

Alveo Accelerator Card Tutorial

 nimbix.net/alveo-fpga-tutorial

- Today we will look at how to utilize FPGAs to accelerate compute workloads and how to create a JARVICE™ application using the [PushToCompute™](#) CI/CD pipeline.

The JARVICE platform is a one stop shop for FPGA kernel development, testing, and deployment. The NIMBIX [App Marketplace](#) includes the latest Xilinx SDAccel Development environment to design FPGA kernels using OpenCL, C/C++, and RTL. We have also purpose built JARVICE to handle a [variety](#) of provisioning schemes to protect 3rd party IP from the end user.

Xilinx SDAccel

SDAccel is a high-level synthesis tool that abstracts away low level details of a hardware platform. This allows software developers to leverage the power of FPGAs without extensive knowledge of the underlying platform. For more information on SDAccel, [see here](#).

For simplicity, we will use a few OpenCL example kernels provided by Xilinx on [GitHub](#).

OpenCL Example [vadd]

A JARVICE account is required to use the NIMBIX cloud. Sign up [here](#)

Limited number of trials available for Alveo FPGA platform

To start a new SDAccel session:

- Login into [JARVICE](#)
- Select **Xilinx SDAccel Development (2018.2 XDF)**. **Note: different from the runtime environment**



The SDAccel™ development environment for OpenCL™, C, and C++, enables up to 25X better performance/watt for data center application acceleration leveraging FPGAs. SDAccel, member of the SDx™ family, combines the industry's first architecturally optimizing compiler supporting any combination of OpenCL, C, and C++ kernels, along with libraries, deve ...

[Desktop Mode](#)[Batch Mode](#)

Start Desktop Mode

Desktop Mode

Run and connect via browser or SSH.

[GENERAL](#)[OPTIONAL](#)[STORAGE](#)[PREVIEW SUBMISSION](#)

Machine

Machine type

Cores

\$1.25/hr

[SUBMIT](#)

Click on **desktop preview** to open session in a web browser



Open a **Terminal** from the Desktop and clone the SDAccel Examples

```
git clone --depth=1 https://github.com/Xilinx/SDAccel_Examples
cd SDAccel_Examples/getting_started/misc/vadd/src
ls
```

You will see two types of source files:

- **host.cpp** := C++ source code for host application running on the CPU
- **krnl_vadd.cl** := OpenCL source code for accelerated kernel running on the FPGA

The host code uses the OpenCL API to interact with the accelerated kernel. [Reference card.](#)

The SDAccel Development environment will compile the `*.cl` code into an `*.xclbin` file for the targeted FPGA platform. The `*.xclbin` file contains a bitstream for the FPGA which will describe the kernels specialized architecture. This process will take several hours depending on the size of the FPGA. To aid in kernel development, the SDAccel Development environment includes a CPU and Hardware emulation mode which compiles in minutes and can run without access to a FPGA accelerator.

To build the **vadd** example, specify the target mode (emulation vs hardware) and target device. For example:

```
cd ~/SDAccel_Examples/getting_started/misc/vadd
make TARGETS=sw_emu DEVICES=xilinx_u250_xdma_201820_1 all
```

The above command will generate `vadd` and

`krnl_vadd.sw_emu.xilinx_u250_xdma_201820_1.xclbin` for software emulation of the Xilinx Alveo u250. Prepare the software emulation environment and run **vadd** example:

```
export XCL_EMULATION_MODE=sw_emu
emconfigutil --platform 'xilinx_u250_xdma_201820_1' --nd 1
./vadd
```

Stop your job using either shutdown from the Desktop menu (logout -> shutdown) or the shutdown button on the [JARVICE dashboard](#)

Alveo options for SDAccel

Flag	Options
TARGETS	<code>sw_emu</code> , <code>hw_emu</code> , <code>hw</code>
DEVICES	<code>xilinx_u200_xdma_201820_1</code> , <code>xilinx_u250_xdma_201820_1</code>

Additional information on using SDAccel Development environment on Nimbix cloud [here](#)

Create FPGA Application on JARVICE

This section will go over how to package the binary files generated by SDAccel into a JARVICE application. JARVICE apps use the [PushToCompute](#) CI/CD pipeline. This walk-through will use Der to containerize our application with the Xilinx Runtime and give an overview of the [Appdef.json](#) required by JARVICE.

This section is intended to use your computer. Review prerequisites [here](#). Use App `Ubuntu Linux for Intel` if running on the Nimbix cloud.

To Start, clone this [repository](#) from GitHub:

```
git clone https://github.com/nimbix/xilinx-tutorial
```

Our application will require a few SDAccel kernels to illustrate the different capabilities of JARVICE. Use the attached `xilinx_u250_xdma_201820_1_golden.tar.gz` or generate the kernels using [Build SDAccel bitstreams](#)

Unpack the archive at `xilinx-tutorial/docker-build/`

```
mv xilinx_u250_xdma_201820_1_golden.tar.gz <path-to-repo>/xilinx-tutorial/docker-build/
cd <path-to-repo>/xilinx-tutorial/docker-build/
tar -xvf xilinx_u250_xdma_201820_1_golden.tar.gz
```

This will create an `exe` and `xclbin` folder

Build SDAccel bitstreams

Skip this section if using `xilinx_u250_xdma_201820_1_golden.tar.gz`

The `build-scripts/build-xcl-examples.sh` uses the [JARVICE API](#) to submit a build job using the [Xilinx SDAccel Development](#) environment. The default kernels to build are `sum_scan` and `vdotprod` from [getting_started/misc](#) section. Update `repo_path` and `kernels` at the beginning of the script to select different kernel from [Xilinx SDAccel Examples](#). The `kernels` string uses `|` as a delimiter.

Use the following to build the default SDAccel kernels for the Alveo u250:

```
./build-scripts/build-xcl-examples.sh -t hw -d xilinx_u250_xdma_201820_1 -u <jarvice-user> -k <jarvice-apikey>
```

Replace `<jarvice-user>` and `<jarvice-apikey>` with your JARVICE account. Your API key is listed at <https://platform.jarvice.com> under the [Account](#) menu on the right.

Note We need to build with the `hw` SDAccel flow. This job will take 6+ hours and automatically terminate

After submitting a job to JARVICE, the script will ask to copy/paste a password:

```
Started JARVICE job: 463856  
Enter this password at prompt: MCAqHDS02aEgj3w
```

This will transfer and run a build script (`run.sh`) on the JARVICE job. The generated files will be saved to your Vault

```
Job 463856 building /data/xcl_pv3/xilinx_u250_xdma_201820_1.tar.gz  
Check JARVICE dashboard for status
```

Transfer the `*.tar.gz` file to your machine using one of [these methods](#)

Create Docker container

The `build-scripts/build-docker.sh` will create a Docker container for a JARVICE application utilizing a Xilinx FPGA machine type. `docker-build` contains the `Dockerfile` and build context for the container. A JARVICE application requires additional metadata provided by `AppDef.json`. The FPGA machine types will also require `*.xclbin` file for each SDAccel kernel. The `docker-build` directory should include `exe` and `xclbin` directories from [Build SDAccel bitstreams](#).

[See additional information using FPGAs on JARVICE](#)

The Dockerfile is based from [Xilinx Runtime](#) available on DockerHub.

```
FROM nimbix/ubuntu-xrt:<xrt-runtime>
```

The FPGA `*.xclbin` files are added to `/opt/example` inside the container

```

# Test FPGA bitstream
ADD xclbin/$DSA/$XCLBIN_PROGRAM /opt/example/test.xclbin
RUN chmod 555 /opt/example/test.xclbin
# Add additional FPGA bitstream to demo removal of unused kernels
ADD xclbin/$DSA/$XCLBIN_REMOVE /opt/example/remove1.xclbin
ADD xclbin/$DSA/$XCLBIN_REMOVE /opt/example/remove2.xclbin
# Test host application
ADD exe/vdotprod /opt/example/vdotprod

```

The following tags in the `Dockerfile` are replaced by `build-scripts/build-docker.sh` :

Tag	Description	Sample Value
<code><xrt-runtime></code>	Xilinx Runtime version	201802.2.1.83_16.04
<code><jarvice-machine></code>	JARVICE FPGA machine type	nx6u
<code><xcl-dsa-name></code>	SDAccel DSA	xilinx_u250_xdma_201820_1

The default container registry is [DockerHub](#). `build-scripts/build-docker.sh` script will build a Docker container and push it to a Docker registry.

Note Pushing to a DockerHub repository that does not exist will create a **public** repository

```

docker login
./build-scripts/build-docker.sh <docker_repo> <docker_tag>

```

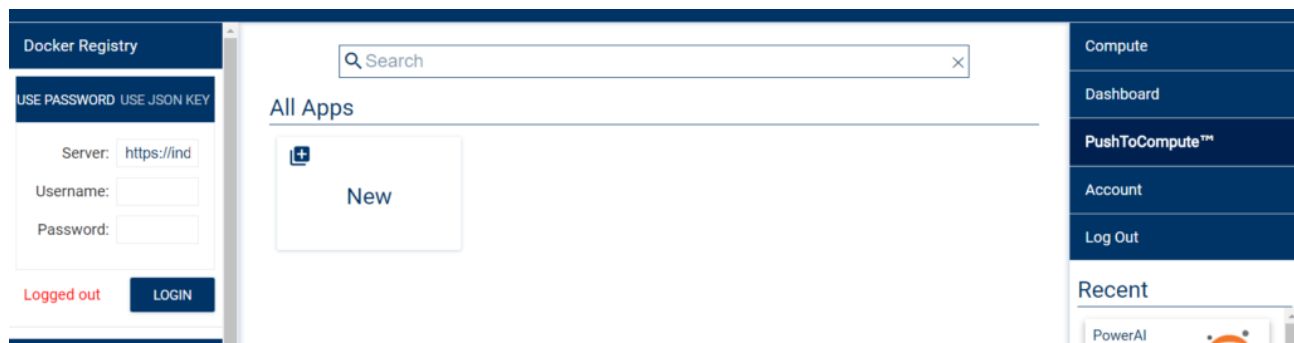
`<docker_repo>` e.g. nimbix/xilinx-tutorial

`<docker_tag>` e.g. latest


PushToCompute flow


PushToCompute is the final step to create our JARVICE application.

- Login into [JARVICE](#)
- Select `PushToCompute` from the menu on the right
- Login into your Docker registry from the menu on the left



Click on the `New` app button and fill out the form

 Create Application
✕


CHANGE ICON

App ID:

Docker Repository:

Git Source URL (to Clone) - required only for building:

System Architecture

Intel x86 64-bit (x86_64) ▾

Team Visible

"Team Visible" setting applies only to private applications. If the application is public, this setting is ignored, since public applications cannot be hidden from team members.

OK

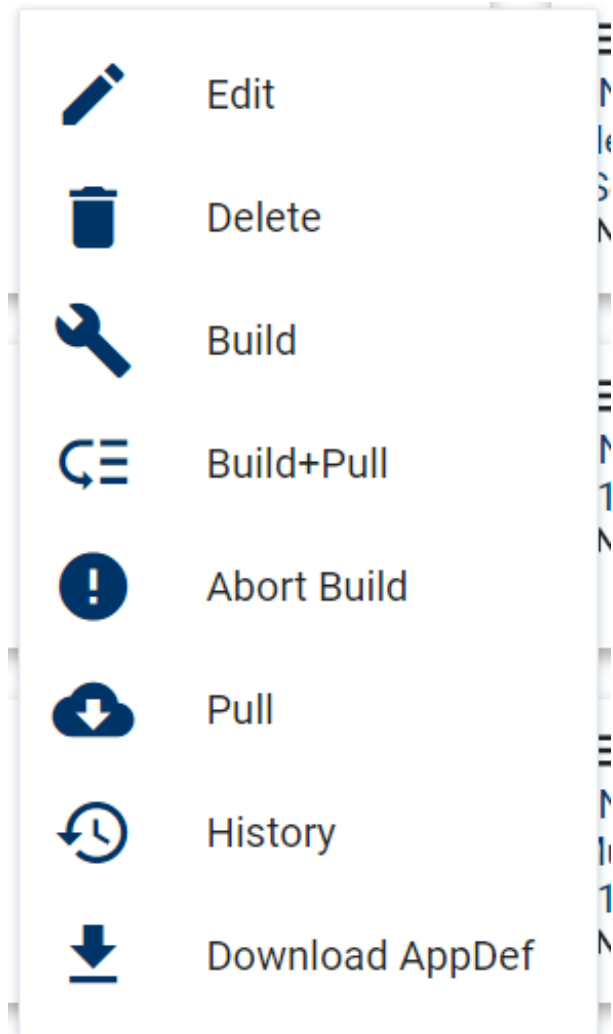
Form	Value
App ID	<code>xilinx_tutorial</code>
Docker or Singularity Repository	Docker repository used in Create Docker container (e.g. nimbix/xilinx-tutorial:latest)
Git Source URL (to Clone)	Leave blank
System Architecture	<code>Intel x86 64-bit (x86_64)</code>

This will create a new App card on PushToCompute that is private to your account (or team if `Team Visible` was selected).

Click on the menu on the top left of the app card

Click on **Pull** to start pull from your Docker Registry

- Use the same menu to check on pull progress from **History**
- Close the **Pull History** when you see **Pull completed**



Test FPGA Application JARVICE

Our application is now ready to run on JARVICE. This simple application will run the **vdotprod** example from the Xilinx **SDAccel_Examples** repository on GitHub. The Dockerfile adds the examples files generated from SDAccel to **/opt/example**.

```
/opt/example/remove1.xclbin  
/opt/example/remove2.xclbin  
/opt/example/test.xclbin  
/opt/example/vdotprod  
/opt/example/run-test.sh
```


The `*.xclbin` files are used to configure the FPGA with the desired kernel, `vdotprod` binary is the CPU executable, and `run-test.sh` is a simple exerciser script for our example.

This application supports 3 flows: `No Xclbin Protection` , `Single Xclbin Protection` , and `Launch vdotprod w/ Xclbin Protection`

Click on the app card from `PushToCompute` to see the different flows



N Xilinx FPGA Protection Test
From \$3.00/hr

NIMBIX

Xilinx FPGA Protection Test
Nimbix, Inc.

Ubuntu Linux w/ Xilinx FPGA runtime

Launch Vdotprod W/ Xclbin Protection Single Xclbin Protection No Xclbin Protection

No Xclbin Protection (standard)

The standard FPGA flow passes all `*.xclbin` files untouched to the user's session. The FPGA can then be configured by the user via the Xilinx runtime.

```
Terminal - nimbix@JARVICENAE-0A0A1883: ~
File Edit View Terminal Tabs Help
nimbix@JARVICENAE-0A0A1883:~$ ls -lh /opt/example/
total 128M
-rw-r--r-- 1 root root 46M Nov 6 16:44 remove1.xclbin
-rw-r--r-- 1 root root 46M Nov 6 16:44 remove2.xclbin
-rwxrwxrwx 1 root root 691 Nov 16 23:14 run-test.sh
-r-xr-xr-x 1 root root 37M Nov 6 16:44 test.xclbin
-rwxr-xr-x 1 root root 697K Oct 31 15:21 vdotprod
nimbix@JARVICENAE-0A0A1883:~$
```

Notice all the `*.xclbin` files are in the user's session and are ~40MB in size. The file size indicates the FPGA bitstream is available.

Single Xclbin Protection

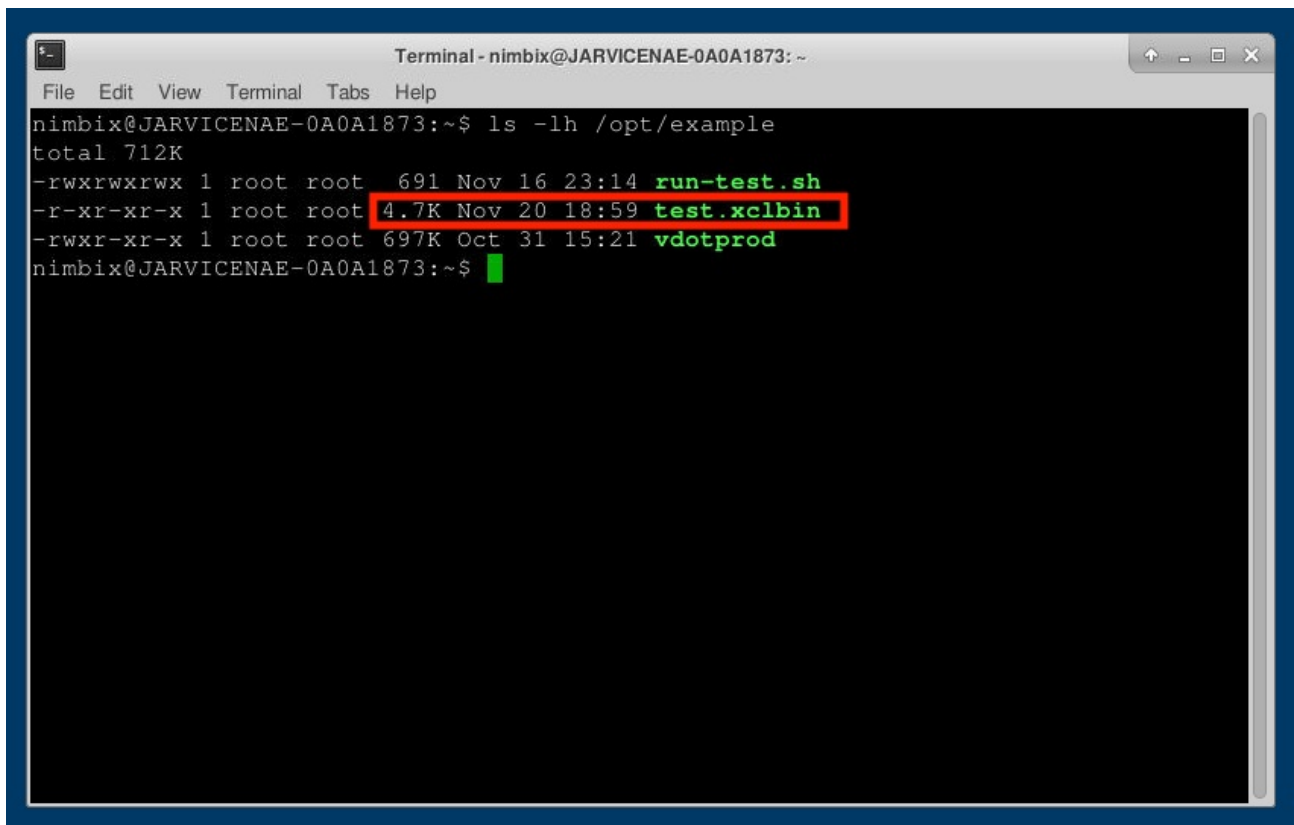
This flow uses a JARVICE platform feature to Protect kernel bitstream. The `Protect` command in the `Appdef.json` sets `XCLBIN_BITSTREAM_PROGRAM` to `/opt/example/test.xclbin`. This instructs the JARVICE platform to configure the FPGA w/ the bitstream contained in `/opt/example/test.xclbin` while provisioning a user's job. The kernel bitstream is then removed from `test.xclbin` before granting a user access to the job session.

```
Terminal - nimbix@JARVICENAE-0A0A1854:~
File Edit View Terminal Tabs Help
nimbix@JARVICENAE-0A0A1854:~$ ls -lh /opt/example/
total 92M
-rw-r--r-- 1 root root 46M Nov 6 16:44 remove1.xclbin
-rw-r--r-- 1 root root 46M Nov 6 16:44 remove2.xclbin
-rwxrwxrwx 1 root root 691 Nov 16 23:14 run-test.sh
-r-xr-xr-x 1 root root 4.7K Nov 20 19:05 test.xclbin
-rwxr-xr-x 1 root root 697K Oct 31 15:21 vdotprod
nimbix@JARVICENAE-0A0A1854:~$
```

Notice the file size for `test.xclbin` after the FPGA bitstream has been removed by the JARVICE platform. This prevents a user from accessing/copying a kernel bitstream from the application.

Launch vdotprod w/ Xclbin Protection

The final flow extends `Single Xclbin Protection` by removing all unused `*.xclbin` files. This is done by adding the `XCLBIN_BITSTREAM_PROTECT` variable to the `Workflow` command in `Appdef.json`. The `|` character is used as a delimiter to specify multiple files to remove.



```
Terminal - nimbix@JARVICENAE-0A0A1873: ~
File Edit View Terminal Tabs Help
nimbix@JARVICENAE-0A0A1873:~$ ls -lh /opt/example
total 712K
-rwxrwxrwx 1 root root 691 Nov 16 23:14 run-test.sh
-r-xr-xr-x 1 root root 4.7K Nov 20 18:59 test.xclbin
-rwxr-xr-x 1 root root 697K Oct 31 15:21 vdotprod
nimbix@JARVICENAE-0A0A1873:~$
```

The `vdotprod` example can be run from any flow with:

```
/opt/example/run-test.sh
```

Conclusion

This post provided a brief introduction on developing accelerated kernels for FPGAs available on the NIMBIX cloud. NIMBIX has partnered with Xilinx to offer a one stop shop to develop and deploy FPGA accelerated applications using the SDAccel Development environment and leveraging the power of the JARVICE platform. In addition to focusing on performance, we have extended JARVICE to optionally protect 3rd party IP when using FPGA accelerators. This added security enables ISVs to offer proprietary accelerated code in the Nimbix application marketplace.

Get started accelerating applications with the JARVICE platform on the Nimbix cloud today:

- Sign up [here](#)
- [*Limited number of Alveo™ trials available*](#)
- [Alveo tutorial](#)