

# **Nakama 2 + Unity Engine**

**Heroic Labs**

# Setup Server

- Local development use Docker or native binaries.

```
docker-compose -f ./docker-compose.yml up
```

- Always run Nakama servers and databases on dedicated hardware at launch.
- Production-ready server environments with our Managed Cloud service.

# Development Configuration

- `--logger.level=debug`

Enable additional logs (can be noisy).

- `--socket.max_message_size_bytes=8192`

Increase message size limits to match max packet size.

- `--session.token_expiry_sec=604800`

Adjust session lifetime to fit gameplay sessions.

# Unity / .NET Client

```
// using Nakama;  
var client = new Client("http", "127.0.0.1", 7350, "defaultkey");
```

- Updated for Unity 2018 or newer.
- Requires .NET4.6 scripting runtime compatibility level.
- Uses async/await for simple asynchronous code.
- Divided into "low level" client and Unity wrapper.
- Unity wrapper contains features specific to the engine.

# Sessions

— Authenticate to register/login. Many sign-in options.

```
var deviceId = SystemInfo.deviceUniqueIdentifier;  
var session = await client.AuthenticateDeviceAsync(deviceId);
```

— Sessions can be cached on device and restored.

```
PlayerPrefs.SetString("nakama.session", session.AuthToken);  
// Restore.  
var authToken = PlayerPrefs.GetString("nakama.session");  
var session = Session.Restore(authToken);
```

— Session used to authenticate all requests.

# Users & Accounts

— Sessions contain essential user details.

```
Debug.Log(session.UserId);    // "ea1e7609-372a-4d67-a495-58f955f3328b"  
Debug.Log(session.Username); // "wRkuUTbKmY"
```

— Each player has an account (private) and a user profile.

```
var account = await client.GetAccountAsync(session);  
Debug.Log(account.User.Id); // "ea1e7609-372a-4d67-a495-58f955f3328b"
```

# Social Accounts

- A user account can "link" additional sign-in options.
- Useful to enable users to sign-in across multiple devices.

```
// using Facebook.Unity;  
var perms = new List<string>(){ "public_profile", "email" };  
FB.LogInWithReadPermissions(perms, async (ILoginResult result) => {  
    if (FB.IsLoggedIn) {  
        var accessToken = Facebook.Unity.AccessToken.CurrentAccessToken;  
        await client.LinkFacebookAsync(session, accessToken);  
    }  
});
```

# Friends

— Create a social graph of friends within the server.

```
// Both users must add each other to become friends. Double opt-in.
await client.AddFriendsAsync(session, new[] { "user id" });
var result = await client.ListFriendsAsync(session);

foreach (var f in result.Friends) {
    Debug.Log("Friend name {0}", f.User.DisplayName);
    // State one of: friend(0), invite_sent(1), invite_received(2), blocked(3)
    Debug.Log("Friend state {0}", f.State);
}
```



# Groups

- Groups have 3 membership levels: superadmin, admin, and member.
- Use groups for guilds, clans, or any kind of team-based gameplay.

```
const string name = "heroic";  
const string desc = "game server devs";  
var group = await client.CreateGroupAsync(session, name, desc);  
Debug.Log("New group {0}", group.Id);
```

# Rpc Functions

— Define functions on the server in Lua or Go.

```
-- in lua
local nk = require("nakama")
local function some_action(context, payload)
    return nk.json_encode({ message = "PONG" })
end
nk.register_rpc(some_action, "<function id>")
```

— Execute them with the client.

```
var rpc = await client.RpcAsync(session, "<function id>");
// using Nakama.TinyJson;
var content = rpc.Payload.FromJson<Dictionary<string, string>>();
Debug.Log("Response content {0}", content);
```

# Leaderboards

- Create unlimited leaderboards.
- A record can have a score and subscore.
- Build friend or guild leaderboards with a filter on user ids.

```
var result = await client.ListLeaderboardRecordsAsync(session, "<id>", null, 100);
foreach (var r in result.Records) {
    Debug.Log("Score '{0}' for user '{1}'", r.Score, r.Username);
}
```

# In-app Notifications

- Send notifications which can be received in realtime.
- Notifications must be sent authoritatively (Lua or Go).

```
-- in lua
local nk = require("nakama")
local notification = { code = 1, content = {}, persistent = true,
    sender_id = "someid", subject = "Match winner!", user_id = "userid" }
nk.notifications_send({ notification })
```

- List notifications received while offline.

```
var result = await client.ListNotificationsAsync(session, 100);
Debug.Log("Received {0} notifications", result.Notifications.Count());
```

# Sockets

- Power chat, multiplayer, status events, in-app notifications, etc.
- Create a socket from a client object.

```
var socket = client.NewSocket();  
socket.Connected += () => Debug.Log("Socket connected.");  
socket.Closed += () => Debug.Log("Socket closed.");  
await socket.ConnectAsync(session);  
Debug.Log("After socket connected.");  
await socket.CloseAsync();
```

- Have separate sockets for multiplayer and chat or,
- Share a single socket for all realtime communication.

## **Lots more APIs**

- Realtime chat
- Status events
- Authoritative multiplayer
- Realtime multiplayer
- Matchmaker
- Storage engine
- Remote configuration, etc...

# Summary

- Designed as production-ready infrastructure.
- Minimal database or other external dependencies.
- First-class Unity engine support.
- Modern client designed for asynchronous code.
- Built for scale by Heroic Labs.