Khan Documentation

Release 4.1.0

Top Free Games

Nov 08, 2018

Contents

Contents:

CHAPTER 1

Overview

What is Khan? Khan is an HTTP "resty" API for managing clans for games. It could be used to manage groups of people, but our aim is players in a game.

Khan allows your app to focus on the interaction required to creating clans and managing applications, instead of the backend required for actually doing it.

1.1 Features

- **Multi-tenant** - Khan already works for as many games as you need, just keep adding new games;
- **Clan Management** - Create and manage clans, their metadata as well as promote and demote people in their rosters;
- **Player Management** - Manage players and their metadata, as well as their applications to clans;
- **Applications** - Khan handles the work involved with applying to clans, inviting people to clans, accepting, denying and kicking;
- **Clan Search** - Search a list of clans to present your player with relevant options;
- **Top Clans** - Choose from a specific dimension to return a list of the top clans in that specific range (SOON);
- **Web Hooks** - Need to integrate your clan system with another application? We got your back! Use our web hooks sytem and plug into whatever events you need;
- **Auditing Trail** - Track every action coming from your games (SOON);
- **New Relic Support** - Natively support new relic with segments in each API route for easy detection of bottlenecks;
- **Easy to deploy** - Khan comes with containers already exported to docker hub for every single of our successful builds. Just pick your choice!

1.2 Architecture

Khan is based on the premise that you have a backend server for your game. That means we do not employ any means of authentication.

There's no validation if the actions you are performing are valid as well. We have TONS of validation around the operations themselves being valid.

What we don't have are validations that test whether the source of the request can perform the request (remember the authentication bit?).

Khan also offers a JSON `metadata` field in its Player and Clan models. This means that your game can store relevant information that can later be used to sort players, for example.

1.3 The Stack

For the devs out there, our code is in Go, but more specifically:

- Web Framework - Echo based on the insanely fast FastHTTP;
- Database - Postgres >= 9.5;
- Cache - Redis.

1.4 Who's Using it

Well, right now, only us at TFG Co, are using it, but it would be great to get a community around the project. Hope to hear from you guys soon!

1.5 How To Contribute?

Just the usual: Fork, Hack, Pull Request. Rinse and Repeat. Also don't forget to include tests and docs (we are very fond of both).

CHAPTER 2

Installing Khan

TBW.

CHAPTER 3

Hosting Khan

There are three ways to host Khan: docker, binaries or from source.

3.1 Docker

Running Khan with docker is rather simple. Our docker container image comes bundled with the API binary. All you need to do is load balance all the containers and you're good to go. The API runs at port `8080` in the docker image.

Khan uses PostgreSQL to store clans information. The container takes environment variables to specify this connection:

- `KHAN_POSTGRES_HOST` - PostgreSQL host to connect to;
- `KHAN_POSTGRES_PORT` - PostgreSQL port to connect to;
- `KHAN_POSTGRES_USER` - Password of the PostgreSQL Server to connect to;
- `KHAN_POSTGRES_DBNAME` - Database name of the PostgreSQL Server to connect to;
- `KHAN_POSTGRES_SSLMODE` - SSL Mode to connect to postgres with;

Other than that, there are a couple more configurations you can pass using environment variables:

- `KHAN_NEWRELIC_KEY` - If you have a New Relic account, you can use this variable to specify your API Key to populate data with New Relic API;
- `KHAN_SENTRY_URL` - If you have a sentry server you can use this variable to specify your project's URL to send errors to;
- `KHAN_EXTENSIONS_DOGSTATSD_HOST` - If you have a statsd datadog daemon, Podium will publish metrics to the given host at a certain port. Ex. localhost:8125;
- `KHAN_EXTENSIONS_DOGSTATSD_RATE` - If you have a statsd daemon, Podium will export metrics to the deamon at the given rate;
- `KHAN_EXTENSIONS_DOGSTATSD_TAGS_PREFIX` - If you have a statsd daemon, you may set a prefix to every tag sent to the daemon;

If you want to expose Khan outside your internal network it's advised to use Basic Authentication. You can specify basic authentication parameters with the following environment variables:

- `KHAN_BASICAUTH_USERNAME` - If you specify this key, Khan will be configured to use basic auth with this user;
- `KHAN_BASICAUTH_PASSWORD` - If you specify `BASICAUTH_USERNAME`, Khan will be configured to use basic auth with this password;

3.1.1 Example command for running with Docker

```
    $ docker pull tfgco/khan
    $ docker run -t --rm -e "KHAN_POSTGRES_HOST=<postgres host>" -e "KHAN_POSTGRES_
↪PORT=<postgres port>" -p 8080:80 tfgco/khan
```

In order to run Khan's workers using docker you just need to send the `KHAN_RUN_WORKER` environment variable as `true`.

3.1.2 Example command for running workers with Docker

```
    $ docker pull tfgco/khan
    $ docker run -t --rm -e "KHAN_POSTGRES_HOST=<postgres host>" -e "KHAN_POSTGRES_
↪PORT=<postgres port>" -e "KHAN_RUN_WORKERS=true" -p 9999:80 tfgco/khan
```

3.2 Binaries

Whenever we publish a new version of Khan, we'll always supply binaries for both Linux and Darwin, on i386 and x86_64 architectures. If you'd rather run your own servers instead of containers, just use the binaries that match your platform and architecture.

The API server is the `khan` binary. It takes a configuration yaml file that specifies the connection to PostgreSQL and some additional parameters. You can learn more about it at default.yaml.

The workers can be started using the same `khan` binary. It takes a configuration yaml file that specifies the connection to PostgreSQL and some additional parameters. You can learn more about it at default.yaml.

3.3 Source

Left as an exercise to the reader.

CHAPTER 4

Game Configuration

Being a multi-tenant clan server, Khan allows for many different configurations per tenant. Each tenant is a different game and is identified by it's game ID.

Before any clan operation can be performed, you must create a game in Khan. The good news here is that creating/updating games are idempotent operations. You can keep executing it any time your game changes. That's ideal to be executed in a deploy script, for instance.

4.1 Creating/Updating a Game

We recommend that the `Update` operation of the Game resource be used in detriment of the `Create` one. The reasoning here is that the Ùpdate‘ operation is idempotent (you can run it as many times as you want with the same result). If your game does not exist yet, it will create it, otherwise just updated it with the new configurations.

To Create/Update your game, just do a `PUT` request to `http://my-khan-server/games/my-game-public-id`, where `my-game-public-id` is the ID you'll be using for all your game's operations in the future. The payload for the request is a JSON object in the body and should be as follows:

```
{
  "name":                          [string],
  "metadata":                      [JSON],
  "membershipLevels":              [JSON],
  "minLevelToAcceptApplication":   [int],
  "minLevelToCreateInvitation":    [int],
  "minLevelToRemoveMember":        [int],
  "minLevelOffsetToPromoteMember": [int],
  "minLevelOffsetToRemoveMember":  [int],
  "minLevelOffsetToDemoteMember":  [int],
  "maxMembers":                    [int],
  "maxClansPerPlayer":             [int],
  "cooldownAfterDeny":             [int],
  "cooldownAfterDelete":           [int],
  "cooldownBeforeInvite":          [int],
```

(continues on next page)

(continued from previous page)

```
    "cooldownBeforeApply":          [int],
    "maxPendingInvites":            [int],
    "clanHookFieldsWhitelist":      [string],
    "playerHookFieldsWhitelist":    [string],
}
```

If the operation is successful, you'll receive a JSON object saying it succeeded:

```
{
  "success": true
}
```

4.2 Game Configuration Settings

As can be seen from the previous section, there are a lot of different configurations you can do per game. These will be thoroughly explained in this section.

4.2.1 name

The name of your game. This is used mainly for easier reasoning of what this game is when debugging.

Type: `string` **Sample Value**: `My Sample Game`

4.2.2 metadata

Metadata related to your clan. This is a JSON object and can store anything you need to. Each game will probably have a different usage for this attribute: clan nationality, clan flag image URL, number of victories for the clan to date, etc.

This value is a black box as far as Khan is concerned. It's not used to decide any rules for clan management.

Type: `JSON` **Sample Value**: `{ "country": "BR", "language": "pt-BR" }`

4.2.3 membershipLevels

The available membership levels the specified game supports. This is a way to specify the hierarchy between members in your game's clans. These levels are used in other configuration settings like `minLevelToRemoveMember` or `minLevelOffsetToPromoteMember`, among others.

The membership values (integer) should grow in importance, with the highest number being the highest member level.

Type: `JSON` **Sample Value**: `{ "member": 1, "leader": 2, "owner": 3 }`

4.2.4 minLevelToAcceptApplication

The minimum member level (as specified in the membershipLevels configuration) required to accept a pending application to the clan.

Type: `integer` **Sample Value**: `2`

4.2.5 minLevelToCreateInvitation

The minimum member level (as specified in the membershipLevels configuration) required to invite someone into a clan.

Type: `integer` **Sample Value**: `2`

4.2.6 minLevelToRemoveMember

The minimum member level (as specified in the membershipLevels configuration) required to remove someone from the clan.

Type: `integer` **Sample Value**: `2`

4.2.7 minLevelOffsetToRemoveMember

This configuration specifies the required difference in level between a player and the player being removed from the clan.

Let's look at an example to make things easier:

```
John has a membership level of 3, Paul has a membership level of 2 and
Ted has a membership level of 1.

If the clan has a minLevelOffsetToRemoveMember of 2, that means that only
John can remove Ted, but if that configuration is 1, then both John and Paul
can remove Ted.
```

Type: `integer` **Sample Value**: `2`

4.2.8 minLevelOffsetToPromoteMember

This configuration specifies the required difference in level between a player and the player being promoted (calculated BEFORE the promotion).

What this means is that a player can only promote another player if that player is `minLevelOffsetToPromoteMember` levels below their own level before the promotion takes place.

If the `minLevelOffsetToPromoteMember` is greater than 1, then only the clan owner can promote someone to the highest available level(s).

Let's look at an example to make things easier:

```
John has a membership level of 5, Paul has a membership level of 3 and
Ted has a membership level of 1.

If the clan has a minLevelOffsetToPromoteMember of 2, that means that only
John can promote Ted up to level 4, but if that configuration is 1,
then both John and Paul can promote Ted (Paul can promote Ted to level 3).
```

Type: `integer` **Sample Value**: `2`

4.2.9 minLevelOffsetToDemoteMember

This configuration specifies the required difference in level between a player and the player being demoted (calculated BEFORE the demotion).

If the `minLevelOffsetToDemoteMember` is greater than 1, then only the clan owner can demote someone from the highest available level(s).

Let's look at an example to make things easier:

```
John has a membership level of 5, Paul has a membership level of 4 and
Ted has a membership level of 3.

If the clan has a minLevelOffsetToDemoteMember of 2, that means that only
John can demote Ted, but if that configuration is 1, then both John and
Paul can demote Ted.
```

Type: `integer` **Sample Value**: `2`

4.2.10 maxMembers

This configuration specifies the maximum number of members a clan can have.

Type: `integer` **Sample Value**: `50`

4.2.11 maxClansPerPlayer

This configuration specifies the maximum number of clans a player can be a member of.

Type: `integer` **Sample Value**: `1`

4.2.12 cooldownAfterDeny

Time (in seconds) the player must wait before applying/being invited to a new membership after the last membership application/invite was denied.

Type: `integer` **Sample Value**: `360`

4.2.13 cooldownAfterDelete

Time (in seconds) the player must wait before applying/being invited to a new membership after the last membership application/invite was deleted.

Type: `integer` **Sample Value**: `720`

4.2.14 cooldownBeforeInvite

Time (in seconds) a clan member must wait before inviting a member to a new membership after the last membership application/invite was created.

Type: `integer` **Sample Value**: `720`

4.2.15 cooldownBeforeApply

Time (in seconds) a player must wait before applying for a clan after the last membership application/invite was created.

Type: `integer` **Sample Value**: `480`

4.2.16 maxPendingInvites

Maximum number of pending invites each player can have withstanding. Set this value to `-1` if your game has no limits on maximum pending invites.

Type: `integer` **Sample Value**: `20`

4.2.17 clanHookFieldsWhitelist

A comma-separated-values list of properties in the clan's metadata that will trigger the Clan Updated hook upon change.

Type: `string` **Sample Value**: `trophies,country`

4.2.18 playerHookFieldsWhitelist

A comma-separated-values list of properties in the player's metadata that will trigger the Player Updated hook upon change.

Type: `string` **Sample Value**: `trophies,country`

CHAPTER 5

Using Web Hooks

You can use khan's web hooks to send detailed event information to other servers. The use cases for this are plenty, since the need for integrating micro-services is very common.

One possible use case would be to notify the chat channel for your game with information about people joining/leaving the clan or for new applications.

5.1 Webhook Workers

Khan uses GoWorkers workers to send webhooks. This means that you need to run the `khan worker` command to start as many workers as you want.

5.2 Webhook Specific Configuration

Khan has some configuration entries specific to webhooks:

- `redis.host` - Redis server host used for GoWorkers;
- `redis.port` - Redis server port used for GoWorkers;
- `redis.database` - Redis server database used for GoWorkers;
- `redis.pool` - Redis connection pool size used for GoWorkers;
- `redis.password` - Redis password used for GoWorkers;
- `webhooks.timeout` - Timeout for webhook HTTP connections;
- `webhooks.workers` - Number of GoWorkers to start with each instance of Khan worker;
- `webhooks.runStats` - Will the GoWorkers stats server run in each Khan worker instance?;
- `webhooks.statsPort` - Port that the stats server of GoWorkers will run in.

5.3 Registering a Web Hook

Registering a web hook is done using the Create Web Hook Route. A hook can also be removed using the Remove Web Hook Route. Just make sure you keep the PublicID that was returned by the Create Hook route as it is required to remove a hook.

5.4 How do Web Hooks work?

When an event happen in Khan, it will look for all the hooks registered for the game that the event happened in.

It will then do a POST request to all of them with the payload relevant to the event (more about the payloads in the Events section). The payload for the events is just a JSON object in the body of the request.

Let's say we want to keep track of all the clans that are created in our game. We could create a web hook for the `ClanCreated` event and Khan would call our hook with this payload:

```
{
  gameID: "my-game",
  publicID: "clan-public-id",
  name: "My Fancy Clan"
}
```

We could then use this information to store this clan in our Database, to integrate with a chat channel, to provision some third-party system for clans, etc.

5.5 URL Format and Flexibility

When registering a new URL, Khan allows you to specify the URL as a Template.

Let's assume you need to use the clan public ID in your URL to store a newly created clan at `http://my-system.com:3030/clans/some-clan/`. Instead of some-clan, we need to use the clan's public ID.

This is very easy to do with Khan. You can interpolate any of the keys in the payload using `{{key}}`. Bear in mind that only two types of keys can be used:

- top-level keys - if you use a key that's in the first level of depth in the payload, you are good with any type of key;
- dot separated keys - this type of key will only work if all the keys in the path are objects (except the last one).

Let's try with the payload for the player created event:

```
    {
        "success": true,
        "gameID":  "some-game",
        "publicID": "playerPublicID",
        "name": "Player Name",
        "metadata": {
            "score": 1200,
            "league": {
                ranking: "diamond",
                position: 30
            }
        }
    }
```

Now imagine we want the player to be included in the league he belongs to. We could use an URL like `http://my-server.com:3030/players/{{publicID}}/leagues/{{metadata.league.ranking}}/`. This would be translated by Khan to `http://my-server.com:3030/players/playerPublicID/leagues/diamond/`.

5.6 Event Types

So what types of events can you create Web Hooks for?

5.6.1 Game Hooks

Game Updated

Event Type: `0`

Payload:

```
{
    "success": true,
    "type": 0,                                  // Event Type
    "publicID": [string],                       // Game ID
    "name": [string],                           // Game Name
    "metadata": [JSON],                         // JSON Object containing game␣
↪metadata.
    "membershipLevels": [JSON],                 // JSON Object mapping membership␣
↪levels
    "minLevelToAcceptApplication": [int],       // Minimum level of membership␣
↪required
                                                // to accept players into clan
    "minLevelToCreateInvitation": [int],        // Minimum level of membership␣
↪required
                                                // to invite players into clan
    "minLevelToRemoveMember": [int],            // Minimum level of membership␣
↪required
                                                // to remove players from clan
    "minLevelOffsetToRemoveMember": [int],      // A player must be at least this␣
↪offset
                                                // higher than the player being␣
↪removed.
    "minLevelOffsetToPromoteMember": [int],     // A player must be at least this␣
↪offset
                                                // higher than the player being␣
↪promoted
    "minLevelOffsetToDemoteMember": [int],      // A player must be at least this␣
↪offset
                                                // higher than the player being␣
↪demoted
    "maxMembers": [int],                        // Maximum number of players in the␣
↪clan
    "maxClansPerPlayer": [int]                  // Maximum number of clans the player␣
↪can be
                                                // member of
}
```

5.6.2 Player Hooks

Player Created

Event Type: 1

Payload:

```
{
    "gameID":  [string],                        // Game ID
    "type": 1,                                  // Event Type
    "publicID": [string],                       // Created Player PublicID This id␣
↪should
                                                // be used when referring to the␣
↪player in
                                                // future operations.
    "name": [string],                           // Player Name
    "metadata": [JSON],                         // JSON Object containing player␣
↪metadata
    "membershipCount": [int],                   // Number of clans this player is a␣
↪member of
    "ownershipCount":  [int],                   // Number of clans this player is an␣
↪owner of
    "id": [UUID],                               // unique id that identifies the hook
    "timestamp": [timestamp]                    // timestamp in the RFC3339 format
}
```

Player Updated

Event Type: 2

Payload:

```
{
    "gameID":  [string],                        // Game ID
    "type": 2,                                  // Event Type
    "publicID": [string],                       // Created Player PublicID This id␣
↪should
                                                // be used when referring to the␣
↪player in
                                                // future operations.
    "name": [string],                           // Player Name
    "metadata": [JSON],                         // JSON Object containing player␣
↪metadata
    "membershipCount": [int],                   // Number of clans this player is a␣
↪member of
    "ownershipCount":  [int],                   // Number of clans this player is an␣
↪owner of
    "id": [UUID],                               // unique id that identifies the hook
    "timestamp": [timestamp]                    // timestamp in the RFC3339 format
}
```

5.6.3 Clan Hooks

Clan Created

Event Type: 3

Payload:

```
{
    "gameID":  [string],                          // Game ID
    "type": 3,                                    // Event Type
    "clan": {
      "publicID": [string],                       // Created Clan PublicID This id␣
↪should
                                                  // be used when referring to the clan␣
↪in
                                                  // future operations.
      "name": [string],                           // Clan Name
      "metadata": [JSON],                         // JSON Object containing clan's␣
↪metadata
      "allowApplication": [bool],                 // Indicates whether this clan acceps␣
↪applications
      "autoJoin": [bool]                          // Indicates whether this clan␣
↪automatically
                                                  // accepts applications
    }
    "id": [UUID],                                 // unique id that identifies the hook
    "timestamp": [timestamp]                      // timestamp in the RFC3339 format
}
```

Clan Updated

Event Type: 4

Payload:

```
{
    "gameID":  [string],                          // Game ID
    "type": 4,                                    // Event Type
    "clan": {
      "publicID": [string],                       // Updated Clan PublicID This id␣
↪should
                                                  // be used when referring to the clan␣
↪in
                                                  // future operations.
      "name": [string],                           // Clan Name
      "metadata": [JSON],                         // JSON Object containing clan's␣
↪metadata
      "allowApplication": [bool],                 // Indicates whether this clan acceps␣
↪applications
      "autoJoin": [bool]                          // Indicates whether this clan␣
↪automatically
                                                  // accepts applications
    }
    "id": [UUID],                                 // unique id that identifies the hook
    "timestamp": [timestamp]                      // timestamp in the RFC3339 format
}
```

WARNING: This event may be skipped if the game's `clanHookFieldsWhitelist` field is not empty and none of the fields that actually changed is in the whitelist. Imagine you have a document like this:

```
{
    "trophies": 30,
    "country": "US",
}
```

The game is configured with `clanHookFieldsWhitelist = "country"`. That means that changing the number of trophies won't dispatch hooks, but changing the country (or any other clan property) will.

Clan Owner Left

Event Type: `5`

Payload:

```
{
    "gameID": [string],
    "type": 5,                                  // Event Type
    "isDeleted": [bool],                        //Indicates whether the clan was␣
↪deleted
                                                //because there were no members␣
↪left
    "clan": {
        "publicID": [string],                   // Updated Clan PublicID
        "name": [string],                       // Clan Name
        "metadata": [JSON],                     // JSON Object containing clan's␣
↪metadata
        "allowApplication": [bool]              // Indicates whether this clan␣
↪acceps applications
        "autoJoin": [bool],                     // Indicates whether this clan␣
↪automatically
                                                // accepts applications
        "membershipCount":  [int],              // Number of members in clan
    },
    "previousOwner": {                          // The owner that left
        "publicID": [string],                   // Previous Owner PublicID
        "name": [string],                       // Player Name
        "metadata": [JSON],                     // JSON Object containing player␣
↪metadata
        "membershipCount": [int],               // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                // Number of clans this player is␣
↪an owner of
    },
    "newOwner": {                               // After the owner left, this is␣
↪the new owner
        "publicID": [string],                   // New Owner PublicID
        "name": [string],                       // Player Name
        "metadata": [JSON],                     // JSON Object containing player␣
↪metadata
        "membershipCount": [int],               // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                // Number of clans this player is␣
↪an owner of
    },
```

(continues on next page)

(continued from previous page)

```
    "id": [UUID],                                   // unique id that identifies the␣
↪hook
    "timestamp": [timestamp]                        // timestamp in the RFC3339 format
}
```

Clan Ownership Transferred to Another Player

Event Type: `6`

Payload:

```
{
    "gameID": [string],
    "type": 6,                                  // Event Type
    "clan": {
        "publicID": [string],                       // Updated Clan PublicID
        "name": [string],                           // Clan Name
        "metadata": [JSON],                         // JSON Object containing clan's␣
↪metadata
        "allowApplication": [bool]                  // Indicates whether this clan␣
↪acceps applications
        "autoJoin": [bool],                         // Indicates whether this clan␣
↪automatically
                                                    // accepts applications
        "membershipCount":  [int],                  // Number of members in clan
    },
    "previousOwner": {                                  // The previous owner
        "publicID": [string],                       // Previous Owner PublicID
        "name": [string],                           // Player Name
        "metadata": [JSON],                         // JSON Object containing player␣
↪metadata
        "membershipCount": [int],                   // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                    // Number of clans this player is␣
↪an owner of
    },
    "newOwner": {                                   // Player that the owner␣
↪transferred ownership to
        "publicID": [string],                       // New Owner PublicID
        "name": [string],                           // Player Name
        "metadata": [JSON],                         // JSON Object containing player␣
↪metadata
        "membershipCount": [int],                   // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                    // Number of clans this player is␣
↪an owner of
    },
    "id": [UUID],                                   // unique id that identifies the␣
↪hook
    "timestamp": [timestamp]                        // timestamp in the RFC3339 format
}
```

5.6.4 Membership Hooks

Membership Created

This event occurs if a player applies to a clan (player == requestor) or if a player is invited to a clan (player != requestor).

Event Type: 7

Payload:

```
{
    "gameID": [string],
    "type": 7,                                  // Event Type
    "clan": {
        "publicID": [string],                   // Clan that player is applying to
        "name": [string],                       // Clan Name
        "metadata": [JSON],                     // JSON Object containing clan's␣
↪metadata
        "allowApplication": [bool]              // Indicates whether this clan␣
↪acceps applications
        "autoJoin": [bool],                     // Indicates whether this clan␣
↪automatically
                                                // accepts applications
        "membershipCount":  [int],              // Number of members in clan
    },
    "player": {                                 // Player that is applying/being␣
↪invited to the clan
        "publicID": [string],                   // Applicant PublicID
        "name": [string],                       // Player Name
        "metadata": [JSON],                     // JSON Object containing player␣
↪metadata
        "membershipCount": [int],               // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int],               // Number of clans this player is␣
↪an owner of
        "membershipLevel":  [string]            // The level of the player's␣
↪membership
    },
    "requestor": {                              // Player that requested this␣
↪membership application/invite
        "publicID": [string],                   // Requestor PublicID
        "name": [string],                       // Player Name
        "metadata": [JSON],                     // JSON Object containing player␣
↪metadata
        "membershipCount": [int],               // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                // Number of clans this player is␣
↪an owner of
    },
    "id": [UUID],                               // unique id that identifies the␣
↪hook
    "timestamp": [timestamp]                    // timestamp in the RFC3339 format
}
```

Membership Approved

This event occurs if an application or an invite to a clan gets approved. If the membership that was approved was an invitation into the clan, the requestor and the player will be the same Player. Otherwise, the requestor will be whomever approved the membership.

Event Type: `8`

Payload:

```
{
    "gameID": [string],
    "type": 8,                                  // Event Type
    "clan": {
        "publicID": [string],                   // Clan that membership was␣
↪approved
        "name": [string],                       // Clan Name
        "metadata": [JSON],                     // JSON Object containing clan's␣
↪metadata
        "allowApplication": [bool]              // Indicates whether this clan␣
↪acceps applications
        "autoJoin": [bool],                     // Indicates whether this clan␣
↪automatically
                                                // accepts applications
        "membershipCount":  [int],              // Number of members in clan
    },
    "player": {                                 // Player that was approved into␣
↪the clan
        "publicID": [string],                   // Player PublicID
        "name": [string],                       // Player Name
        "metadata": [JSON],                     // JSON Object containing player␣
↪metadata
        "membershipCount": [int],               // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int],               // Number of clans this player is␣
↪an owner of
        "membershipLevel":  [string]            // The level of the player's␣
↪membership
    },
    "requestor": {                              // Player that approved the␣
↪membership
        "publicID": [string],                   // Requestor PublicID
        "name": [string],                       // Player Name
        "metadata": [JSON],                     // JSON Object containing player␣
↪metadata
        "membershipCount": [int],               // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                // Number of clans this player is␣
↪an owner of
    },
    "creator": {                                // Player that created the␣
↪membership (invited or applied)
        "publicID": [string],                   // Creator PublicID
        "name": [string],                       // Creator Name
        "metadata": [JSON],                     // JSON Object containing creator␣
↪metadata
        "membershipCount": [int],               // Number of clans this creator␣
↪is a member of
```

(continues on next page)

(continued from previous page)

```
        "ownershipCount":  [int]                    // Number of clans this creator␣
↪is an owner of
    },
    "id": [UUID],                                   // unique id that identifies the␣
↪hook
    "timestamp": [timestamp]                        // timestamp in the RFC3339 format
}
```

Membership Denied

This event occurs if an application or an invite to a clan gets denied. If the membership that was denied was an invitation into the clan, the requestor and the player will be the same Player. Otherwise, the requestor will be whomever denied the membership.

Event Type: 9

Payload:

```
{
    "gameID": [string],
    "type": 9,                                  // Event Type
    "clan": {
        "publicID": [string],                       // Clan that membership was denied
        "name": [string],                           // Clan Name
        "metadata": [JSON],                         // JSON Object containing clan's␣
↪metadata
        "allowApplication": [bool]                  // Indicates whether this clan␣
↪acceps applications
        "autoJoin": [bool],                         // Indicates whether this clan␣
↪automatically
                                                    // accepts applications
        "membershipCount":  [int],                  // Number of members in clan
    },
    "player": {                                     // Player that was denied into␣
↪the clan
        "publicID": [string],                       // Player PublicID
        "name": [string],                           // Player Name
        "metadata": [JSON],                         // JSON Object containing player␣
↪metadata
        "membershipCount": [int],                   // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int],                   // Number of clans this player is␣
↪an owner of
        "membershipLevel":  [string]                // The level of the player's␣
↪membership
    },
    "requestor": {                                  // Player that denied the␣
↪membership
        "publicID": [string],                       // Requestor PublicID
        "name": [string],                           // Player Name
        "metadata": [JSON],                         // JSON Object containing player␣
↪metadata
        "membershipCount": [int],                   // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                    // Number of clans this player is␣
↪an owner of
```

(continues on next page)

(continued from previous page)

```
    },
    "creator": {                                    // Player that created the␣
↪membership (invited or applied)
        "publicID": [string],                       // Creator PublicID
        "name": [string],                           // Creator Name
        "metadata": [JSON],                         // JSON Object containing creator␣
↪metadata
        "membershipCount": [int],                   // Number of clans this creator␣
↪is a member of
        "ownershipCount":  [int]                    // Number of clans this creator␣
↪is an owner of
    },
    "id": [UUID],                                   // unique id that identifies the␣
↪hook
    "timestamp": [timestamp]                        // timestamp in the RFC3339 format
}
```

Member Promoted

Event Type: `10`

Payload:

```
{
    "gameID": [string],
    "type": 10,                                 // Event Type
    "clan": {
        "publicID": [string],                   // Clan that member was promoted
        "name": [string],                       // Clan Name
        "metadata": [JSON],                     // JSON Object containing clan's␣
↪metadata
        "allowApplication": [bool]              // Indicates whether this clan␣
↪acceps applications
        "autoJoin": [bool],                     // Indicates whether this clan␣
↪automatically
                                                // accepts applications
        "membershipCount":  [int],              // Number of members in clan
    },
    "player": {                                 // Player that was promoted
        "publicID": [string],                   // Player PublicID
        "name": [string],                       // Player Name
        "metadata": [JSON],                     // JSON Object containing player␣
↪metadata
        "membershipCount": [int],               // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int],               // Number of clans this player is␣
↪an owner of
        "membershipLevel":  [string]            // The new level of the player's␣
↪membership
    },
    "requestor": {                              // Player that promoted this␣
↪member
        "publicID": [string],                   // Requestor PublicID
        "name": [string],                       // Player Name
        "metadata": [JSON],                     // JSON Object containing player␣
↪metadata
```

(continues on next page)

(continued from previous page)

```
        "membershipCount": [int],                    // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                     // Number of clans this player is␣
↪an owner of
    },
    "id": [UUID],                                    // unique id that identifies the␣
↪hook
    "timestamp": [timestamp]                         // timestamp in the RFC3339 format
}
```

Member Demoted

Event Type: `11`

Payload:

```
{
    "gameID": [string],
    "type": 11,                                  // Event Type
    "clan": {
        "publicID": [string],                    // Clan that member was demoted
        "name": [string],                        // Clan Name
        "metadata": [JSON],                      // JSON Object containing clan's␣
↪metadata
        "allowApplication": [bool]               // Indicates whether this clan␣
↪acceps applications
        "autoJoin": [bool],                      // Indicates whether this clan␣
↪automatically
                                                 // accepts applications
        "membershipCount":  [int],               // Number of members in clan
    },
    "player": {                                  // Player that was demoted
        "publicID": [string],                    // Player PublicID
        "name": [string],                        // Player Name
        "metadata": [JSON],                      // JSON Object containing player␣
↪metadata
        "membershipCount": [int],                // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                 // Number of clans this player is␣
↪an owner of,
        "membershipLevel":  [string]             // The new level of the player's␣
↪membership
    },
    "requestor": {                               // Player that demoted this member
        "publicID": [string],                    // Requestor PublicID
        "name": [string],                        // Player Name
        "metadata": [JSON],                      // JSON Object containing player␣
↪metadata
        "membershipCount": [int],                // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                 // Number of clans this player is␣
↪an owner of
    },
    "id": [UUID],                                // unique id that identifies the␣
↪hook
```

(continues on next page)

(continued from previous page)

```
    "timestamp": [timestamp]                        // timestamp in the RFC3339 format
}
```

Member Left

Event Type: `12`

Payload:

```
{
    "gameID": [string],
    "type": 12,                                     // Event Type
    "clan": {
        "publicID": [string],                       // Clan that member left
        "name": [string],                           // Clan Name
        "metadata": [JSON],                         // JSON Object containing clan's␣
↪metadata
        "allowApplication": [bool]                  // Indicates whether this clan␣
↪acceps applications
        "autoJoin": [bool],                         // Indicates whether this clan␣
↪automatically
                                                    // accepts applications
        "membershipCount":  [int],                  // Number of members in clan
    },
    "player": {                                     // Player that left
        "publicID": [string],                       // Player PublicID
        "name": [string],                           // Player Name
        "metadata": [JSON],                         // JSON Object containing player␣
↪metadata
        "membershipCount": [int],                   // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int],                   // Number of clans this player is␣
↪an owner of
        "membershipLevel":  [string]                // The level of the player's␣
↪membership
    },
    "requestor": {                                  // Player that removed leaving␣
↪player (if they left
                                                    // on their own, then this is the␣
↪same as player)
        "publicID": [string],                       // Requestor PublicID
        "name": [string],                           // Player Name
        "metadata": [JSON],                         // JSON Object containing player␣
↪metadata
        "membershipCount": [int],                   // Number of clans this player is␣
↪a member of
        "ownershipCount":  [int]                    // Number of clans this player is␣
↪an owner of
    },
    "id": [UUID],                                   // unique id that identifies the␣
↪hook
    "timestamp": [timestamp]                        // timestamp in the RFC3339 format
}
```

CHAPTER 6

Khan API

6.1 Healthcheck Routes

6.1.1 Healthcheck

`GET /healthcheck`

Validates that the app is still up, including the database connection.

- Success Response
  - Code: `200`
  - Content:

    ```
    "WORKING"
    ```

  - Headers:

    It will add an `KHAN-VERSION` header with the current khan module version.
- Error Response

  It will return an error if it failed to connect to the database.
  - Code: `500`
  - Content:

    ```
    "Error connecting to database: <error-details>"
    ```

6.2 Status Routes

6.2.1 Status

`GET /status`

Returns statistics on the health of khan.

- Success Response
  - Code: `200`
  - Content:

```
  {
    "app": {
      "errorRate": [float]        // Exponentially Weighted Moving Average␣
↪Error Rate
    },
    "dispatch": {
      "pendingJobs": [int]        // Pending hook jobs to be sent
    }
  }
```

6.3 Game Routes

6.3.1 Create Game

`POST /games`

Creates a new game with the given parameters.

- Payload

```
{
  "publicID":                     [string],  // 36 characters max, must be unique
  "name":                         [string],  // 2000 characters max
  "metadata":                     [JSON],
  "membershipLevels":             [JSON],
  "minLevelToAcceptApplication":  [int],
  "minLevelToCreateInvitation":   [int],
  "minLevelToRemoveMember":       [int],
  "minLevelOffsetToRemoveMember": [int],
  "minLevelOffsetToPromoteMember":[int],
  "minLevelOffsetToDemoteMember": [int],
  "maxMembers":                   [int],
  "maxClansPerPlayer":            [int],
  "cooldownAfterDeny":            [int],
  "cooldownAfterDelete":          [int],
  "cooldownBeforeInvite":         [int],
  "cooldownBeforeApply":          [int],
  "maxPendingInvites":            [int],
  "clanHookFieldsWhitelist":      [string],
  "playerHookFieldsWhitelist":    [string],
}
```

Metadata is intended to include all game specific configuration that is not directly related to the khan's clan management. For example, clank ranking, clan trophies, clan description, etc.

- Some parameters require special attention:

  membershipLevels: Should be a JSON mapping levels (strings) to integers:

```
{
  "Member": 1,
  "Elder": 2,
  "CoLeader": 3
}
```

  The integer part of the level will be used to compare with `minLevel...` and `minLevelOffset...` when performing membership operations.

  minLevelToAcceptApplication: A member cannot accept a player's application to join the clan unless their level is greater or equal to this parameter.

  minLevelToCreateInvitation: A member cannot invite a player to join the clan unless their level is greater or equal to this parameter.

  minLevelOffsetToRemoveMember: A member cannot remove another member unless their level is at least `MinLevelOffsetToRemoveMember` levels greater than the level of the member they wish to promote.

  minLevelOffsetToPromoteMember: A member cannot promote another member unless their level is at least `minLevelOffsetToPromoteMember` levels greater than the level of the member they wish to promote.

  minLevelOffsetToDemoteMember: A member cannot demote another member unless their level is at least `minLevelOffsetToDemoteMember` levels greater than the level of the member they wish to demote.

  maxMembers: Maximum number of members a clan of this game can have.

  maxClansPerPlayer: Maximum number of clans a player can be member of.

  cooldownAfterDeny: Time (in seconds) the player must wait before applying/being invited to a new membership after the last membership application/invite was denied.

  cooldownAfterDelete: Time (in seconds) the player must wait before applying/being invited to a new membership after the last membership application/invite was deleted.

  cooldownBeforeInvite: Time (in seconds) a clan member must wait before inviting a member to a new membership after the last membership application/invite was created.

  cooldownBeforeApply: Time (in seconds) a player must wait before applying for a clan after the last membership application/invite was created.

  maxPendingInvites: Maximum number of pending invites each player can have withstanding. Set this value to -1 if your game has no limits on maximum pending invites.

  clanHookFieldsWhitelist: If you change metadata very frequently in clans, you can specify here the fields in your metadata document for which you'd like to have the clan updated hook triggered. If no fields are specified, the hook will be triggered in all updates. If you don't want any metadata changes to trigger hooks, just set this to "none" or any key that does not exist in your metadata document.

  playerHookFieldsWhitelist: If you change metadata very frequently in players, you can specify here the fields in your metadata document for which you'd like to have the player updated hook triggered. If no fields are specified, the hook will be triggered in all updates. If you don't want any metadata changes to trigger hooks, just set this to "none" or any key that does not exist in your metadata document.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "publicID": [string]  // game public id
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  It will return an error if there are invalid parameters.

  - Code: `422`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.3.2 Update Game

`PUT /games/:gameID`

Updates the game with that has publicID `gameID`.

- Payload

```
{
  "name":                          [string],  // 2000 characters max
  "metadata":                      [JSON],
  "membershipLevels":              [JSON],
  "minLevelToAcceptApplication":   [int],
  "minLevelToCreateInvitation":    [int],
  "minLevelToRemoveMember":        [int],
  "minLevelOffsetToPromoteMember": [int],
```

(continues on next page)

(continued from previous page)

```
  "minLevelOffsetToDemoteMember":  [int],
  "maxMembers":                    [int],
  "maxClansPerPlayer":             [int],
  "cooldownAfterDeny":             [int],
  "cooldownAfterDelete":           [int],
  "cooldownBeforeInvite":          [int],
  "cooldownBeforeApply":           [int],
  "maxPendingInvites":             [int],
  "clanHookFieldsWhitelist":       [string],
  "playerHookFieldsWhitelist":     [string]
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  It will return an error if there are invalid parameters.

  - Code: `422`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.4 Hook Routes

More about web hooks can be found in Using WebHooks.

6.4.1 Supported Web Hook Event Types

- `0 Game Updated` - Happens when a game is updated;
- `1 Player Created` - Happens when a new player is created;
- `2 Player Updated` - Happens when an existing player is updated;
- `3 Clan Created` - Happens when a new clan is created;
- `4 Clan Updated` - Happens when an existing clan is updated;
- `5 Clan Owner Left` - Happens when the owner of a clan leaves the clan without transferring the ownership to someone else;
- `6 Ownership of the Clan transferred` - Happens when the owner of a clan transfers ownership to another player;
- `7 Membership Created` - Happens when either someone applies to a clan or is invited;
- `8 Membership Approved` - Happens when a pending membership to a clan is approved;
- `9 Membership Denied` - Happens when a pending membership to a clan is denied;
- `10 Member Promoted` - Happens when a member of the clan is promoted;
- `11 Member Demoted` - Happens when a pending member of the clan is demoted;
- `12 Member Left` - Happens when a member of the clan is either removed or leaves the clan.

6.4.2 Create Hook

`POST /games/:gameID/hooks`

Creates a new web hook for the specified game when the specified event type happens.

- Payload

```
{
  "type": [int],              // Event Type
  "hookURL": [string]         // the URL to call with the payload
                              // for the specified event.
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
  "publicID": [uuid]        // This is the id required to remove the hook.
                            // It should be stored with the client app.
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

- Code: `500`
- Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.4.3 Remove Hook

`DELETE /games/:gameID/hooks/:hookPublicID`

Removes a web hook created with the Create Hook route. No payload is required for this route.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
}
```

- Error Response
  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.5 Player Routes

6.5.1 Create Player

`POST /games/:gameID/players`

Creates a new player for the game with publicID=`gameID` with the given parameters.

- Payload

```
{
  "publicID":                          [string],   // 255 characters max, must be␣
↪unique for a given game
  "name":                              [string],   // 2000 characters max
```

(continues on next page)

(continued from previous page)

```
  "metadata":                          [JSON]
}
```

Metadata is intended to include all the player's game specific informations that are not directly related to the khan's clan management. For example, player ranking, player trophies, player level, etc.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "publicID": [string]  // player public id
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  It will return an error if there are invalid parameters.

  - Code: `422`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.5.2 Update Player

`PUT /games/:gameID/players/:playerPublicID`

Updates the player with the given publicID.

- Payload

```
{
  "name":                               [string],  // 2000 characters max
  "metadata":                           [JSON]
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  It will return an error if there are invalid parameters.
  - Code: `422`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.5.3 Retrieve Player

`GET /games/:gameID/players/:playerPublicID`

Gets the player with the given publicID.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "publicID": [string], // Player publicID
  "name": [string], // Player Name
  "metadata": [JSON], // Player Metadata
  "createdAt": [int64], // timestamp in milliseconds of when the player was␣
↪created
  "updatedAt": [int64]  // timestamp in milliseconds of when the player was␣
↪last updated

  //All clans the player is involved with show here
  "clans":{
    // Clans the player ownership
    "owned":[
      { "name": [string], "publicID": [string] }, // clan name and publicID
    ],

    // Clans the player has been approved and is currently a member of
    "approved":[
      { "name": [string], "publicID": [string] }, // clan name and publicID
    ],

    // Clans the player has been banned from
    "banned":[
      { "name": [string], "publicID": [string] }, // clan name and publicID
    ],

    // Clans the player has been rejected from
    "denied":[
      { "name": [string], "publicID": [string] }, // clan name and publicID
    ],

    // Clans the player has pending applications to
    "pendingApplications":[
      { "name": [string], "publicID": [string] }, // clan name and publicID
    ],

    // Clans the player has pending invites to
    "pendingInvites":[
      { "name": [string], "publicID": [string] }, // clan name and publicID
    ]
  },

  // All memberships this player has with details
  "memberships":[
    {
      //if approved, denied and banned are false, the membership is pending␣
↪approval
      "approved": [bool],
      "denied": [bool],
      "banned": [bool],

      //clan the player applied to
      "clan":{
        "metadata": [JSON],
        "name": [string],
        "publicID": [string],
```

(continues on next page)

(continued from previous page)

```
        "membershipCount": [int]
      },
      "createdAt":  [int64], // timestamp the player applied to a clan
      "updatedAt":  [int64], // timestamp that the membership was last updated
      "deletedAt":  [int64], // timestamp that the player was banned
      "approvedAt": [int64], // timestamp that the player was approved
      "deniedAt":   [int64], // timestamp that the player was denied

      "level": [string],    // level of the player in this clan

      "message": [string], // empty if membership not created using an␣
↪application
                           // or the application message otherwise

      // Player that requested membership
      // If the player was invited, this should be another player.
      // Otherwise, this is the same as the player.
      // If the membership level is 'owner' this key does not exist.
      "requestor":{
        "publicID": [string]
        "name": [string],
        "metadata": [JSON],
      },

      // Player that approved this membership
      // If the membership is not yet approved or the level is 'owner' this␣
↪key does not exist
      "approver":{
        "publicID": [string]
        "name": [string],
        "metadata": [JSON],
      },

      // Player that denied this membership
      // If the membership is not yet denied
      "denier":{
        "publicID": [string]
        "name": [string],
        "metadata": [JSON],
      },

    }
  ]
}
```

- Error Response
  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.6 Clan Routes

6.6.1 Create Clan

`POST /games/:gameID/clans`

Creates a new clan for the game with publicID=`gameID` with the given parameters.

- Payload

```
{
  "publicID":                    [string],  // 255 characters max, must be␣
↪unique for a given game
  "name":                        [string],  // 2000 characters max
  "metadata":                    [JSON],
  "ownerPublicID":               [string],  // must reference an existing player
  "allowApplication":            [boolean],
  "autoJoin":                    [boolean]
}
```

  Metadata is intended to include all the clan's game specific informations that are not directly related to the khan's clan management.

  - Some parameters require special attention:

    allowApplication: if set to false only the clan owner and members can invite players to join the clan, otherwise players can request to be added to a given clan.

    autoJoin: if set to true, when a player applies their membership is automatically approved. If set to false, the clan owner or one of its members must approve or deny the player's application.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "publicID": [string]  // clan public id
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.6.2 Update Clan

`PUT /games/:gameID/clans/:clanPublicID`

Updates the clan with the given publicID.

- Payload

```
{
  "name":                          [string],  // 2000 characters max
  "metadata":                      [JSON],
  "ownerPublicID":                 [string],  // must match the clan owner's␣
↪public id
  "allowApplication":              [boolean],
  "autoJoin":                      [boolean]
}
```

  All parameters but the `ownerPublicID` will be updated.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.6.3 Retrieve Clan

`GET /games/:gameID/clans/:clanPublicID`

Optional query string argument: shortID, if `true` then the first 8 characters of clanID are accepted.

Retrieves the clan with the given publicID. It will list all the clan information and its members.

The roster, as well as the memberships return a list of players, following this structure:

```
{
    "level": [string],  // not returned for denied/banned memberships
    "message": [string], // the message sent with the application
                         // or "" if the membership was not created with and␣
↪application
    "player": {
      "publicID": [string],
      "name":     [string],
      "metadata": [JSON],
      "approver": {  // player that approved this membership
        "publicID": [string],
        "name":     [string],
      }
    }
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "publicID": [string],
  "success": true,
  "name": [string],
  "metadata": [JSON],
  "allowApplication": [bool],
  "autoJoin": [bool],
  "membershipCount": [int],
  "owner": {
      "publicID": [string],
      "name":     [string],
      "metadata": [JSON],
  },
  "roster": [
    [membership],     //a list of the above membership structure
  ],
  "memberships": {
    "pendingApplications": [
      [membership],   //a list of all the pending applications in this clan
    ],
    "pendingInvites": [
      [membership],   //a list of all the pending invites in this clan
    ],
    "denied": [
      [membership],   //a list of all the denied memberships in this clan
    ],
    "banned": [
      [membership],   //a list of all the banned memberships in this clan
```

(continues on next page)

(continued from previous page)

```
    ],
  }
}
```

- Error Response
  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.6.4 Clan Summary

`GET /games/:gameID/clans/:clanPublicID/summary`

Returns a summary of the details of the clan with the given publicID.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "publicID": [string],
  "name": [string],
  "metadata": [JSON],
  "allowApplication": [bool],
  "autoJoin": [bool],
  "membershipCount": [int]
}
```

- Error Response
  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.6.5 Clans Summary

`GET /games/:gameID/clans-summary?clanPublicIds=clan1,clan2,clan3`

Returns a summary of the details for each one of the clans with the given publicIDs.

- Success Response
  - Code: `200`

  - Content:

```
{
  "success": true,
  clans: [
    {
      "publicID": [string],
      "name": [string],
      "metadata": [JSON],
      "allowApplication": [bool],
      "autoJoin": [bool],
      "membershipCount": [int]
    },
    {
      "publicID": [string],
      "name": [string],
      "metadata": [JSON],
      "allowApplication": [bool],
      "autoJoin": [bool],
      "membershipCount": [int]
    },
    ...
  ]
}
```

If no clanPublicIDs are provided:

- Error Response
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

If at least one of the clan was not found:

- Error Response
  - Code: `404`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

In case of other errors:

- Error Response
  - Code: `500`
  - Content:

```
{
  "success": false,
```

(continues on next page)

(continued from previous page)

```
  "reason": [string]
}
```

6.6.6 List Clans

`GET /games/:gameID/clans`

List all clans for the game with publicID=`gameID`.

Warning

Depending on the number of clans in your game this can be a **VERY** expensive operation! Be wary of using this. A better way of getting clans is using clan search.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "clans": [
    {
      "name": [string],
      "metadata": [JSON],
      "membershipCount": [int],
      "publicID": [string],
      "allowApplication": [bool],
      "autoJoin": [bool]
    }
  ]
}
```

An empty list will be returned if there are no clans for the given game.

6.6.7 Search Clans

`GET /games/:gameID/clans/search`

Searches for clans of a given game where the name include the term passed in the query string, or term is a publicID.

Results are limited by "search.pageSize" set via config YAML or environment variable KHAN_SEARCH_PAGESIZE

- URL Parameters

```
term=[string]
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "clans": [
```

(continues on next page)

(continued from previous page)

```
    {
      "name": [string],
      "metadata": [JSON],
      "membershipCount": [int],
      "publicID": [string],
      "allowApplication": [bool],
      "autoJoin": [bool]
    }
  ]
}
```

An empty list will be returned if no clans match the term.

- Error Response

  It will return an error if an empty search term is sent.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": "A search term was not provided to find a clan."
}
```

6.6.8 Leave Clan

`POST /games/:gameID/clans/:clanPublicID/leave`

Allows the owner to leave the clan. If there are no clan members the clan will be deleted. Otherwise, the new clan owner will be the member with the highest level which has the oldest creation date.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "isDeleted":          [bool],       //Indicates whether the clan was deleted
                                      //because there were no members left
  "previousOwner": {
    "publicID":         [string],
    "name":             [string],
    "metadata":         [JSON],
    "membershipCount": [int],
    "ownershipCount":   [int]
  },
  "newOwner": {
    "publicID":         [string],
    "name":             [string],
    "metadata":         [JSON],
    "membershipCount": [int],
    "ownershipCount":   [int]
  }
}
```

- Error Response

  It will return an error if clan is not found.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.6.9 Transfer Clan Ownership

`POST /games/:gameID/clans/:clanPublicID/transfer-ownership`

Allows the owner to transfer the clan's ownership to another clan member of their choice. The previous owner will then be a member with the maximum level allowed for the clan.

- Payload

```
{
  "playerPublicID": [string]  // must match a clan member's public id
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "previousOwner": {
    "publicID":        [string],
    "name":            [string],
    "metadata":        [JSON],
    "membershipCount": [int],
    "ownershipCount":  [int]
  },
  "newOwner": {
    "publicID":        [string],
    "name":            [string],
    "metadata":        [JSON],
    "membershipCount": [int],
    "ownershipCount":  [int]
  }
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.7 Membership Routes

6.7.1 Apply For Membership

`POST /games/:gameID/clans/:clanPublicID/memberships/application`

Allows a player to ask to join the clan with the given publicID. If the clan's autoJoin property is true the member will be automatically approved. Otherwise, the membership must be approved by the clan owner or one of the clan members.

- Payload

```
{
  "level": [string],          // the level of the membership
  "playerPublicID": [string], // the player's public id
  "message": [string]         // optional, a message sent by the player
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true,
  "approved": [bool]  // it will be true if the membership does not require␣
↪additional approval (i.e. clan autoJoin is true)
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

- Code: `500`
- Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.7.2 Approve Or Deny Membership Application

`POST /games/:gameID/clans/:clanPublicID/memberships/application/:action`

`:action` must be either 'approve' or 'deny'.

Allows the clan owner or a clan member to approve or deny a player's application to join the clan. The member's membership level must be at least the game's `minLevelToAcceptApplication`.

- Payload

```
{
  "playerPublicID": [string]    // the public id of player who made the␣
↪application
  "requestorPublicID": [string] // the public id of the clan member or the owner␣
↪who will approve or deny the application
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.7.3 Invite For Membership

`POST /games/:gameID/clans/:clanPublicID/memberships/invitation`

Allows the clan owner or a clan member to invite a player to join the clan with the given publicID. If the request is made by a member of the clan, their membership level must be at least the game's `minLevelToCreateInvitation`. The membership must be approved by the player being invited.

- Payload

```
{
  "level": [string],            // the level of the membership
  "playerPublicID": [string],   // the public id player being invited
  "requestorPublicID": [string] // the public id of the member or the clan owner
↪who is inviting
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.7.4 Approve Or Deny Membership Invitation

`POST /games/:gameID/clans/:clanPublicID/memberships/invitation/:action`

`:action` must be either 'approve' or 'deny'.

Allows a player member to approve or deny a player's invitation to join a given clan.

- Payload

```
{
  "playerPublicID": [string] // the public id of player who was invited
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.7.5 Promote Or Demote Member

`POST /games/:gameID/clans/:clanPublicID/memberships/:action`

`:action` must be either 'promote' or 'demote'.

Allows the clan owner or a clan member to promote or demote another member. When promoting, the member's membership level will be increased by one, when demoting it will be decreased by one. The member's membership level must be at least `minLevelOffsetToPromoteMember` or `minLevelOffsetToDemoteMember` levels greater than the level of the player being promoted or demoted.

- Payload

```
{
  "playerPublicID": [string],   // the public id player being promoted or demoted
  "requestorPublicID": [string] // the public id of the member or the clan owner␣
↪who is promoting or demoting
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

6.7.6 Delete Membership

`POST /games/:gameID/clans/:clanPublicID/memberships/delete`

Allows the clan owner or a clan member to remove another member from the clan. The member's membership level must be at least `minLevelToRemoveMember`. A member can leave the clan by sending the same `playerPublicID` and `requestorPublicID`.

- Payload

```
{
  "playerPublicID": [string],   // the public id player being deleted
  "requestorPublicID": [string] // the public id of the member or the clan owner␣
↪who is deleting the membership
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

- Code: `400`
- Content:

```
{
  "success": false,
  "reason": [string]
}
```

- Code: `500`
- Content:

```
{
  "success": false,
  "reason": [string]
}
```

CHAPTER 7

Pruning Stale Data

Depending on your usage of Khan, you will accumulate some stale data. Some examples include:

- Pending Applications to a clan;
- Pending Invitations to a clan;
- Deleted Memberships (member left or was banned);
- Denied Memberships.

While there's nothing wrong with keeping this data in the Data Store, it will slow Khan down considerably depending on your usage of it.

7.1 Gotchas

One good example of how this sort of stale data can go wrong is when building a clan suggestion for players. If you always suggest the same clans, eventually those clans will have thousands of unfulfilled applications.

That way, anytime someone requests info for these clans, Khan will have a hard time to fulfill that request.

7.2 Pruning Stale Data

Khan has a `prune` command built-in, designed for the purpose of keeping your data balanced. It looks for some configuration keys in your games, and then decides on what data should be deleted.

7.2.1 WARNING

This command performs a **HARD** delete on the memberships row and can't be undone. Please ensure you have frequent backups of your data store before applying pruning.

7.3 Configuring Games to be Pruned

Configuring a game to be pruned is as easy as including some keys in the game's metadata property:

- `pendingApplicationsExpiration`: the number of **SECONDS** to wait before deleting a pending application;
- `pendingInvitesExpiration`: the number of **SECONDS** to wait before deleting a pending invitation;
- `deniedMembershipsExpiration`: the number of **SECONDS** to wait before deleting a denied membership;
- `deletedMembershipsExpiration`: the number of **SECONDS** to wait before deleting a deleted membership (either the member left or was banned).

PLEASE take note that all the expirations are in **SECONDS**. The timestamp used to compare the expiration to is the `updated_at` field of memberships.

Khan will delete any membership that meets one of the criteria above **AND** has an `updated_at` timestamp older than the relevant configuration subtracted in seconds from NOW.

7.3.1 NOTICE

If you want a game to be pruned, **ALL** expiration keys **MUST** be set. Otherwise, Khan will ignore that game as far as pruning goes.

7.4 Periodically Running Pruning

Khan's command line for pruning is:

```
$ khan prune -c /path/to/config.yaml
```

Khan will use the connection details in your specified config file. Double-check the config file being used to ensure that you won't lose any unwanted information.

7.5 Pruning with a Container

Since Khan has container offers, you can also use a container for running pruning in any PaaS that supports Docker containers.

In order to use it, you need to configure these environment variables in the container:

- `KHAN_POSTGRES_HOST` - PostgreSQL to prune hostname;
- `KHAN_POSTGRES_PORT` - PostgreSQL to prune port;
- `KHAN_POSTGRES_USER` - PostgreSQL to prune username;
- `KHAN_POSTGRES_PASSWORD` - PostgreSQL to prune password;
- `KHAN_POSTGRES_DBNAME` - PostgreSQL to prune database name;
- `KHAN_SENTRY_URL` - Sentry URL to send errors to. If you do not use sentry, just leave this unset;
- `KHAN_PRUNING_SLEEP` - Number of seconds to sleep between pruning operations. Defaults to 3600.

The image can be found at our official Docker Hub repository.

CHAPTER 8

Using Postman with Khan

Khan supports Postman to make it easier on users to test their Khan server.

Using Postman with Khan is as simple as importing it's operations and environment into Postman.

8.1 Importing Khan's environment

To import Khan's environment, download it's environment file and import it in Postman.

8.2 Importing Khan's operations

To import Khan's operations, download it's operations file and import it in Postman.

8.3 Running Khan's operations with a different environment

Just configure a new environment and make sure it contains the `baseKhanURL` variable with a value like `http://my-khan-server/`. Do not forget the ending slash or it won't work.

CHAPTER 9

Khan's Benchmarks

You can see khan's benchmarks in our CI server as they get run with every build.

9.1 Creating the performance database

To create a new database for running your benchmarks, just run:

```
$ make drop-perf migrate-perf
```

9.2 Running Benchmarks

If you want to run your own benchmarks, just download the project, and run:

```
$ make run-test-khan run-perf
```

9.3 Generating test data

If you want to run your perf tests against a database with more volume of data, just run this command prior to running the above one:

```
$ make drop-perf migrate-perf db-perf
```

Warning: This will take a long time running (around 30m).

9.4 Results

The results should be similar to these:

```
BenchmarkCreateClan-2                          2000       3053999 ns/op
BenchmarkUpdateClan-2                          2000       2000650 ns/op
BenchmarkRetrieveClan-2                         500      10522248 ns/op
BenchmarkRetrieveClanSummary-2                 5000       1187486 ns/op
BenchmarkSearchClan-2                          5000       1205325 ns/op
BenchmarkListClans-2                           5000       1135555 ns/op
BenchmarkLeaveClan-2                           1000       3824284 ns/op
BenchmarkTransferOwnership-2                    500       8642818 ns/op
BenchmarkCreateGame-2                          3000       1248042 ns/op
BenchmarkUpdateGame-2                          2000       2141705 ns/op
BenchmarkApplyForMembership-2                  1000       5695344 ns/op
BenchmarkInviteForMembership-2                  500       8916792 ns/op
BenchmarkApproveMembershipApplication-2         500      13480574 ns/op
BenchmarkApproveMembershipInvitation-2         1000      10517905 ns/op
BenchmarkDeleteMembership-2                     500       9548314 ns/op
BenchmarkPromoteMembership-2                    500       8961424 ns/op
BenchmarkDemoteMembership-2                     500       9202060 ns/op
BenchmarkCreatePlayer-2                        3000       1344267 ns/op
BenchmarkUpdatePlayer-2                        3000       1829329 ns/op
BenchmarkRetrievePlayer-2                       300      14412830 ns/op
```

CHAPTER 10

Indices and tables

- genindex
- modindex
- search