INDIEGAMELAB

Contents

IndieGamesLab

IndieGamesLab is a messaging platform built on Microsoft Azure. The basis of the platform has been evolving since 2009 (previously known as AzureGames) and targets indie game developers wanting a low cost but flexible cloud solution.

Goals

The goals of IndieGamesLab are as follows:

- Create a messaging platform for indie game developers
- Setting up the messaging platform should be as simple as possible
- The platform should be easily customized for different scenarios
- The platform should be secure
- The platform should be compatible with many game development frameworks including Unity and Construct2

IGL has been designed to not compete with existing messaging solution like Photon and Unity Multiplayer in regards to real time networking. IGL is more suited for messaging scenarios requiring custom back-end processing, auditing/logging or turn-based games.

Source

The community repository for the framework is hosted on GIT: IndieGamesLab. The framework contains the common classes and core functionality for IGL. The community repository for the samples is hosted on GIT: IndieGamesLab.Samples. The samples illustrate how the framework can be used in game development. The source is only required if you wish to contribute to the community by making a change.

Overview

The solution is broken into two main components: Service and Client. The intention is there will be multiple clients sending and receiving messages from a central service. Each message will be sent securely with encrypted content limiting the chance for packets to be intercepted and altered. The communication uses the Azure Service Bus and can support communication at a global level though the speed and cost can be improved when multiple Azure data-centers are used and when the service bus and IGL Service are located in the same data-center.

Getting Started in Azure

There are many great resources to getting started in Azure. Creating a subscription is simple:

Create your free Azure account today

For the IGL backbone, you need to create a service bus at a minimum.

Service Bus

A service could be hosted in a number of ways from a local pc to a hosted service. For example, a cloud service provides a simple economic way of getting a service in the cloud:

Cloud Service Documentation

Credit

Also, if you are a student or a startup there are Microsoft programs created to provide you with credit and guidance to get started. And, if you are an existing MSDN subscriber you probably already have benefits.

Microsoft Image for Students

Microsoft BizSpark

MSDN Dev/Test Credit

Service

The service defines tasks for processing messages posted to the IGL backbone. This is a .net library, IGL.Service, and is available via Nuget: IGL.Service.

Online documentation on the IGL Service.

Cloud Service

An implementation of a role task service handles messages posted to the IGL backbone and is intended to be hosted on a different server than the machine where the client is running. The service is defined in a .Net library that provides a pattern for hosting a service that receives messages from clients.

IRoleTask

The role task interface defines the methods that an IGL service task must implement.

```
public interface IRoleTask
{
    /// <summary>
    /// Event called when a GameEvent has been successfully processed.
    /// </summary>
    event EventHandler<GamePacketArgs> OnGamePacketCompleted;
    /// <summary>
    /// Event called when a failure has happened processing a GameEvent.
    /// </summary>
    event EventHandler<GamePacketErrorArgs> OnGamePacketError;

    Task ProcessReceivedMessage(Task<IEnumerable<BrokeredMessage>> task);
}
```

RoleTaskRunner

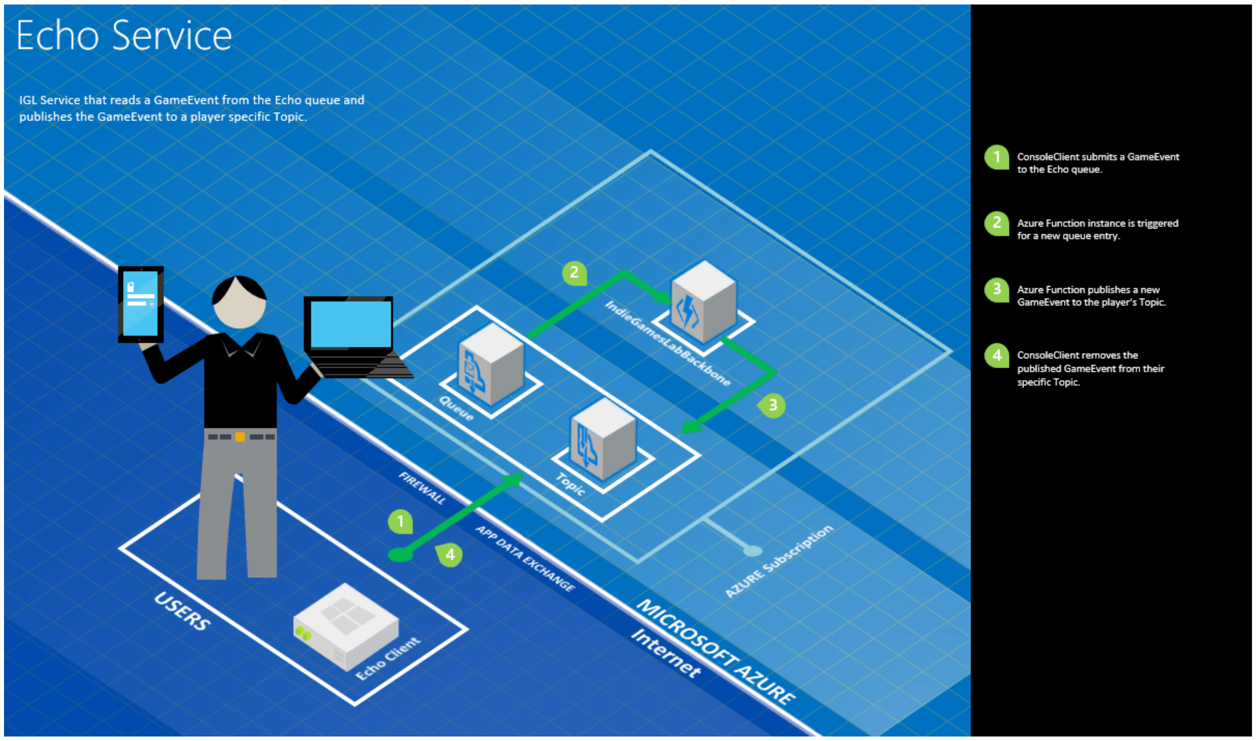
The RoleTaskRunner exposes a static method for launching an IGL service task.

```
public static async Task RunAsync<TRoleTask>(CancellationToken cancellationToken, string
queueName, int batchSize, TimeSpan serverWaitTime) where TRoleTask : IRoleTask, new()
```

Azure Function

Hosting the IGL Service as an Azure Function is ideal in many circumstances because of its scalability and low cost of hosting. With 1 million free executions a month, most indie games will have zero cost during the initial phases of a new game.

The Echo Service is triggered when a new GameEvent is posted to the Echo Queue. The service will then publish a new message to the player's PlayerEvents Topic. The following illustrates the behavior of the Echo Service:



1. ConsoleClient submits a GameEvent to the Echo queue.
2. Azure Function instance is triggered for a new queue entry.
3. Azure Function publishes a new GameEvent to the player's Topic.
4. ConsoleClient removes the published GameEvent from their specific Topic.

The IGL.Samples project has the files required to host the Echo Service as well as a sample Echo client for testing the service.

Note: While the 1 million free executions lasts, the service will be available to testing while the cost of the IGLBackbone is less than IGL's monthly budget. The current IGLBackbone status can be found on the Azure Function online documentation.

Client

A client is any technology that can make a https call. There is a .net client library, IGL.Client, that designed to simplify the messaging and is available via Nuget: IGL.Client.

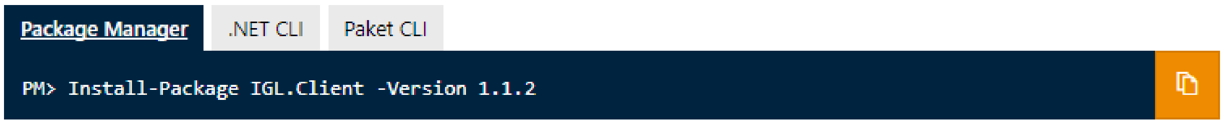
Console Application

The following are the steps to create a console application. First is to create a new console application project illustrating how to send a message to an IGL service bus.

Using Package Manager add a reference to the IGL Nuget packages (Note: visit the links below to get the current version number):

- IGL.Common
- IGL.Client

Be sure to specify the project when adding the reference:



```
PM> Install-Package IGL.Client -Version 1.1.2
```

In the Program.cs file, the information on how to connect to the specific IndieGamesLab framework needs to be added (Please see the Service page for more information on determining these values):

```
// set the following once
CommonConfiguration.Instance.BackboneConfiguration.IssuerName = "IGLGuestClient";
CommonConfiguration.Instance.BackboneConfiguration.IssuerSecret =
"{packagewillhavelatestkey}";
CommonConfiguration.Instance.BackboneConfiguration.ServiceNamespace = "indiegameslab";
```

The next step will be to create the message you want to send:

```
var gameevent = new GameEvent
{
    Properties = new Dictionary<string, string>()
    {
        { "PlayerName", "Joe Smith" },
        { "StartMilliseconds", _stopwath.ElapsedMilliseconds.ToString() }
    }
};
```

And finally the SubmitGameEvent() method is called specifying the queue to send the message to, an event id and the created GameEvent.

```
bool wasSuccessful = IGL.Client.ServiceBusWriter.SubmitGameEvent("Echo", 100, gameevent);
```

Important Links

GitHub Source - Service

The IndieGamesLab (IGL) solution is a collection of projects designed to help game development teams to leverage the Azure cloud platform. The projects in this solution make up the basis for sample projects in the IndieGamesLab.Samples repository.

Technical Requirements

IGL is a Visual Studio solution with the following requirements:

- Visual Studio Enterprise
- Enterprise is required as the project uses Microsoft Fakes
- .Net 4.5+
- Azure Subscription
- The project will build and can be tested without an active subscription using Microsoft Fakes

https://github.com/spikesoftware/IndieGamesLab

IndieGamesLab-Samples

A collection of messaging samples built using the IndieGamesLab framework.

Technical Requirements

IGL.Samples is a Visual Studio solution with the following requirements:

- Visual Studio
- .Net 3.5, 4.5+ ** The client is built in .Net 3.5 for compatibility with Unity3D
- Azure Subscription
-

https://github.com/spikesoftware/IndieGamesLab-Samples

Nuget Packages

Service Library for IndieGamesLab server components.

https://www.nuget.org/packages/IGL.Service/

Client library for communicating with IGL Services.

https://www.nuget.org/packages/IGL.Client/

Common library for IndieGamesLab components shared between client and server.

https://www.nuget.org/packages/IGL.Common/

Contact

You can reach us either by using the contact form on the online documentation or by sending an email directly to indiegameslab@outlook.com.

Also for issues with the packages please post to either project:

https://github.com/spikesoftware/IndieGamesLab/issues

https://github.com/spikesoftware/IndieGamesLab-Samples/issues