Build a high-level application in Linux

- 07/29/2020

A high-level application runs on the Azure Sphere OS, uses the Azure Sphere application libraries, and can communicate with the internet and with cloud-based services.
See Overview of Azure Sphere applications for basic information about high-level applications.

In this tutorial, you learn how to:

- Prepare your device for development and debugging
- Build and run a high-level application
- Reenable application updates

Prerequisites

- Install the SDK for Windows or for Linux
- Choose a tenant and claim your device
- Configure networking and update the device OS

Prepare your device for development and debugging

Before you can build a sample application on your Azure Sphere device or develop new applications for it, you must enable development and sideloading. By default, Azure Sphere devices are "locked"; that is, they do not allow applications under development to be loaded from a computer, and they do not allow debugging of applications. Preparing the device for sideloading removes this restriction.

The **azsphere device enable-development** command configures the device to accept applications for debugging, loads the debugging server onto the device, and assigns the device to a device group that does not allow cloud application updates. During application development and debugging, you should leave the device in this group so that cloud application updates do not overwrite the application under development.

1. Make sure that your Azure Sphere device is connected to your computer, and your computer is connected to the internet.
2. Open an Azure Sphere Developer Command Prompt (Windows) or a terminal window (Linux).
3. Enter the following command:

```
azsphere device enable-development
```

You should see output similar to the following:

Copy

```
Getting device capability configuration for application development.
Downloading device capability configuration for device ID '<device ID>'.
Successfully downloaded device capability configuration.
Successfully wrote device capability configuration file
'C:\Users\user\AppData\Local\Temp\tmpD732.tmp'.
Setting device group ID 'a6df7013-c7c2-4764-8424-00cbacb431e5' for device
with ID '<device ID>'.
Successfully disabled over-the-air updates.
Enabling application development capability on attached device.
Applying device capability configuration to device.
Successfully applied device capability configuration to device.
The device is rebooting.
Installing debugging server to device.
Deploying 'C:\Program Files (x86)\Microsoft Azure Sphere
SDK\DebugTools\gdbserver.imagepackage' to the attached device.
Image package 'C:\Program Files (x86)\Microsoft Azure Sphere
SDK\DebugTools\gdbserver.imagepackage' has been deployed to the attached
device.
Application development capability enabled.
Successfully set up device '<device ID>' for application development, and
disabled over-the-air updates.
Command completed successfully in 00:00:38.3299276.
```

If the **azsphere device enable-development** command fails with the following error message, see Troubleshoot Azure Sphere issues for help.

```
error: The device did not accept the device capability configuration. Please check
the Azure Sphere OS on your device is up-to-date using 'azsphere device show-
deployment-status'.
```

Build the sample on the command line

To create the build and .imagepackage files for the HelloWorld_HighLevelApp sample application, follow these steps.

1. Install Git.
2. Clone the entire Azure Sphere samples repo:

```
git clone https://github.com/Azure/azure-sphere-samples.git
```

3. Create or navigate to the directory that will contain the build .imagepackage files that will be generated during the build process. For example, to create and open a new directory called "buildfiles" you would enter the following commands:

   shCopy
   ```
   mkdir buildfiles
   cd buildfiles
   ```

4. Update the sample to target your hardware, if necessary. By default, the samples target hardware that follows the MT3620 reference board design (RDB), such as the MT3620 Development Kit from Seeed Studios. Additional target hardware definitions for the sample applications are available in the HardwareDefinitions directory of the Azure Sphere Samples repo. For example, the hardware definition files for the Avnet MT3620 Starter Kit are in the HardwareDefinitions/avnet_mt3620_sk subdirectory.
   - Open CMakeLists.txt and update the TARGET_DIRECTORY parameter in the **azure_target_hardware_definition** function to point at the subdirectory for your hardware. For example:

     ```
     azsphere_target_hardware_definition(${PROJECT_NAME}
     TARGET_DIRECTORY
     "../../../HardwareDefinitions/avnet_mt3620_sk"
     TARGET_DEFINITION "sample_appliance.json")
     ```

5. Run CMake. Set the final parameter to the pathname of the directory that contains the source files for the HelloWorld_HighLevelApp sample application on your local machine.

   ```
   cmake \
    -G "Ninja" \
    -
   DCMAKE_TOOLCHAIN_FILE="/opt/azurespheresdk/CMakeFiles/AzureSphereToolchai
   n.cmake" \
    -DAZURE_SPHERE_TARGET_API_SET="latest-lts" \
    -DCMAKE_BUILD_TYPE="Debug" \
    <path to cloned samples repo>/azure-sphere-
   samples/Samples/HelloWorld/HelloWorld_HighLevelApp
   ```

6. Run ninja to build the application and create the image package file.

   ```
   ninja
   ```

Run the sample

1. If your device is already running an application, delete the application:

   ```
   azsphere device sideload delete
   ```

2. Change to the directory that contains the build and .imagepackage files created previously.
3. Load the image package onto your device by running the **azsphere device sideload deploy** command with the --imagepackage option. For example:

   ```
   azsphere device sideload deploy --imagepackage
   HelloWorld_HighLevelApp.imagepackage
   ```

   This command loads the image package and starts the application. You should see an LED blink.

Tip

Note the path of the image package. You'll use the image package later in the **Deployment Quickstart**.

Debug the sample

1. Change to the directory that contains the build and .imagepackage files created previously.
2. Get the component ID if you don't already have it:

   ```
   azsphere image-package show -f HelloWorld_HighLevelApp.imagepackage
   ```

3. If the application is running, stop it and then restart it with the --debug option:

   ```
   azsphere device app stop --componentid <ComponentId>
   ```

   ```
   azsphere device app start --debug --componentid <ComponentId>
   ```

   You should see:

   shCopy

   ```
   <ComponentId>
   App state   : debugging
   ```

```
GDB port    : 2345
Output port : 2342
Core        : High-level

Command completed successfully in 00:00:00.9121174.
```

4. Use any terminal client to read the output stream from the process. Specify 192.168.35.2 as the IP address and 2342 as the port.
5. Open another Azure Sphere Developer Command Prompt window (Windows) or terminal window (Linux). Start the **gdb** command-line debugger and pass the .out app binary from your build as a parameter. This will enable full source code debugging.

   ```
   $ /opt/azurespheresdk/Sysroots/5/tools/sysroots/x86_64-pokysdk-
   linux/usr/bin/arm-poky-linux-musleabi/arm-poky-linux-musleabi-gdb
   HelloWorld_HighLevelApp.out
   ```

6. Set the remote debugging target to IP address 192.168.35.2 on port 2345:

   ```
   target remote 192.168.35.2:2345
   ```

7. Issue whatever other gdb commands you choose. For example, the following commands set a breakpoint upon entry to main() and then continue execution after the breakpoint, respectively.

   ```
   break main

   c
   ```

   For more information about debugging with **gdb**, see GDB: The GNU Project Debugger or one of the other numerous sources on the subject.