

Manual para la conexión entre Unreal Engine 5 y ROS usando ROSIntegration

En este tutorial se usa un PC con Windows 10, en donde hemos instalado Unreal Engine 5.1.1., y Ubuntu 20.04 que corre en una máquina virtual de VirtualBox 7.0.

El objetivo es poder conectar una aplicación de ROS 1 Noetic que se encuentra en la máquina virtual con la simulación de un robot que se lleva a cabo en Unreal Engine en Windows.

Esto se lleva a cabo usando el paquete de rosbridge en la máquina virtual y el plugin de ROSIntegration en Unreal Engine.

Para la instalación de Ubuntu 20.04 en la máquina virtual de VirtualBox por favor leer los tutoriales correspondientes.

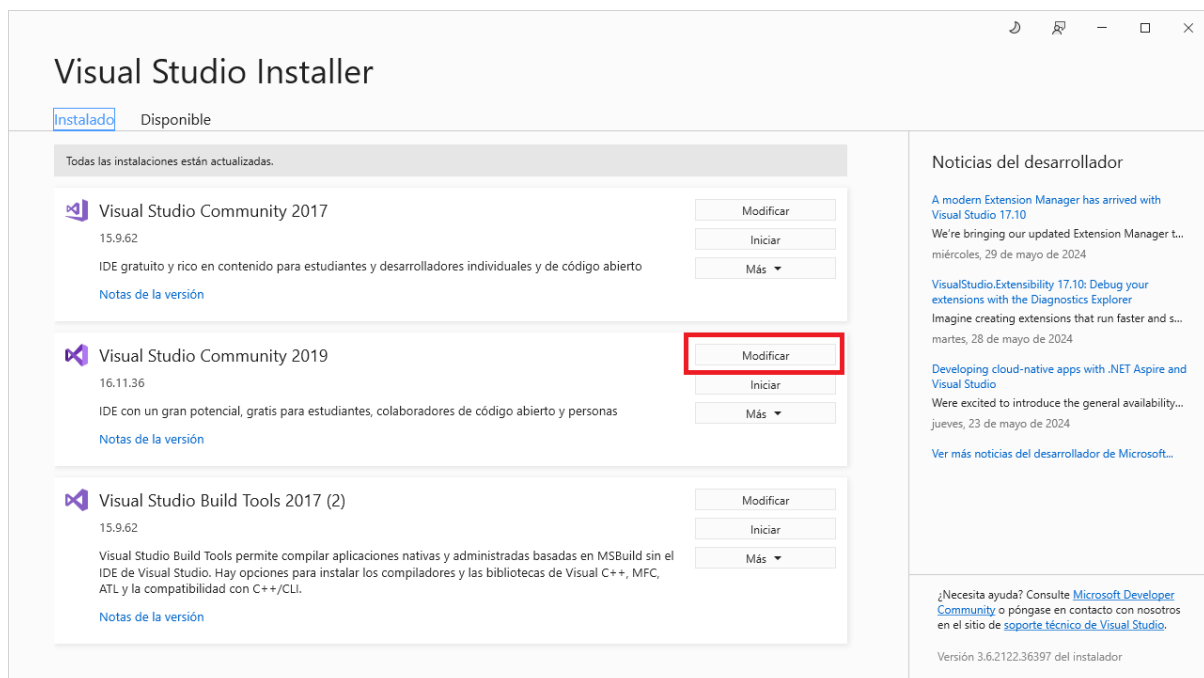
Instalación de Unreal Engine 5.1.1.

Configurar Visual Studio 2019 para Unreal Engine.

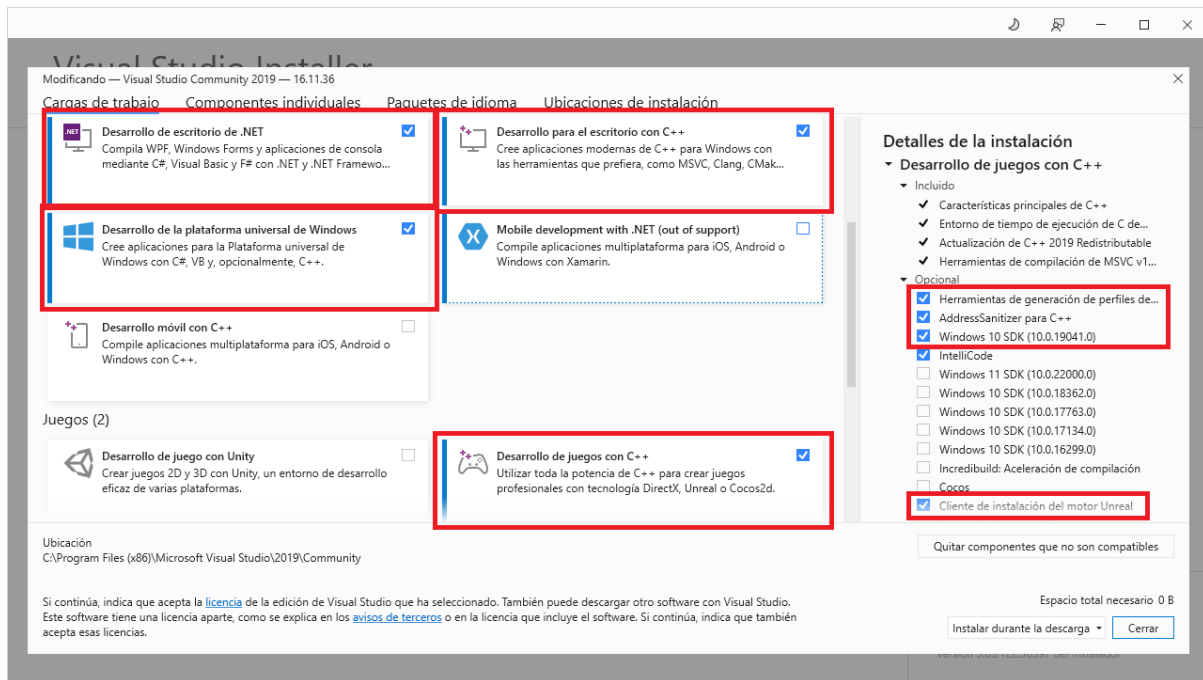
Para cualquier consulta aquí está la guía oficial:

<https://dev.epicgames.com/documentation/en-us/unreal-engine/setting-up-visual-studio-development-environment-for-cplusplus-projects-in-unreal-engine>

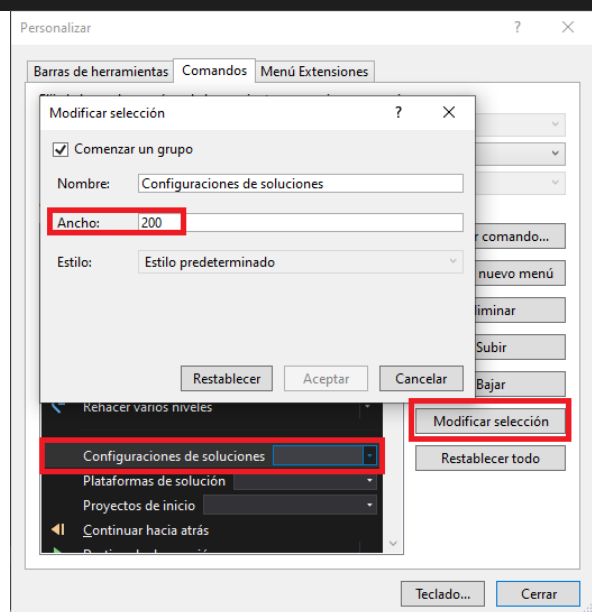
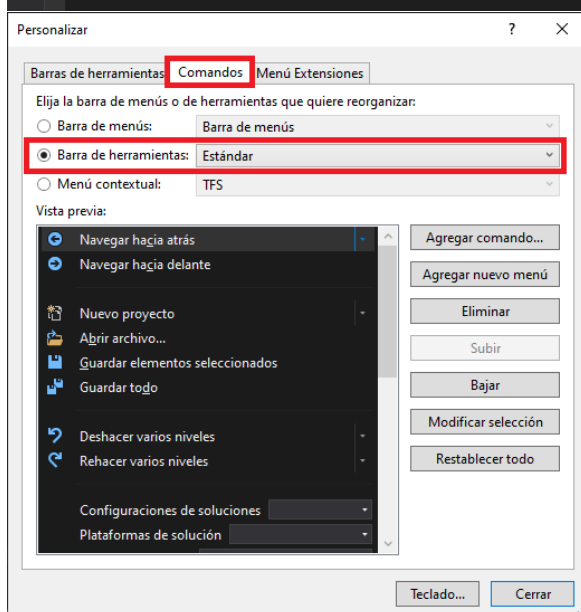
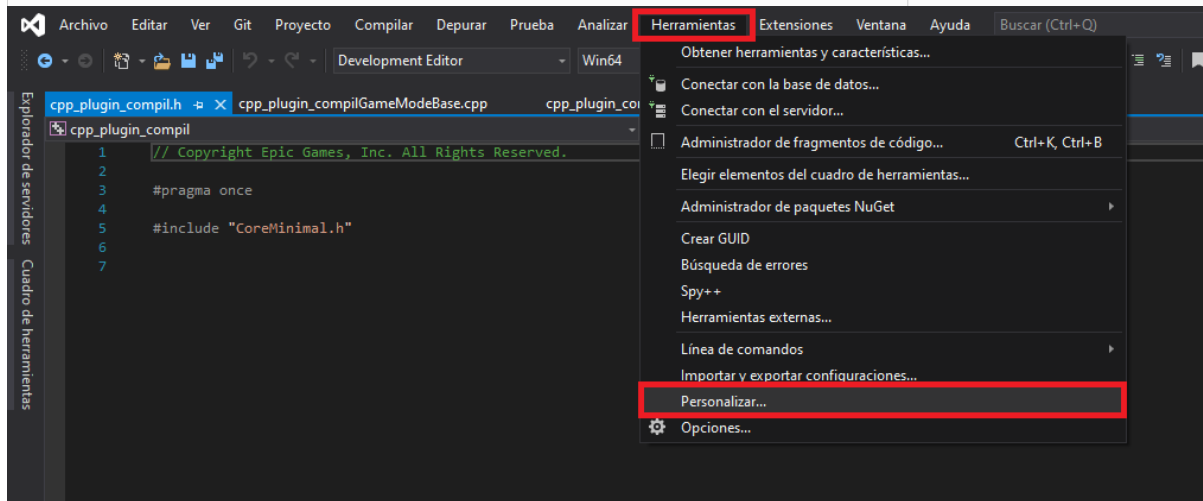
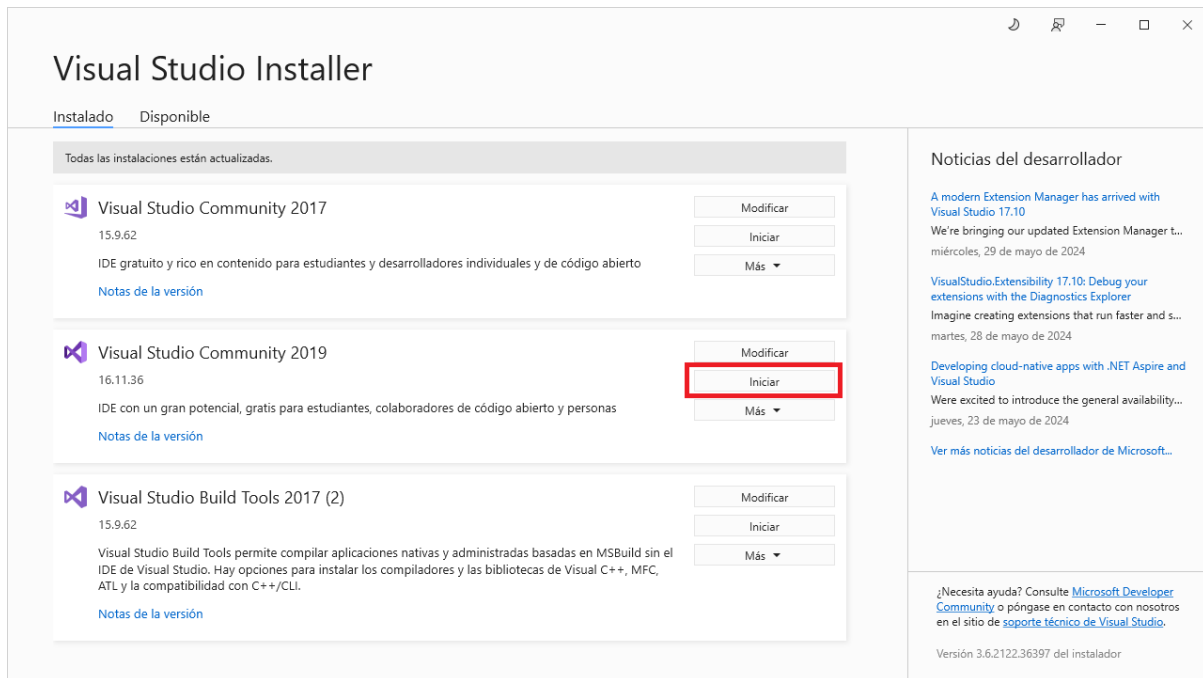
Abre Visual Studio Installer y haz clic en modificar.

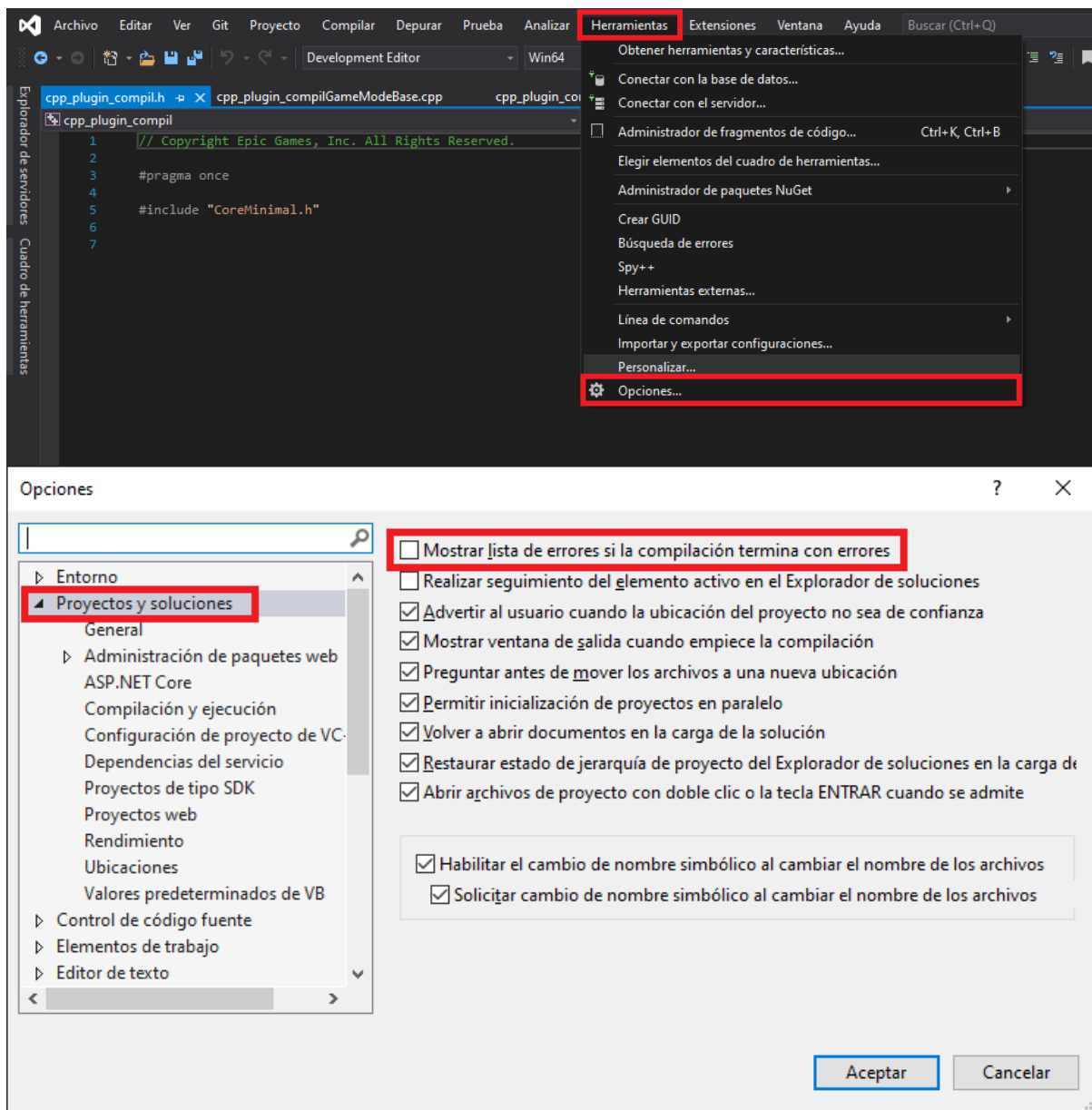


Si no tienes instalado Visual Studio primero descargar Visual Studio Installer y descargar la versión 2019 con la siguiente configuración. Si ya lo tienes instalado esta es la configuración que deberás modificar para tener al menos los paquetes que se señalan:

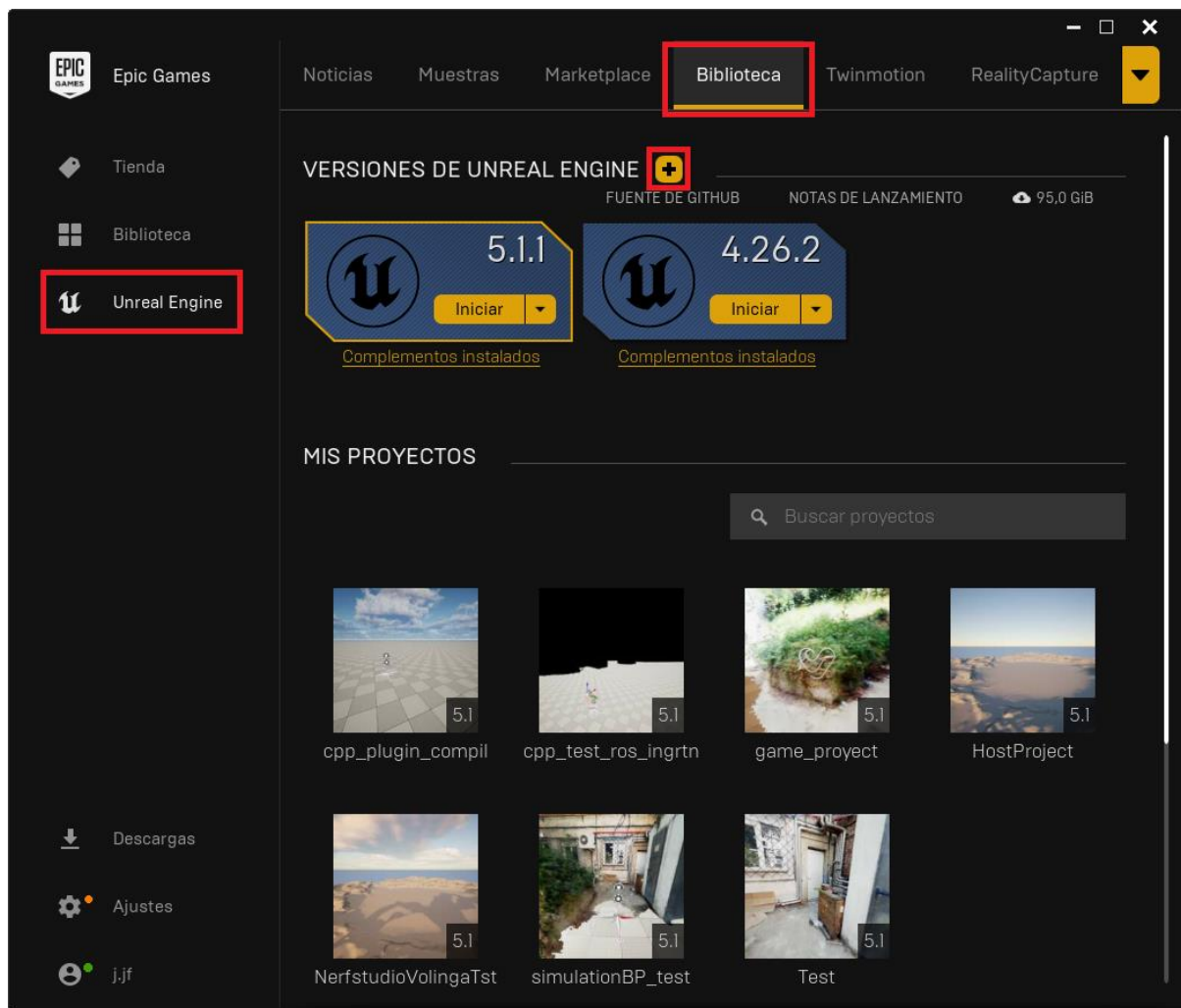


Una vez hecho esto abrir Visual Studio y cambia la siguiente configuración:



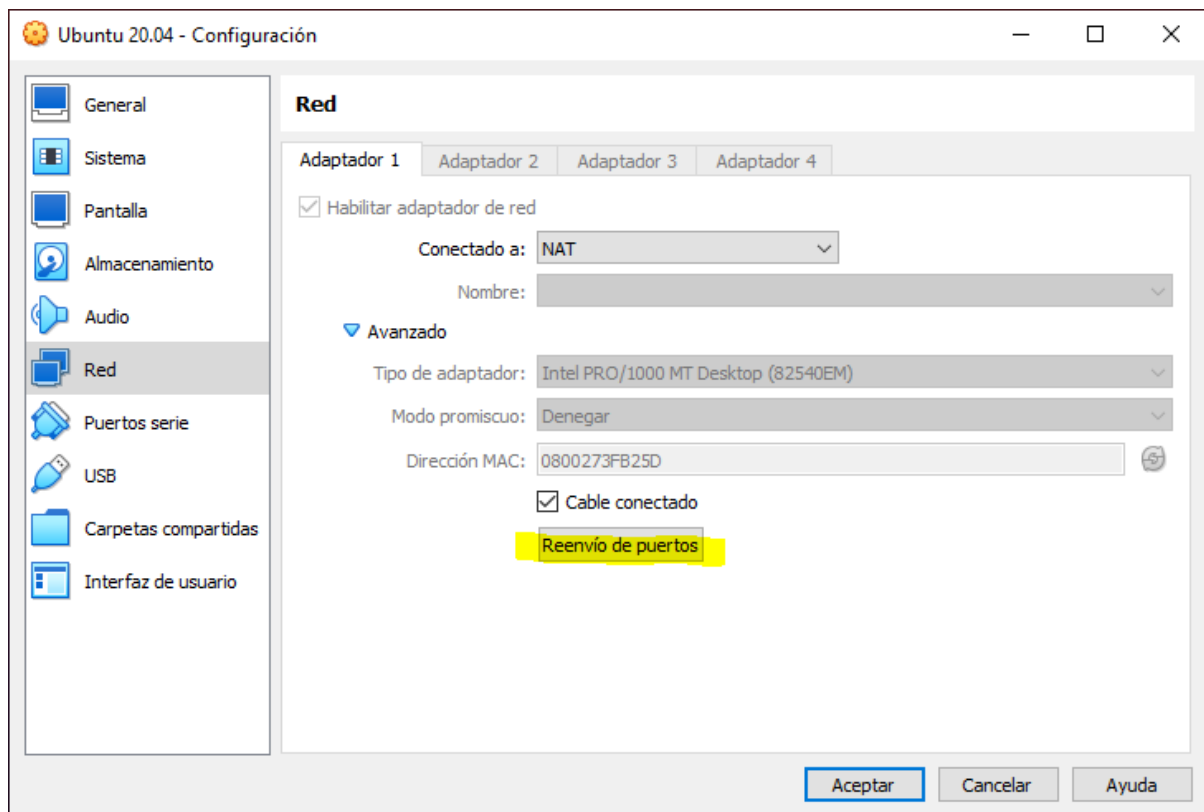


Una vez configurado Visual Studio descargar el Iniciador de Epic Games y crear una cuenta en Epic. Buscar Unreal Engine en el iniciador. Dentro del menú de Unreal Engine ir a la pestaña de Biblioteca y hacer clic sobre el símbolo de + para instalar la versión deseada:

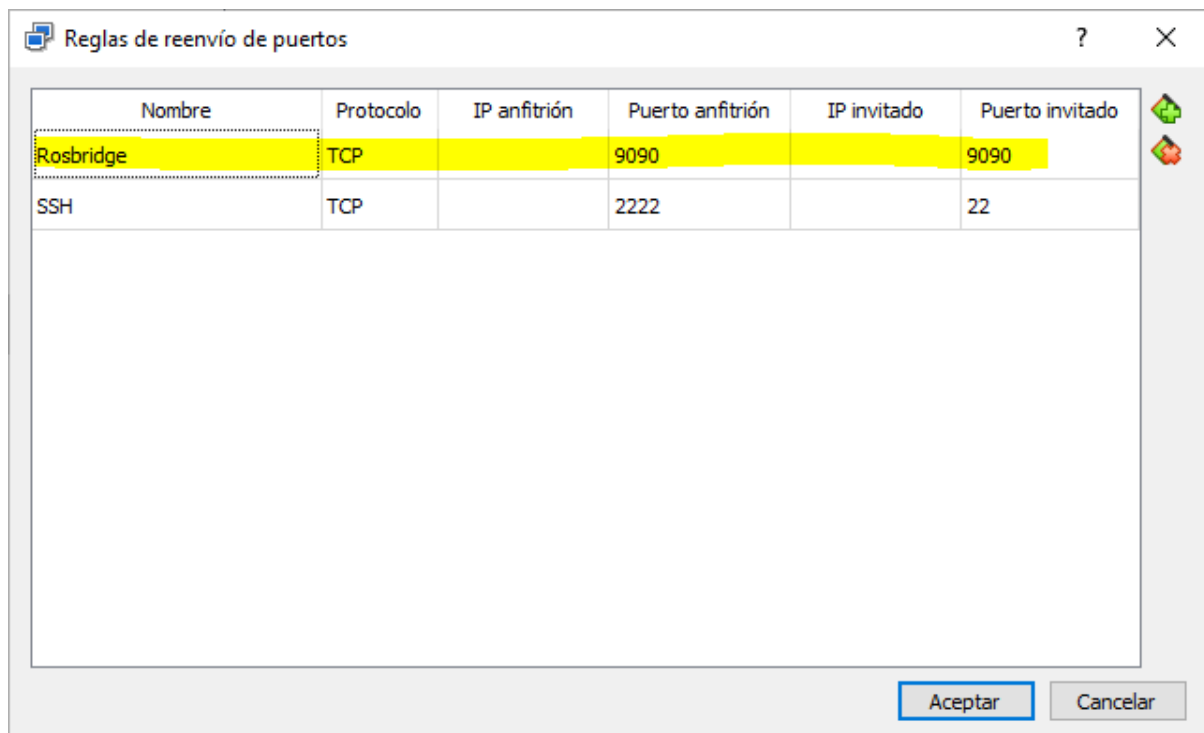


Configuración del puerto 9090

En primer lugar, abriremos el puerto 9090 de nuestra máquina virtual para la comunicación TCP. En VirtualBox entramos en la configuración de la máquina de Ubuntu 20.04 y en Red, hacemos click en el desplegable de Avanzado y accedemos a Reenvío de Puertos:

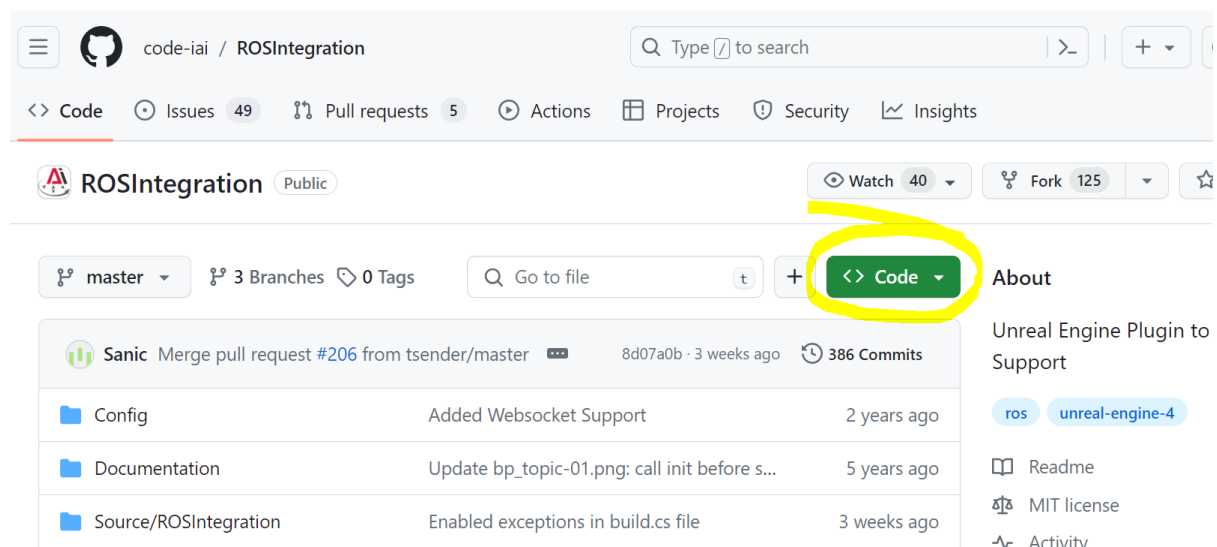


En reenvío de puertos añadimos una regla usando el icono del más y configuramos la regla de la siguiente manera:



Una vez hecho esto ya podemos continuar.

Instalación de rosbridge



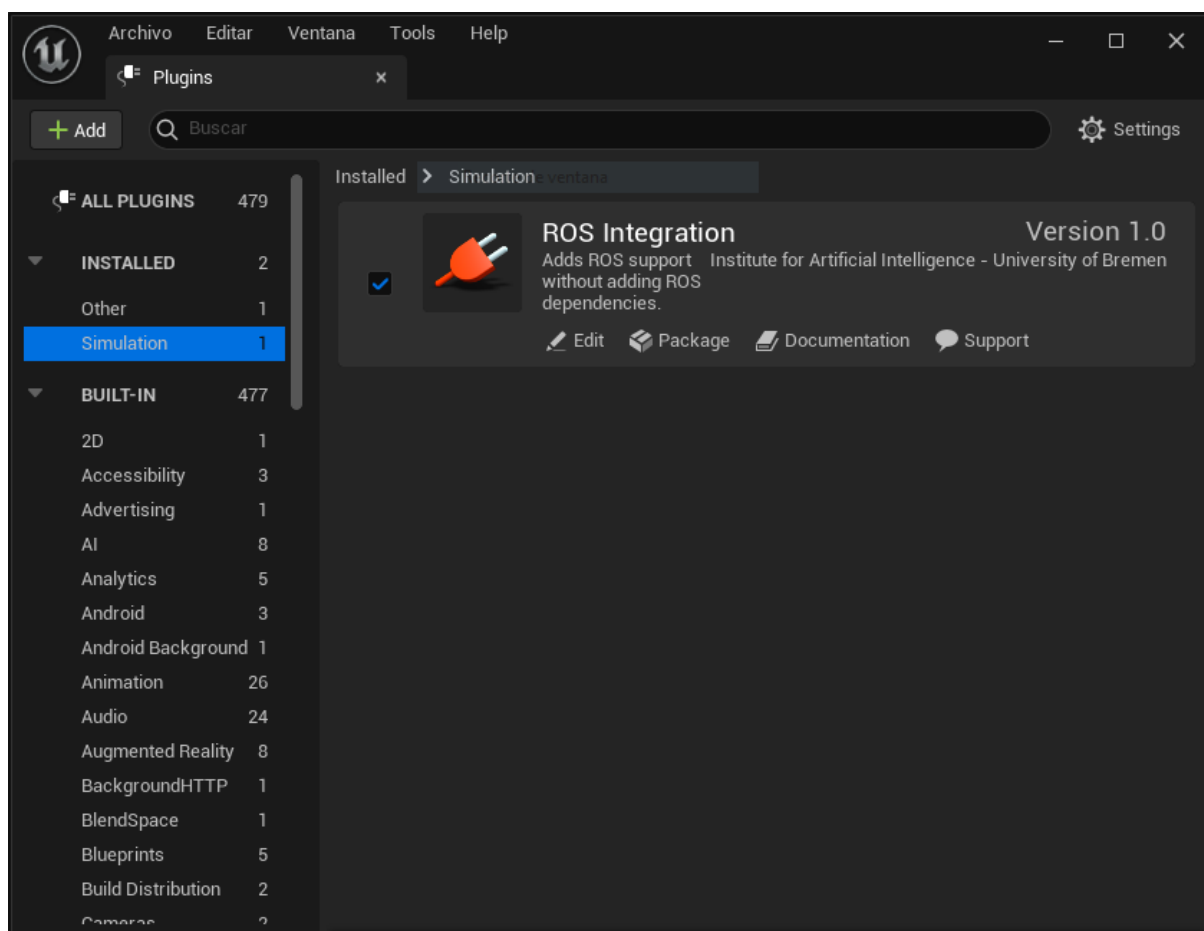
Dado que el plugin fue originalmente creado para UE4 no funcionará directamente en UE5. Para que lo haga hay que recompilarlo antes. Por favor usa uno de los siguientes métodos para hacerlo (el más sencillo es usar un proyecto C++ explicado a continuación):

<https://dev.epicgames.com/community/learning/tutorials/qz93/unreal-engine-building-plugins>

Método creando un proyecto C++. Para llevarlo a cabo crear un proyecto de C++ vacío en UE y añadir el plugin a la carpeta de Plugins de éste. Si no existe una carpeta Plugins dentro de nuestro proyecto la crearemos y copiaremos la carpeta de ROSIntegration dentro. Una vez hecho esto abrir el proyecto y en el menú Edit entrar en Plugins. Buscar el plugin de ROSIntegration y marcar la casilla de añadir. Para que se complete la acción hay que reiniciar el proyecto. Al volver a entrar se preguntará si se desea recompilar el plugin, darle a sí y esperar unos 15 min hasta que termine (aunque no aparezca ninguna ventana ni nada no reintentar o se romperá y la compilación no funcionará).

Una vez recompilado lo copiaremos y pegaremos en la carpeta de plugins de nuestro proyecto. Si no existe una carpeta Plugins dentro de nuestro proyecto la crearemos y copiaremos el plugin dentro.

Una vez hecho esto abrimos nuestro proyecto en Unreal Engine. Dentro del proyecto en el menú Editar accedemos a la lista de plugins: Editar>Plugins. En la lista buscamos ROSIntegration y seleccionamos la casilla para activar el Plugin.



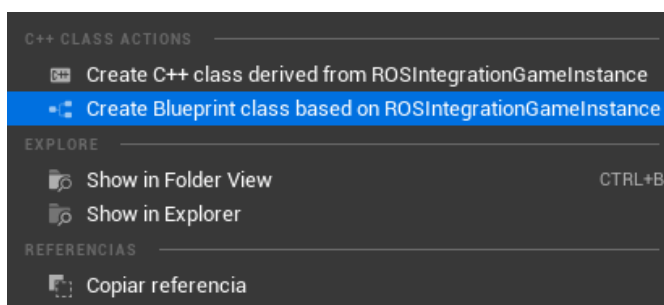
Si hemos añadido el plugin con el proyecto abierto es posible que haga falta refrescar o cerrar el proyecto y volver a abrir para que aparezca en la lista. Una vez activado nos pedirá cerrar el proyecto y volverlo a abrir para poder aplicar los cambios. Entonces lo podremos usar normalmente.

Para especificar un servidor de ROS con el que conectarnos (que será nuestra máquina virtual), hay que crear una `GameInstance` personalizada que herede las características de `ROSIntegrationGameInstance` (la `GameInstance` por defecto del plugin).

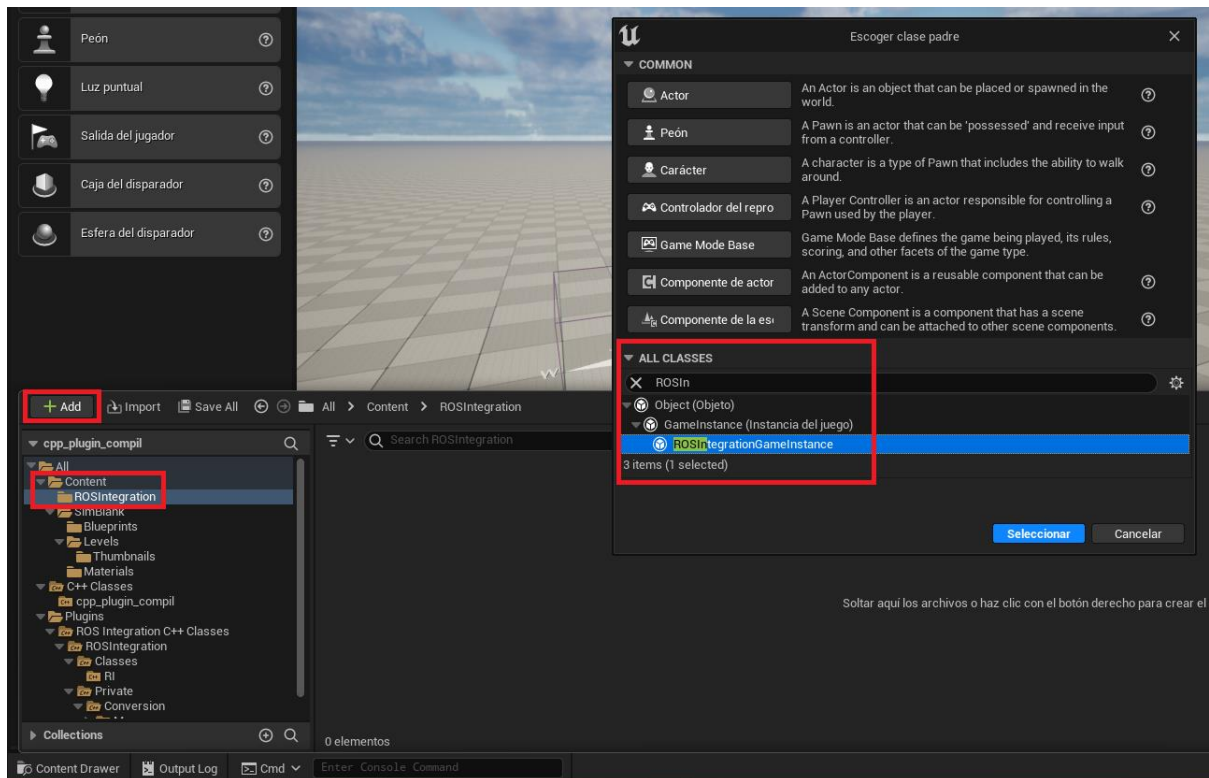
Para ello abriremos el Content Drawer y escribiremos `ROSIntegrationGameInstance` en el buscador:



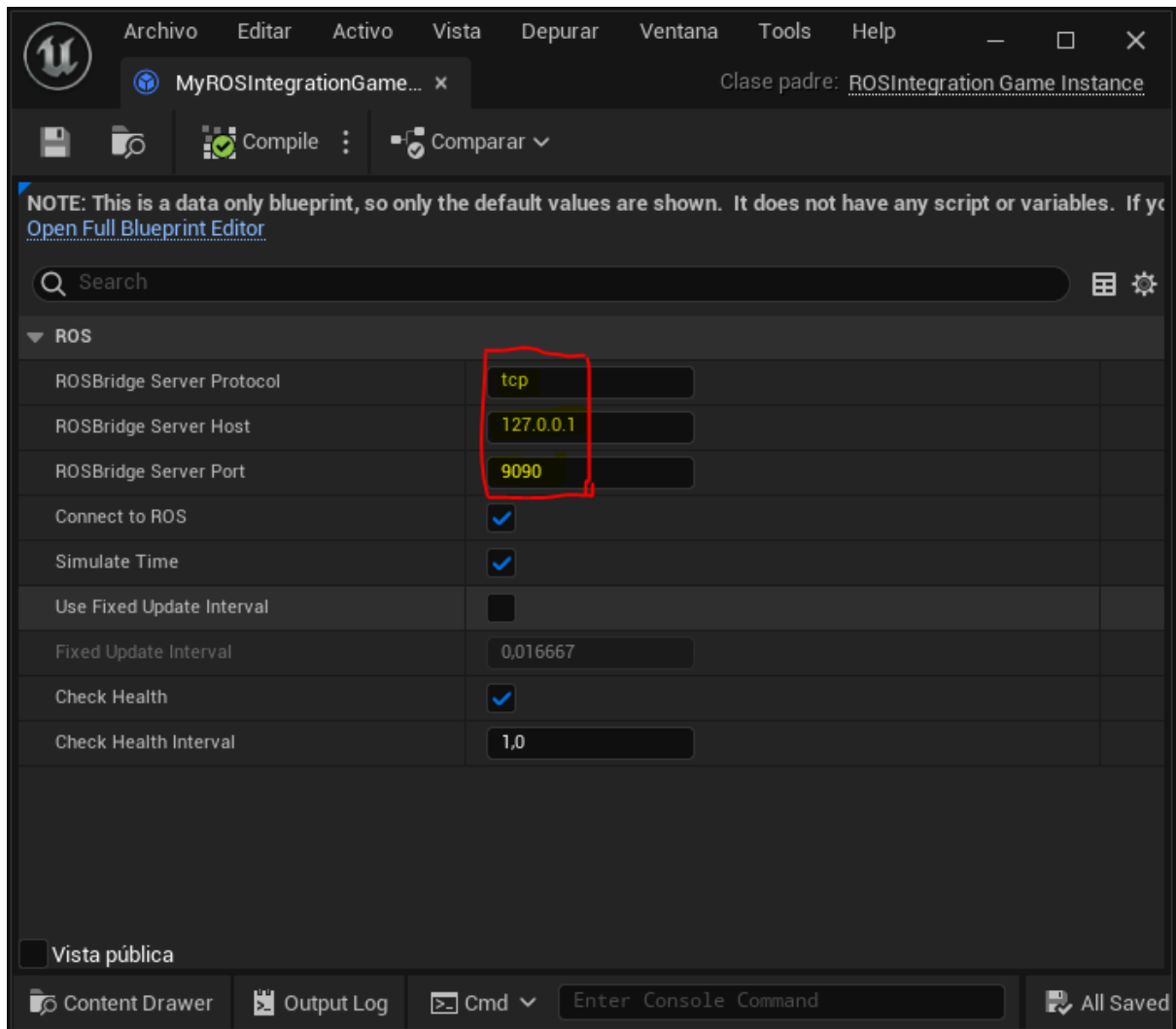
Haciendo click derecho sobre `ROSIntegrationGameInstance` seleccionamos 'Create Blueprint class based on `ROSIntegrationGameInstance`'.



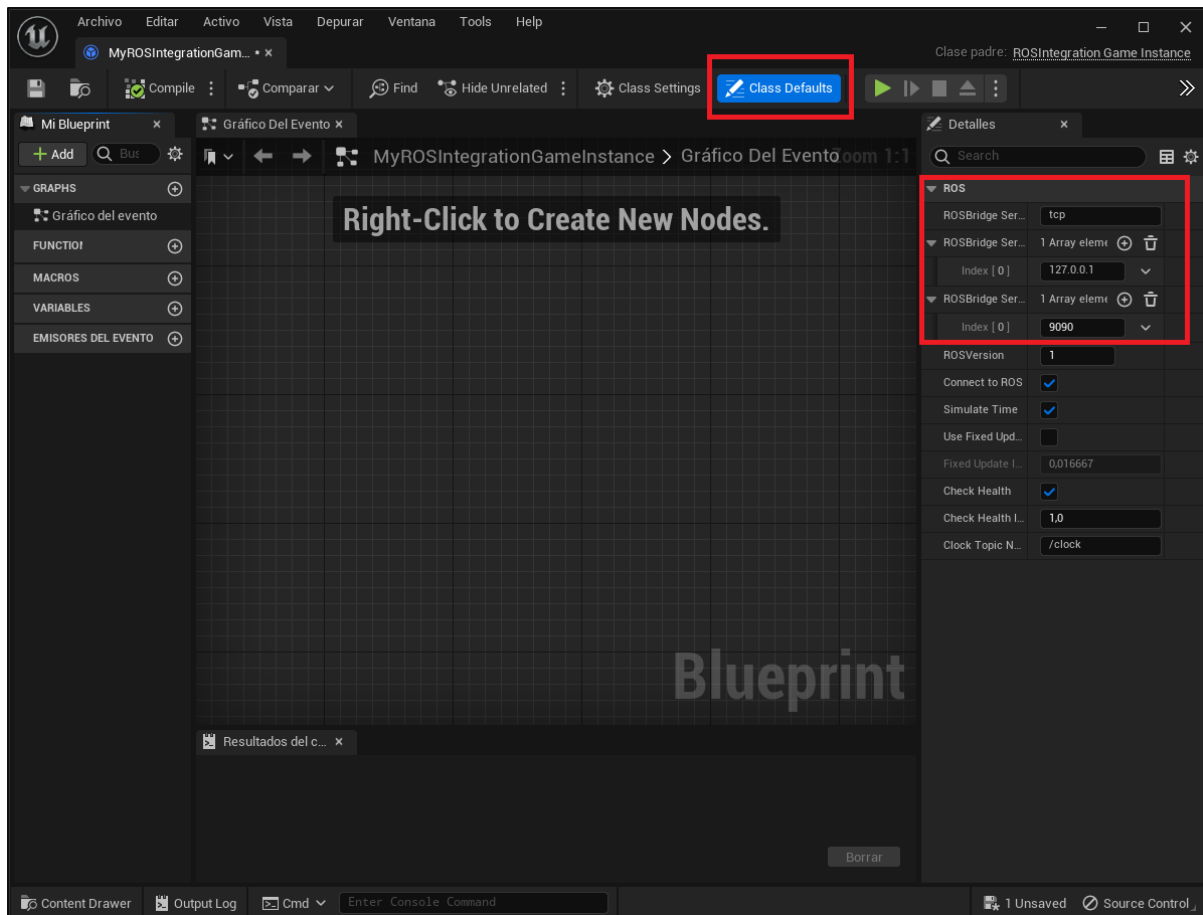
Otra forma más elegante es en nuestra carpeta de content, crear una carpeta de `ROSIntegration` y allí crear un nuevo blueprint, en el creador de blueprints buscar la clase padre con el nombre de `ROSIntegrationGameInstance` y darle a crear. La llamaremos `MyROSIntegrationGameInstance`.



De nuevo en el Content Drawer hacemos doble click sobre la nueva clase y verificamos que los Class Defaults son correctos: el protocolo de conexión está establecido como TCP, el puerto como el 9090 y el Host es 127.0.0.1 que es el de la máquina virtual en nuestro propio equipo, pero si nuestro servidor se encontrase en otra dirección habría que escribir aquí la dirección correspondiente:



Puede que la ventana anterior se muestre de la siguiente manera:



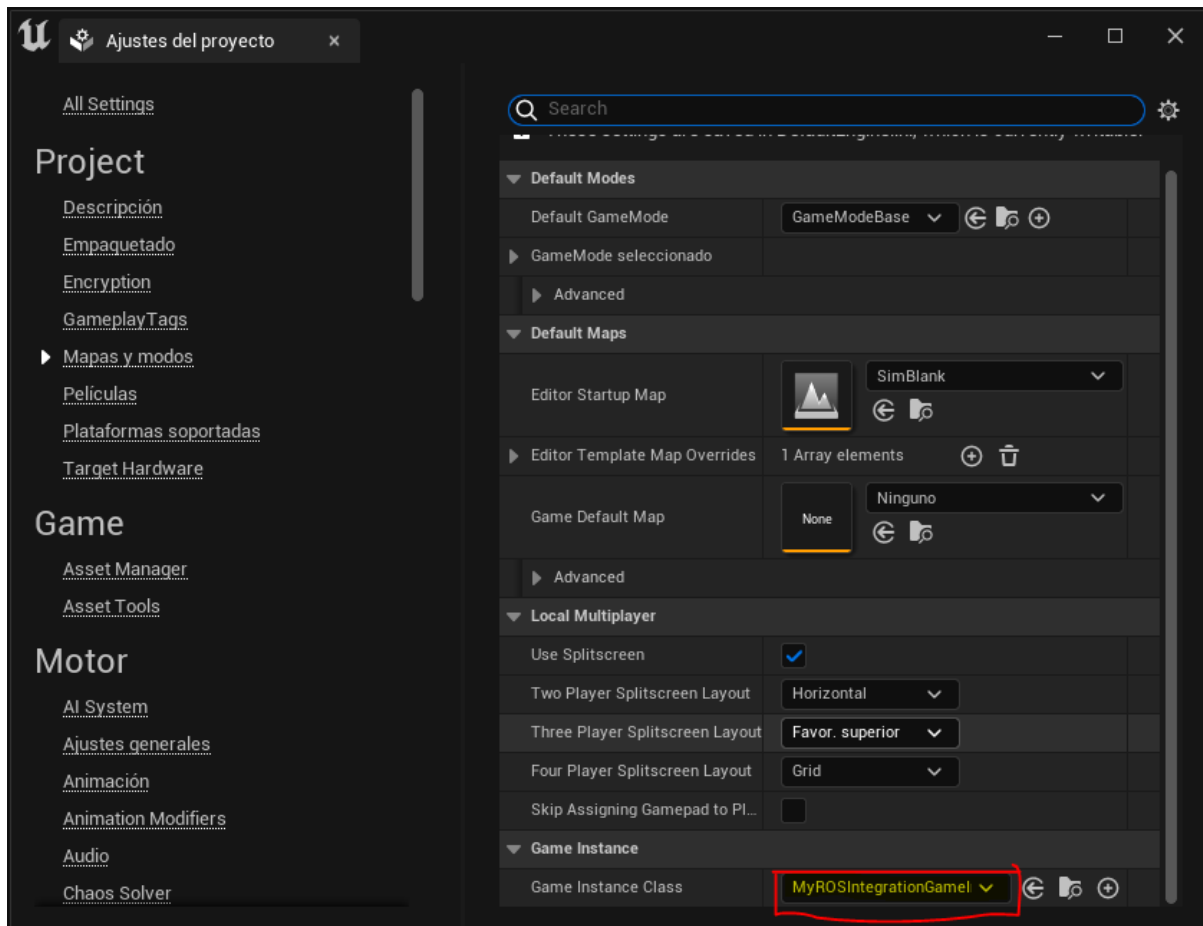
Nota: si la máquina virtual o el ordenador de ubuntu con el que nos queremos conectar está conectado directamente a internet, es decir, tiene su propia IP, entonces hay que editar el archivo “.bashrc” de ubuntu y añadir las siguientes líneas para configurar estas variables de entorno:

```
export ROS_MASTER_URI=http://[IP]:11311
export ROS_IP=[IP]
export ROS_HOSTNAME=[IP]
```

Donde [IP] es la dirección IP de la máquina donde corre ubuntu. Por ejemplo:

```
123
124 # IP CONFIG para ROS si la maquina virtual esta conectada en adaptador
125 # fuente (para la conexion a internet a traves del host) y por tanto
126 # tiene su propia IP
127 export ROS_MASTER_URI=http://192.168.2.109:11311 # ip del Master
128 export ROS_IP=192.168.2.109 # ip del esclavo
129 export ROS_HOSTNAME=192.168.2.109 # ip del esclavo
130
```

Por último, para aplicar la configuración hay que hacer click en Edit > Project Settings > Maps and Modes. Y en la parte inferior de la ventana seleccionar la Game Instance Class que acabamos de crear, en mi caso la he llamado con el nombre por defecto MyROSIntegrationGameInstance:



Recuerda guardar todo (Ctrl + Shift + S)

Establecimiento de la conexión entre RosBridge y ROSIntegration

Para testear seguiremos las instrucciones que aparecen en el repositorio de GitHub del plugin de ROSIntegration.

Para lanzar el paquete de rosbridge con la configuración necesaria para conectar con ROSIntegration en Unreal Engine hay que usar la siguiente línea de comandos en nuestra máquina virtual:

```
roslaunch rosbridge_server rosbridge_tcp.launch bson_only_mode:=True
```

Una vez hecho esto nuestro rosbridge se quedará a la espera de una conexión a través del puerto 9090 de nuestra máquina virtual.

Podemos comprobar que está funcionando abriendo el navegador en windows y escribiendo en la barra de direcciones:

```
localhost:9090
```

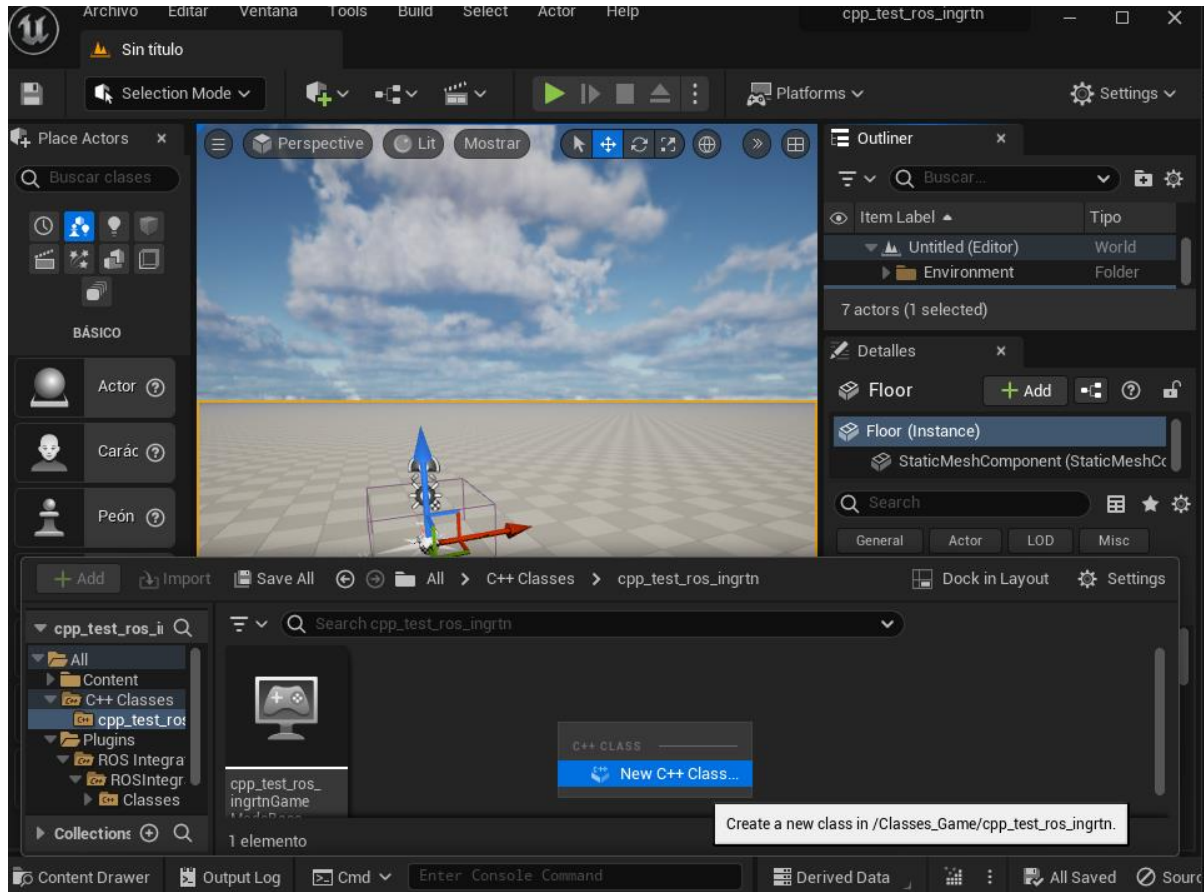
Cuando se lleva a cabo esta acción se puede comprobar que en la ventana de comandos donde hicimos roslaunch se añade un cliente.

Ahora por ejemplo crearemos un Publisher en Unreal Engine y un Subscriber en nuestra máquina virtual para comprobar que los mensajes se envían y se reciben correctamente.

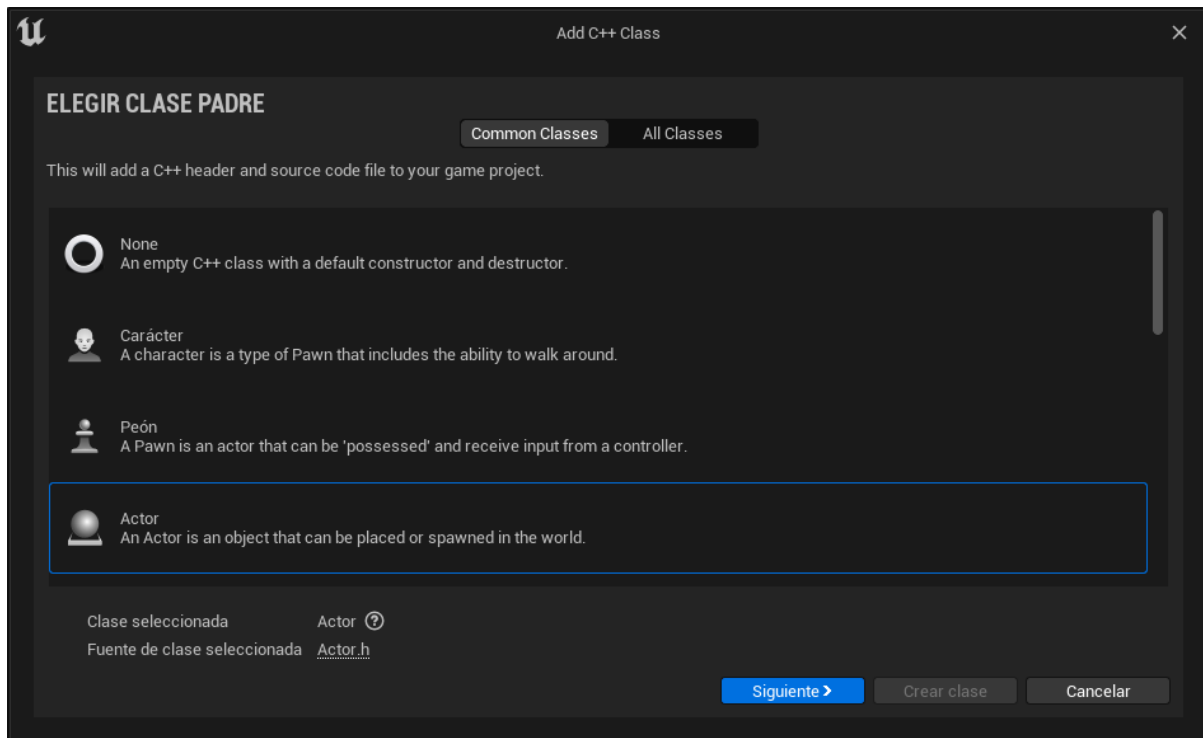
Creación de un Publisher y un Subscriber como clases de C++

Nota: se recomienda crearlos usando blueprints, para ello ver el capítulo siguiente: “Creación de un Publisher y un Subscriber con blueprints”.

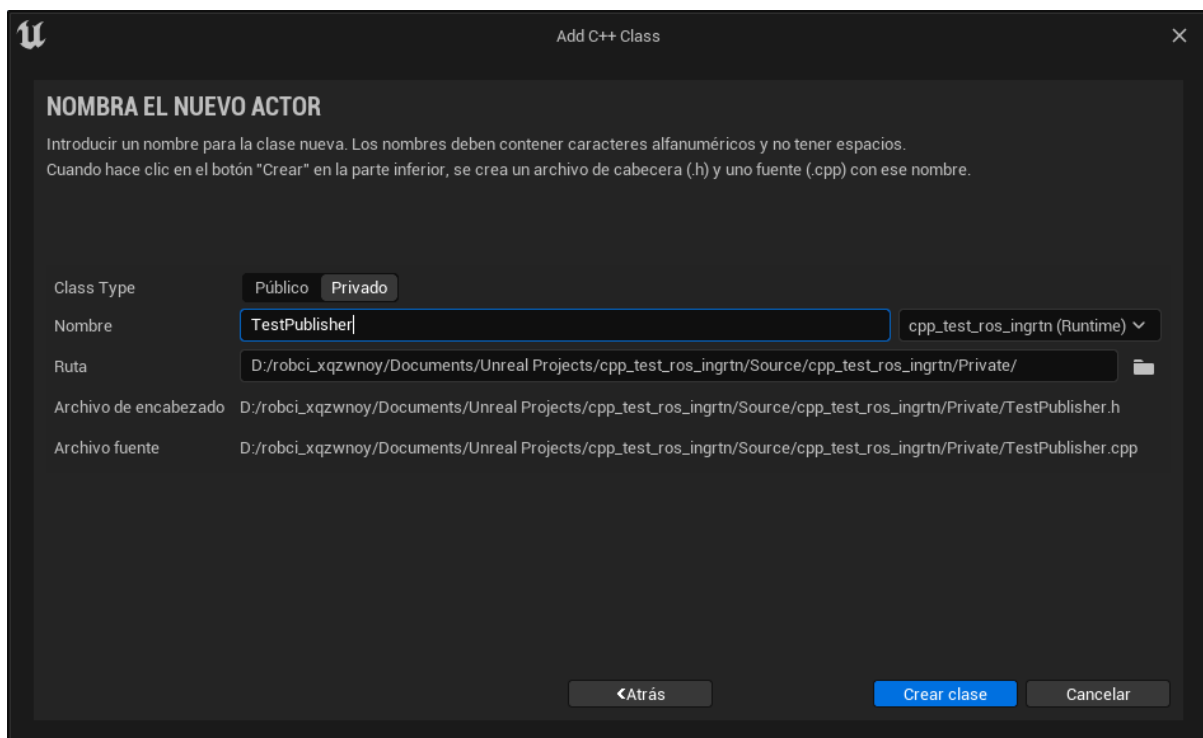
Vamos a crear un Publisher de ROS en Unreal Engine, para ello crearemos una nueva clase C++ cuya clase padre es Actor. Hacer click derecho en el Content Drawer dentro de la carpeta donde deseemos guardar la clase y seleccionar New C++ Class.



Se nos abrirá el siguiente menú y seleccionaremos Topic como clase padre:



Seleccionaremos tipo privado y el nombre:



Al hacer click en Crear clase se abrirá el proyecto en Microsoft Visual Studio y aparecerán dos nuevos archivos: TestPublisher.cpp y TestPublisher.h. Estos serán los que tendremos que editar. Al abrir Microsoft Visual Studio, Unreal Engine se pondrá a recompilar el proyecto. Debemos esperar a que termine antes de editar los archivos o si no se puede estropear el proyecto. Cuando acabe, usaremos el código de los ejemplos del github de ROSIntegration (<https://github.com/code-iai/ROSIntegration/tree/master?tab=readme-ov-file#c-topic-publish-example>) para dejar nuestros archivos así:


```
TestPublisher.cpp  TestPublisher.h
cpp_test_ros_ingrtn  → ATestPublisher

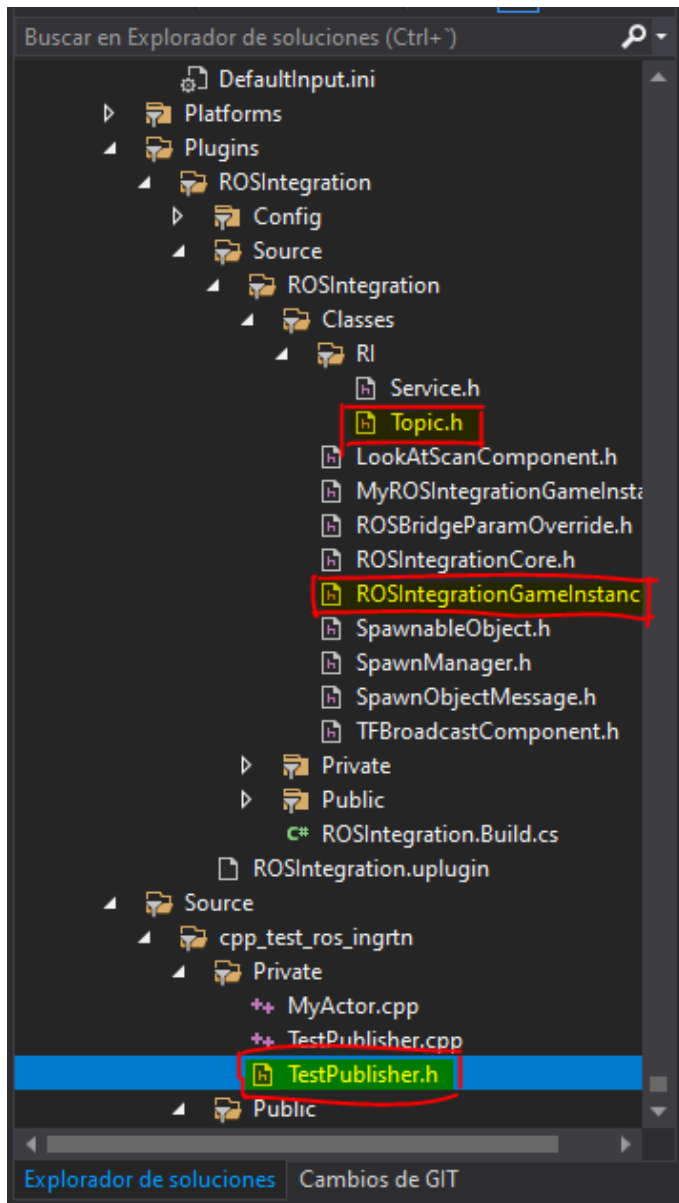
1 // Fill out your copyright notice in the Description page of Project Settings.
2
3
4 #include "TestPublisher.h"
5
6 // Sets default values
7 ATestPublisher::ATestPublisher()
8 {
9     // Set this actor to call Tick() every frame. You can turn this off to improve performance if you don't need it.
10    PrimaryActorTick.bCanEverTick = true;
11 }
12
13
14 // Called when the game starts or when spawned
15 void ATestPublisher::BeginPlay()
16 {
17     Super::BeginPlay();
18
19     // Initialize a topic
20     UTopic* ExampleTopic = NewObject<UTopic>(UTopic::StaticClass());
21     UROSIntegrationGameInstance* rosinst = Cast<UROSIntegrationGameInstance>(GetGameInstance());
22     ExampleTopic->Init(rosinst->ROSIntegrationCore, TEXT("/example_topic"), TEXT("std_msgs/String"));
23
24     // (Optional) Advertise the topic
25     ExampleTopic->Advertise();
26
27     // Publish a string to the topic
28     TSharedPtr<ROSMessages::std_msgs::String> StringMessage(new ROSMessages::std_msgs::String("This is an example"));
29     ExampleTopic->Publish(StringMessage);
30 }
31
32 // Called every frame
33 void ATestPublisher::Tick(float DeltaTime)
34 {
35     Super::Tick(DeltaTime);
36 }
37
38
39
```

El código del Publisher se ejecuta una vez cuando le damos al botón de Play this level en Unreal Engine.

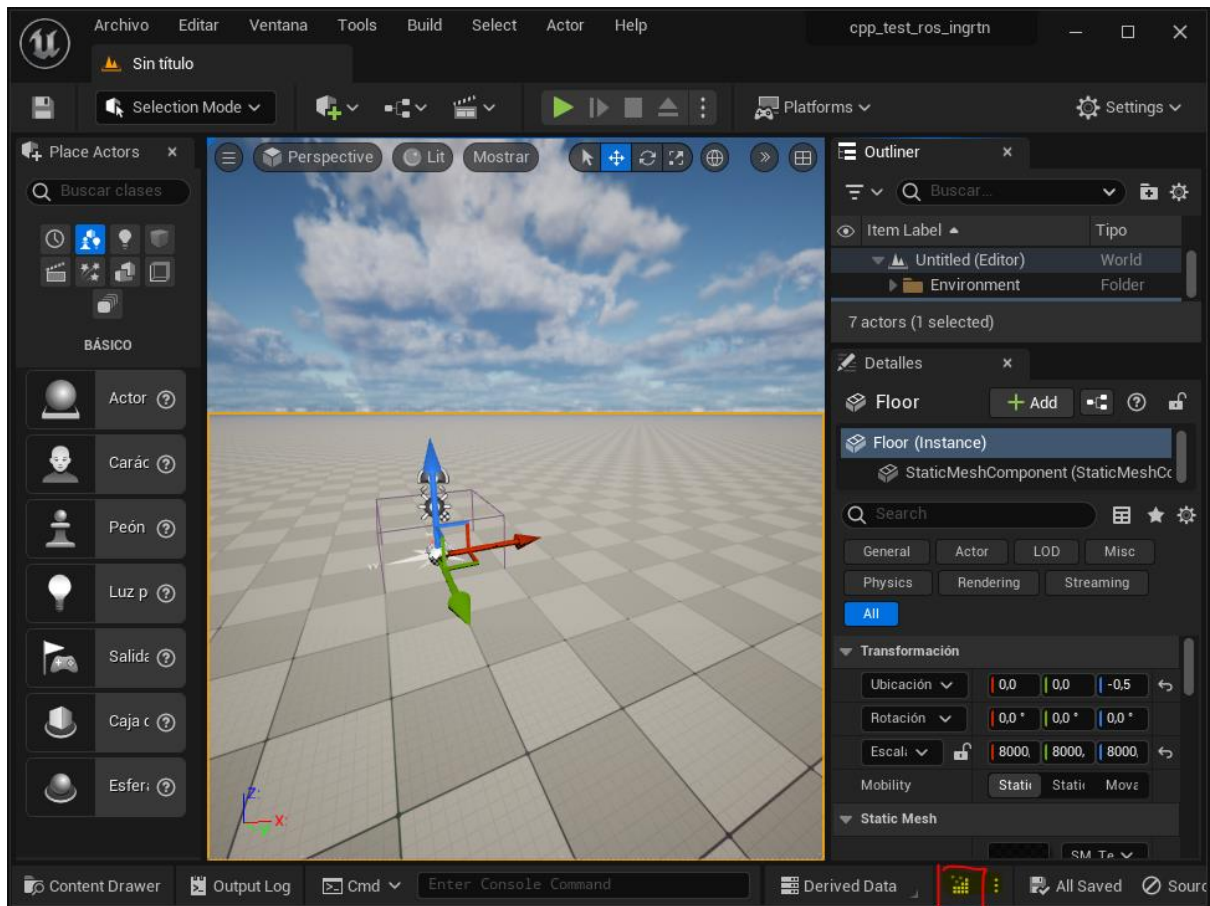
```
TestPublisher.cpp  TestPublisher.h  (Ámbito global)
cpp_test_ros_ingrtn

1 // Fill out your copyright notice in the Description page of Project Settings.
2
3 #pragma once
4
5 #include "../Plugins/ROSIntegration/Source/ROSIntegration/Classes/RI/Topic.h"
6 #include "../Plugins/ROSIntegration/Source/ROSIntegration/Classes/ROSIntegrationGameInstance.h"
7 #include "../Plugins/ROSIntegration/Source/ROSIntegration/Public/std_msgs/String.h"
8
9 #include "CoreMinimal.h"
10 #include "GameFramework/Actor.h"
11 #include "TestPublisher.generated.h"
12
13 UCLASS()
14 class ATestPublisher : public AActor
15 {
16     GENERATED_BODY()
17
18 public:
19     // Sets default values for this actor's properties
20     ATestPublisher();
21
22 protected:
23     // Called when the game starts or when spawned
24     virtual void BeginPlay() override;
25
26 public:
27     // Called every frame
28     virtual void Tick(float DeltaTime) override;
29
30 };
31
```


En el header verifica que las rutas relativas del resto de headers con respecto de TestPublisher.h son las correctas:

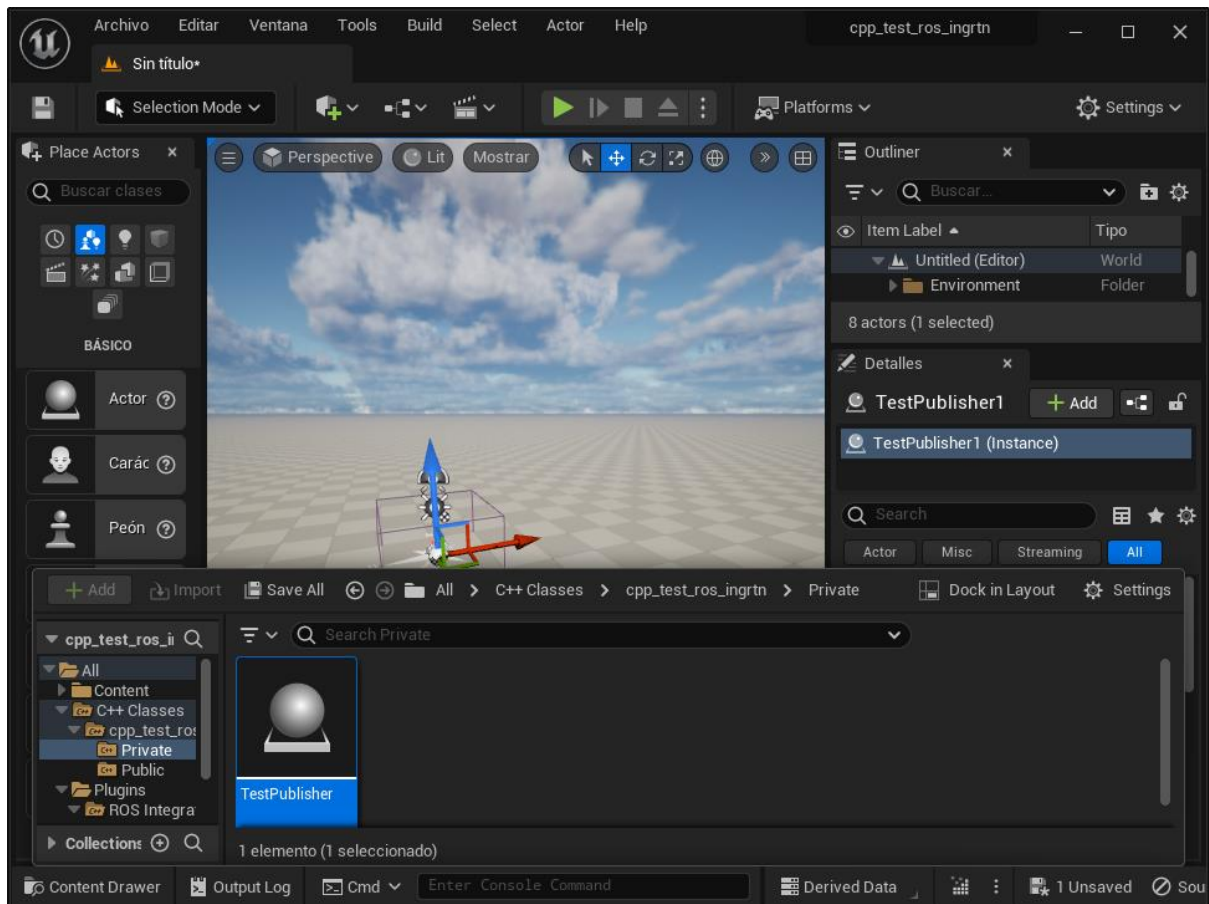


Guarda todo (Ctrl + Shift + S) y recompila el proyecto desde Unreal Engine.



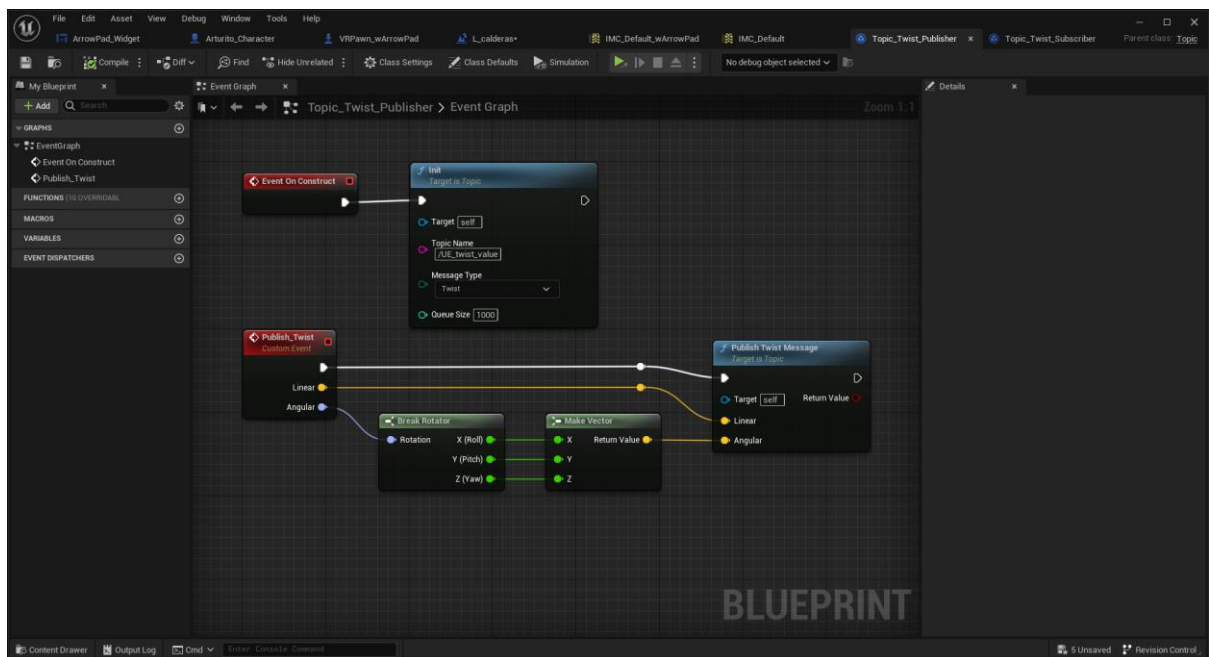
Mientras termina de compilar (puede tardar bastante ~15min) podemos crear el Subscriber en nuestra máquina virtual. Para ver un ejemplo de Subscriber en C++ ir al capítulo “Creación del Subscriber en la Máquina Virtual” de este manual.

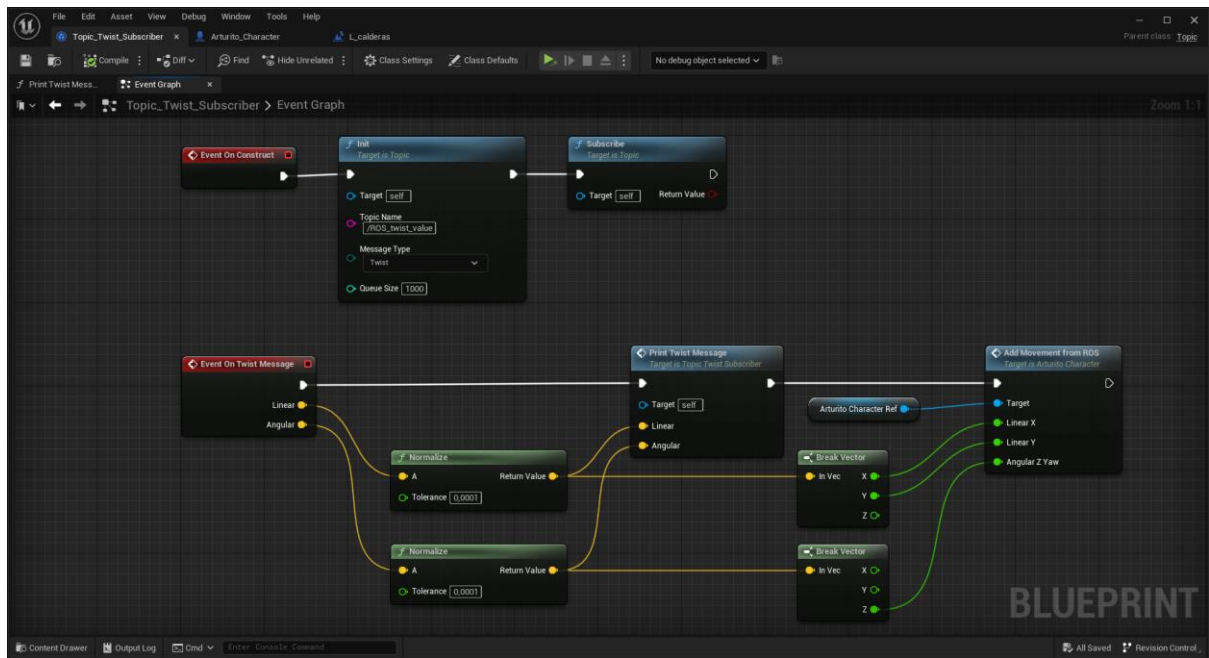
Cuando haya terminado la compilación añade un nuevo actor de tipo TestPublisher al juego. Para ello haz click sobre éste en el Content Drawer y arrástralo a la pantalla de juego.



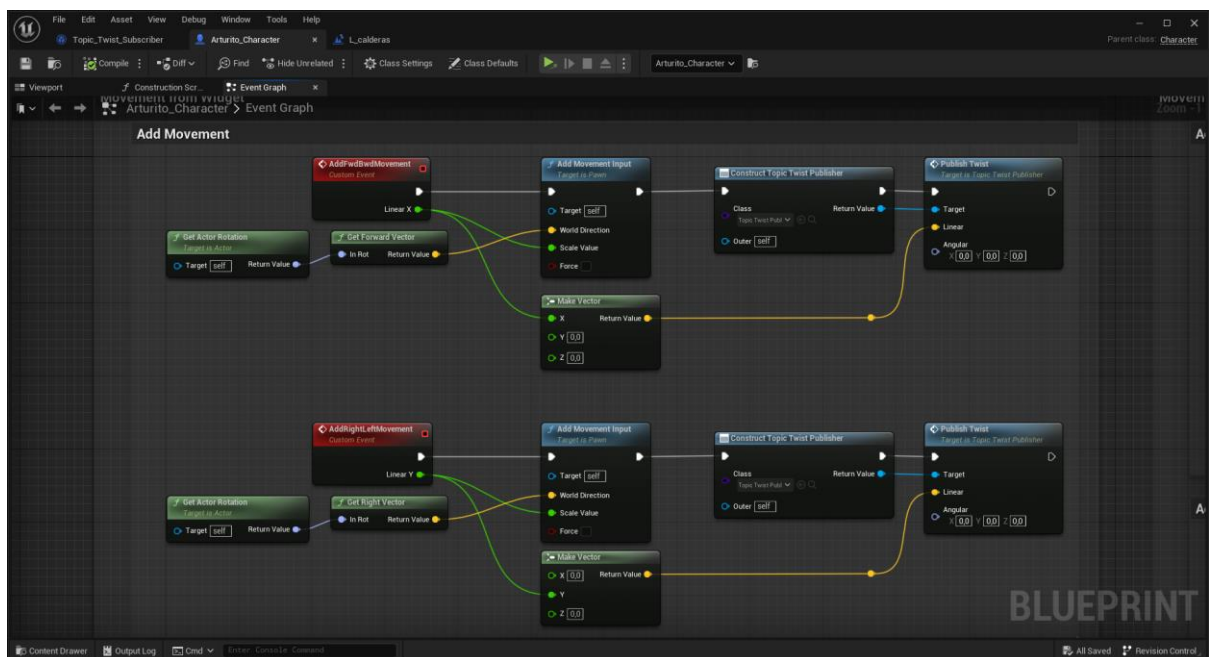
Creación de un Publisher y un Subscriber con blueprints

Otra manera de configurar los Publisher y Subscriber es creando un blueprint que herede las características de la clase Topic que incluye el plugin de ROS Integration.

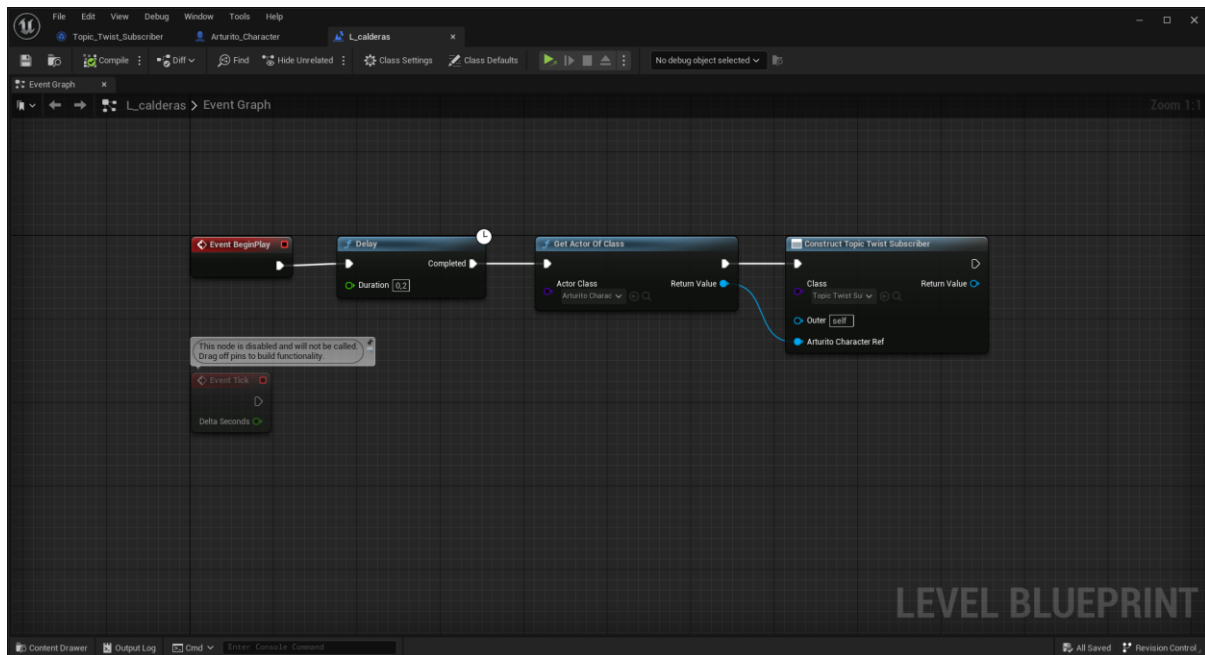




De esta forma crearemos una instancia del Publisher cada vez que queramos enviar un mensaje, por ejemplo, cuando un actor realice una acción en la simulación o de manera periódica en el blueprint del nivel.



Para el Subscriber, si no sabemos cuándo vamos a recibir el mensaje, no tenemos más remedio que crear una instancia en el blueprint del nivel.



Creación del Subscriber en la Máquina Virtual

Antes de establecer la conexión entre UE y ROS no te olvides crear un Subscriber o Publisher en Python o C++ en la máquina virtual para poder comunicarte con él. A continuación, se muestra un ejemplo de Subscriber en C++:

```

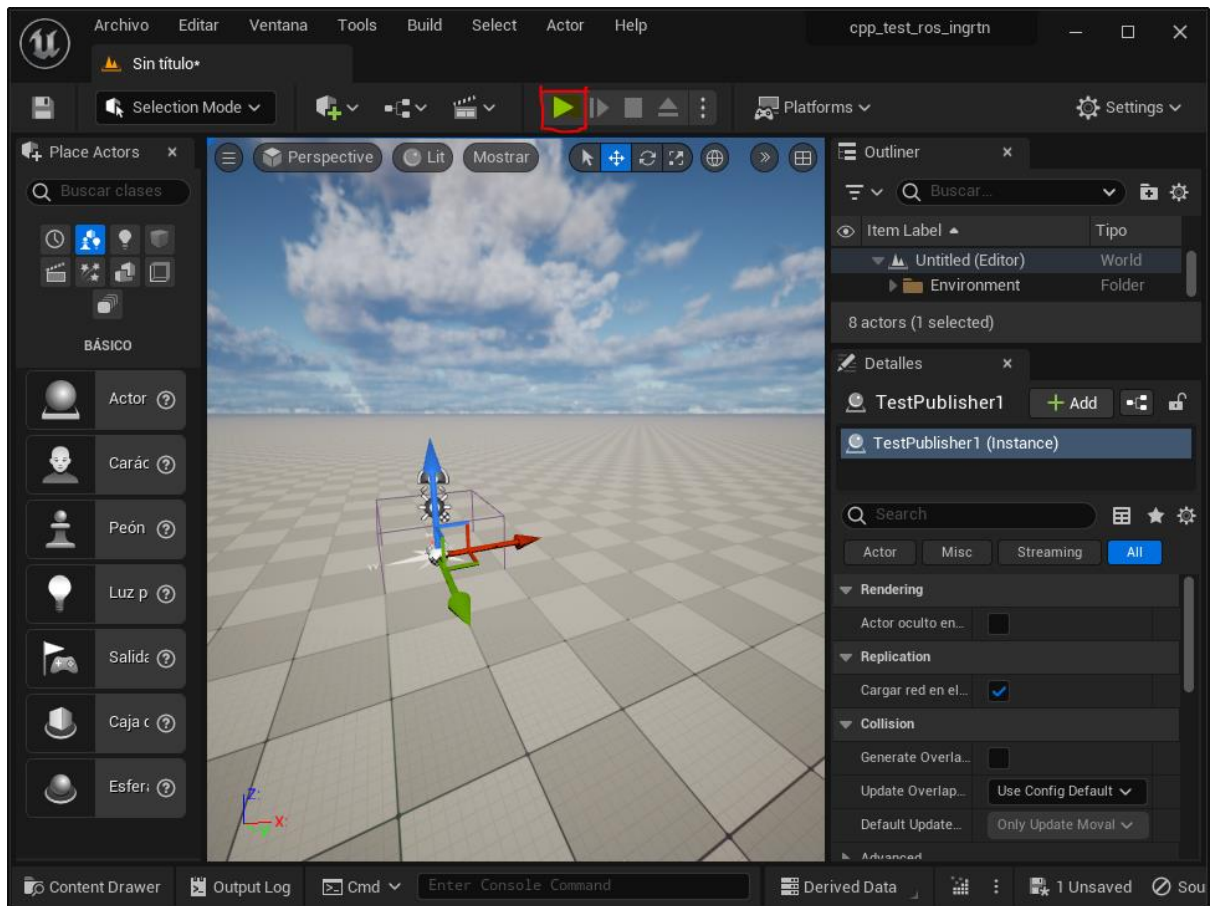
robcib@Ubuntu-20: ~/catkin_ws/src/tfg_ros_nerf_ue5/src
1 #include <ros/ros.h>
2 #include <std_msgs/String.h>
3
4 void    callback_receive_msg(const std_msgs::String &msg)
5 {
6     ROS_INFO("MESSAGE RECEIVED: %s", msg.data.c_str());
7 }
8
9 int     main(int argc, char **argv)
10 {
11     ros::init(argc, argv, "rosbridge_test");
12     ros::NodeHandle nh;
13
14     ros::Subscriber sub = nh.subscribe("/example_topic", 1000, callback_receive_msg);
15
16     ros::spin();
17 }
  
```

Uso de la conexión entre UE y ROS

Por último, compila el paquete de ROS con `catkin_make` y haz `roslaunch` del nodo. Recuerda que debes haber lanzado el ROS master usando el comando:

```
roslaunch rosbridge_server rosbridge_tcp.launch bson_only_mode:=True
```

En este momento ya estás listo para darle a play en Unreal Engine



Verifica en la ventana donde lanzaste el nodo Subscriber que efectivamente se ha recibido el mensaje. Sí es así enhorabuena, la conexión funciona correctamente.

```
/opt/ros/noetic/share/rosbridge_server/launch/rosbridge_tcp.launch http://localhost:11311
robcib@Ubuntu-20:~$ roslaunch rosbridge_server rosbridge_tcp.launch bson_only_mode:=True
... logging to /home/robcib/.ros/log/be8e7644-dbae-11ee-bc98-c7a0c602db56/roslaunch-Ubuntu-20-4
3117.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://Ubuntu-20:35513/

SUMMARY
=====

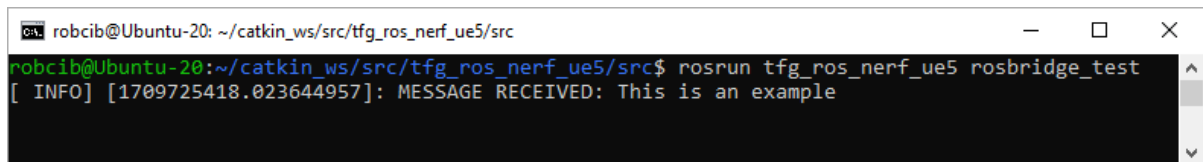
PARAMETERS
* /rosapi/params_glob: [*]
* /rosapi/services_glob: [*]
* /rosapi/topics_glob: [*]
* /rosbridge_tcp/authenticate: False
* /rosbridge_tcp/bson_only_mode: True
* /rosbridge_tcp/delay_between_messages: 0
* /rosbridge_tcp/fragment_timeout: 600
* /rosbridge_tcp/host:
* /rosbridge_tcp/incoming_buffer: 65536
* /rosbridge_tcp/max_message_size: None
* /rosbridge_tcp/params_glob: [*]
* /rosbridge_tcp/port: 9090
* /rosbridge_tcp/retry_startup_delay: 5
* /rosbridge_tcp/services_glob: [*]
* /rosbridge_tcp/socket_timeout: 10
* /rosbridge_tcp/topics_glob: [*]
* /rosbridge_tcp/unregister_timeout: 10
* /roscpp: noetic
* /roscpp: 1.16.0

NODES
/
  rosapi (rosapi/rosapi_node)
  rosbridge_tcp (rosbridge_server/rosbridge_tcp)

auto-starting new master
process[master]: started with pid [43125]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to be8e7644-dbae-11ee-bc98-c7a0c602db56
process[rosout-1]: started with pid [43135]
started core service [/rosout]
process[rosbridge_tcp-2]: started with pid [43142]
process[rosapi-3]: started with pid [43143]
registered capabilities (classes):
- <class 'rosbridge_library.capabilities.call_service.CallService'>
- <class 'rosbridge_library.capabilities.advertise.Advertise'>
- <class 'rosbridge_library.capabilities.publish.Publish'>
- <class 'rosbridge_library.capabilities.subscribe.Subscribe'>
- <class 'rosbridge_library.capabilities.defragmentation.Defragment'>
- <class 'rosbridge_library.capabilities.advertise_service.AdvertiseService'>
- <class 'rosbridge_library.capabilities.service_response.ServiceResponse'>
- <class 'rosbridge_library.capabilities.unadvertise_service.UnadvertiseService'>
trying to start rosbridge TCP server..

[INFO] [1709725407.623585]: Rosbridge TCP server started on port 9090
[INFO] [1709725417.234883]: [Client 0] connected. 1 client total.
the rosdep view is empty: call 'sudo rosdep init' and 'rosdep update'
[INFO] [1709725417.709665]: [Client 0] Subscribed to /unreal_ros/spawn_objects
[INFO] [1709725417.717837]: [Client 0] Subscribed to /unreal_ros/spawn_objects_array
```

A terminal window titled 'robcib@Ubuntu-20: ~/catkin_ws/src/tfg_ros_nerf_ue5/src'. The prompt is 'robcib@Ubuntu-20:~/catkin_ws/src/tfg_ros_nerf_ue5/src\$'. The command entered is 'roslaunch tfg_ros_nerf_ue5 rosbridge_test'. The output is '[INFO] [1709725418.023644957]: MESSAGE RECEIVED: This is an example'.

Modificación del código fuente del plugin original para poder enviar más tipos de mensajes

El código fuente del plugin original incluye la configuración necesaria para suscribirse a una gran variedad de topics haciendo uso de blueprints dentro de UE. Sin embargo, para la publicación sólo incluye por defecto el envío de topics con mensajes de tipo String.

Este es el método para añadir la publicación de mensajes de tipo Twist al blueprint de topic por defecto del plugin de ROSIntegration. Se puede proceder de manera similar para diferentes tipos de mensaje.

Hay que modificar los siguientes archivos:

[...]\ROSIntegration\Source\ROSIntegration\Classes\RI\Topic.h

[...]\ROSIntegration\Source\ROSIntegration\Private\Topic.cpp

Para la correcta definición del tipo de mensaje y conocer su equivalente en UE ver las definiciones para las clases de los mensajes en la subcarpeta correspondiente al tipo de mensaje en:

[...]\ROSIntegration\Source\ROSIntegration\Public

Por ejemplo para Twist:

[...]\ROSIntegration\Source\ROSIntegration\Public\geometry_msgs\Twist.h


```
C Twist.h  X
1  #pragma once
2
3  #include "ROSBaseMsg.h"
4
5  #include "geometry_msgs/Vector3.h"
6
7  namespace ROSMessages {
8      namespace geometry_msgs {
9          class Twist : public FROSBaseMsg {
10             public:
11                 Twist() {
12                     _MessageType = "geometry_msgs/Twist";
13                 }
14
15                 geometry_msgs::Vector3 linear;
16                 geometry_msgs::Vector3 angular;
17             };
18         }
19     }
20
```

El código a añadir en Topic.h es (nótese que es una función privada):

```

122
123 private:
124
125 struct State
126 {
127     bool Connected;
128     bool Advertised;
129     bool Subscribed;
130     bool Blueprint;
131     EMessageType BlueprintMessageType;
132 } _State;
133
134
135 UFUNCTION(BlueprintCallable, Category = "ROS|Topic")
136 void Init(const FString& TopicName, EMessageType MessageType, int32 QueueSize = 1);
137
138 /**
139  * Subscribe to the given topic
140  */
141 UFUNCTION(BlueprintCallable, Category = "ROS|Topic")
142 bool Subscribe();
143
144 UFUNCTION(BlueprintCallable, Category = "ROS|Topic")
145 bool PublishStringMessage(const FString& Message);
146
147 UFUNCTION(BlueprintCallable, Category = "ROS|Topic")
148 bool PublishTwistMessage(const FVector& linear, const FVector& angular);
149
150 // Helper to keep track of self-destruction for async functions
151 TSharedPtr<UTopic, ESPMode::ThreadSafe> _SelfPtr;
152
153 // PIMPL
154 class Impl;
155 Impl *_Implementation = nullptr;
156 };
157

```

Y el código a añadir en Topic.cpp es:

```

535
536
537 bool UTopic::PublishStringMessage(const FString& Message)
538 {
539     check(_Implementation->_MessageType == TEXT("std_msgs/String"));
540
541     if (!_State.Advertised)
542     {
543         if (!Advertise())
544         {
545             return false;
546         }
547     }
548
549     TSharedPtr<ROSMessages::std_msgs::String> msg = MakeShareable(new ROSMessages::std_msgs::String);
550     msg->_Data = Message;
551     return _Implementation->Publish(msg);
552 }
553
554
555 bool UTopic::PublishTwistMessage(const FVector& linear, const FVector& angular)
556 {
557     check(_Implementation->_MessageType == TEXT("geometry_msgs/Twist"));
558
559     if (!_State.Advertised)
560     {
561         if (!Advertise())
562         {
563             return false;
564         }
565     }
566
567     TSharedPtr<ROSMessages::geometry_msgs::Twist> msg = MakeShareable(new ROSMessages::geometry_msgs::Twist);
568     msg->linear = linear;
569     msg->angular = angular;
570     return _Implementation->Publish(msg);
571 }

```

Para poder compilar el código y poder usarlo en un proyecto de UE se debe hacer la misma operación que durante la instalación del plugin. Para evitar problemas se recomienda usar el código fuente original (que no está compilado aún en la versión 5 de UE) para realizar cualquier cambio como los indicados en las capturas anteriores. A continuación, crear un proyecto de C++ vacío en UE y añadir el plugin a éste. Cuando se añada preguntará si se desea salir y volver a entrar para aplicar los cambios. Una vez hecho esto se preguntará si se desea recompilar el plugin, darle a sí y esperar unos 15 min hasta que termine (aunque no aparezca ninguna ventana ni nada no reintentar o se romperá y la compilación no funcionará).