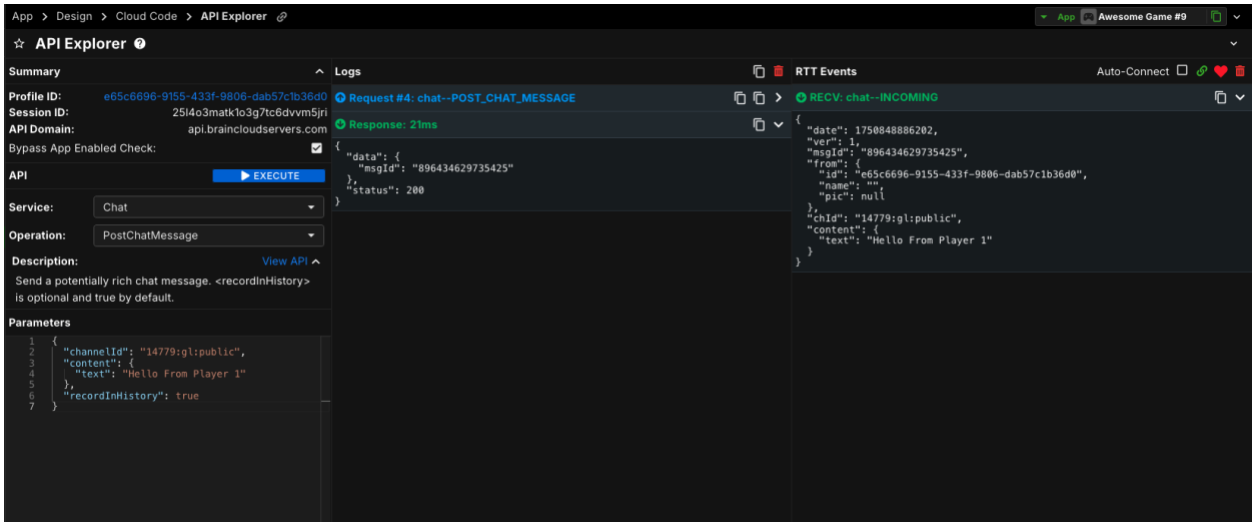
Note
For your actual client you will also need to enable the RTT service using EnableRTT. Check out the Chat documentation for more details.

Next, use the Chat service and the **ChannelConnect** operation with the public channel code. In this example, since there are no messages currently in the public channel, you'll see no messages in the response array.

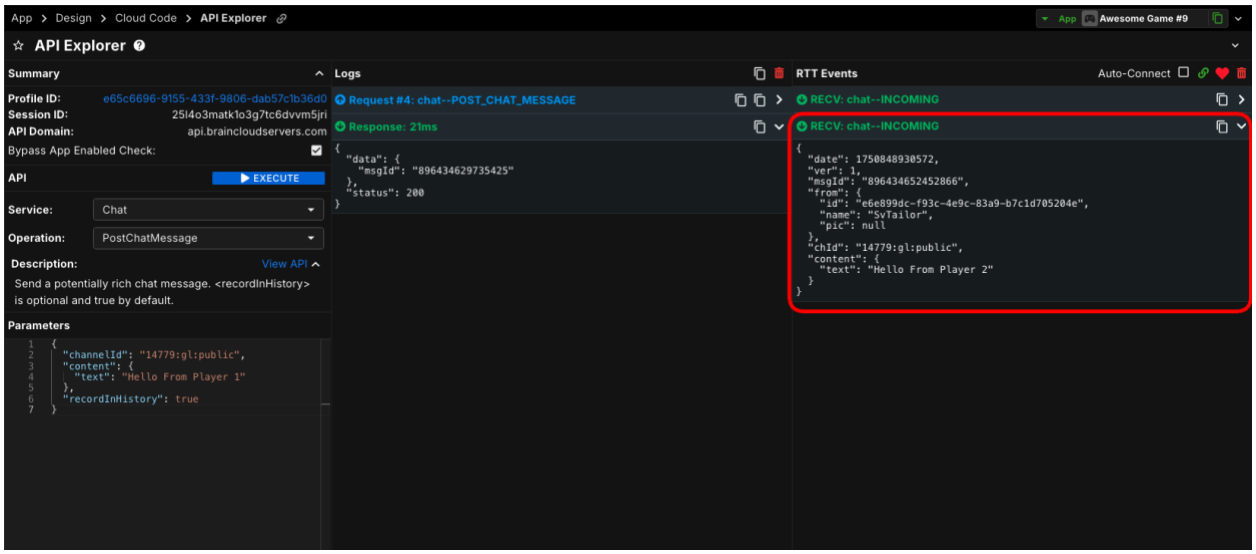


Posting Messages

Now that we're connected to the channel, we can start posting messages using the PostChatMessage request. You'll need the channel ID for posting, along with a JSON object containing your message content.



The RTT Events panel shows an incoming message from this player, indicating it was posted to the channel. Your second player should receive it as well. To test the channel, have the second player post a message, which will appear in this player's RTT Events panel.



Chat Moderation

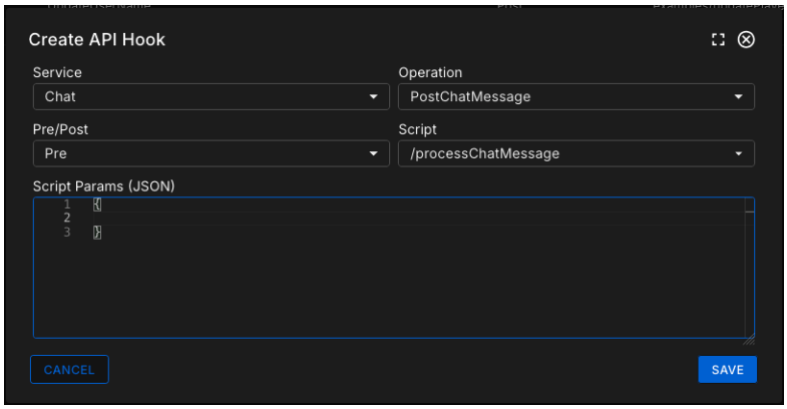
In a public chat system, profanity filtering and moderation are important. Before finishing this section, we'll briefly show how to catch chat messages in-flight for processing. Create a new cloud-code script called processChatMessage and add the following code.

```
"use strict";


function main() {

 var response = {};


 let isMessageValid = true;


 // [1] - Get the message contents //

 const messageTxt = data.message.content.text;


 // [X] - check messages for profanity //

 //      your code here //

  // [X] - send message to 3rd party service //

 //      your code here //


 // if the message is invalid //
```

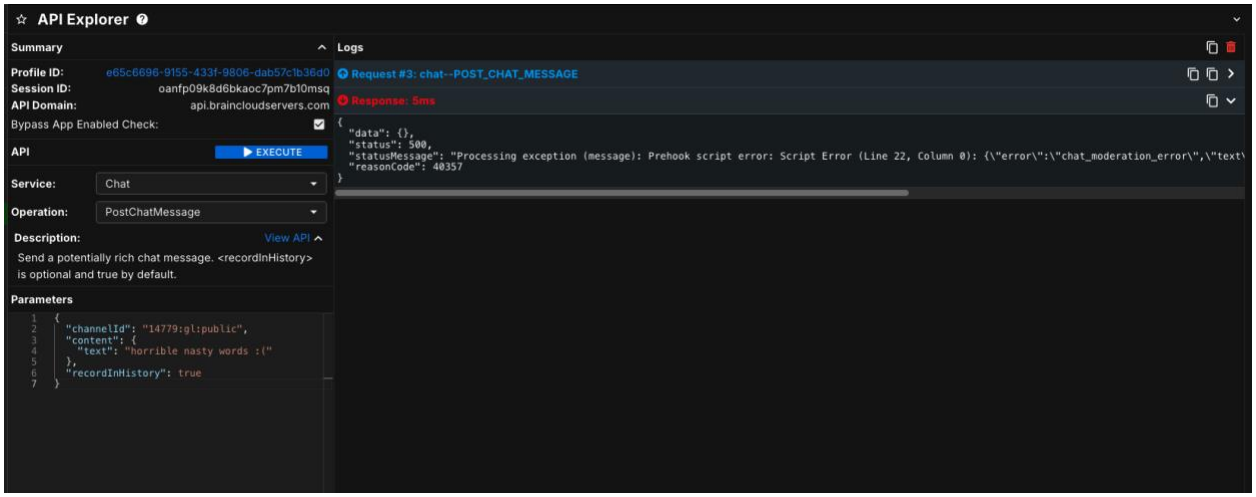
```
 isMessageValid = false;


 // Throw Error //
 if(!isMessageValid) {
   throw JSON.stringify({
     "error" : "chat_moderation_error",
     "text" : messageTxt,
     "warning" : "inappropriate_content"
   });
 }


 return response;
}


main();
```

In this example, an error is thrown if inappropriate content is found, preventing the message from posting. You could extend this to send the message to a third-party service using an HTTP request, or you could use Global or Custom Entities to store a list of inappropriate words or phrases which you could query before posting the message (see the Title Content, UGC & Files. for a guide where this is covered).

Finally, to ensure this script executes before posting the message, add it to the **Chat → PostChatMessage** API Pre Hook.



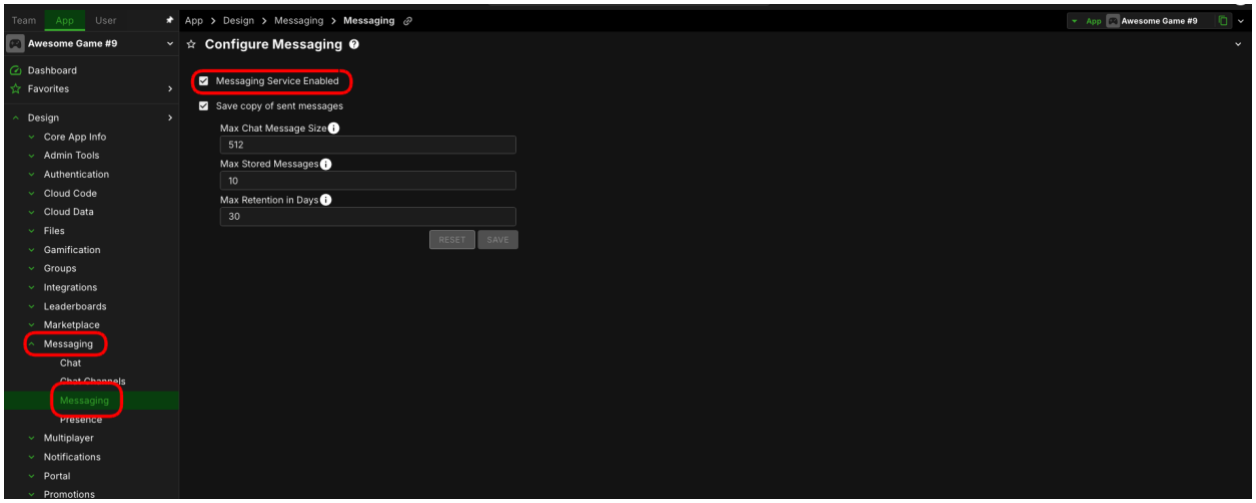
Now, if you test the PostChatMessage request, you'll see your error returned to the player posting. Importantly, the message won't appear in the RTT Event panel for either player.



Messages & Private Chat

As mentioned earlier, brainCloud's Chat feature supports communication between groups of players, typically more than two. For private messages between two players, you can use the Messaging service.

Like the Chat service, the Messaging service has a dashboard for settings and configuration, found in the **Messaging → Messaging** menu. You can leave these settings at default for now, but ensure the messaging service is enabled.



Sending Messages

Messages can be sent using the Send Message request, requiring only the profile ID of the recipient(s).

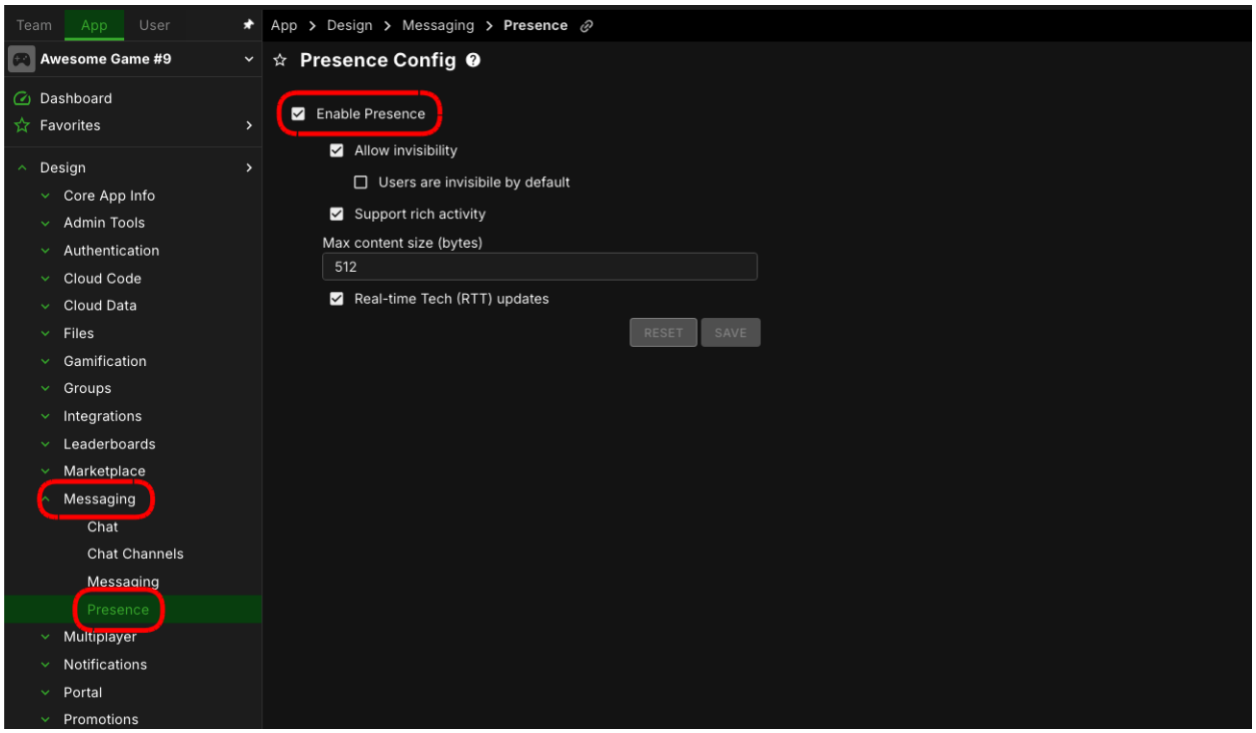
Inbox & Listing Messages

The Messaging service is useful for private messages and can function as an inbox system, where players can view sent, received, read/unread messages, and delete them. You can explore the Messaging API in more detail here.

Presence

Presence allows players to see the online status of others, such as friends or group members. This is especially useful with chat, messaging, or matchmaking, enabling players to check online status before initiating conversations or match requests.

Presence settings can be found in the **Messaging → Presence** menu and must be enabled for the feature to work.



Once the presence system has been enabled you can register listeners on the client to get informed of changes to the presence of friends, groups or other users, as well as updating your own state. For more on Presence, check out the guide here and the Presence API documentation here.