



inaccel

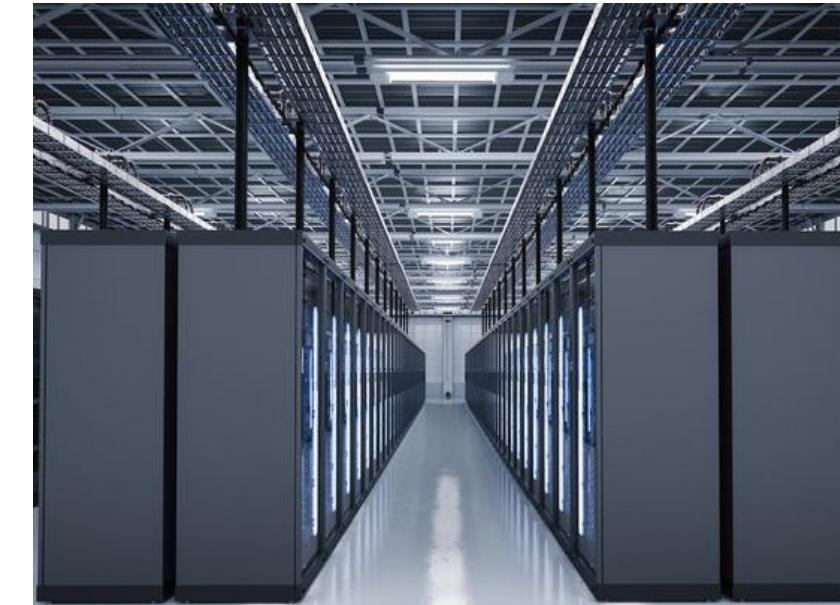
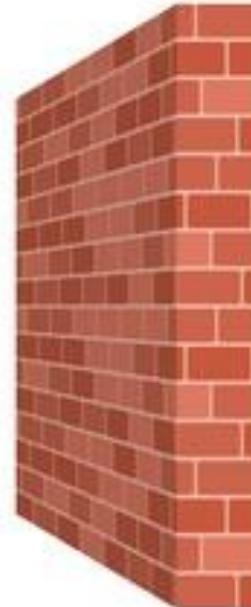
Instant FPGA deployment and scaling on the cloud

*Chris Kachris
Elias Koromilas
Ioannis Stamelos*

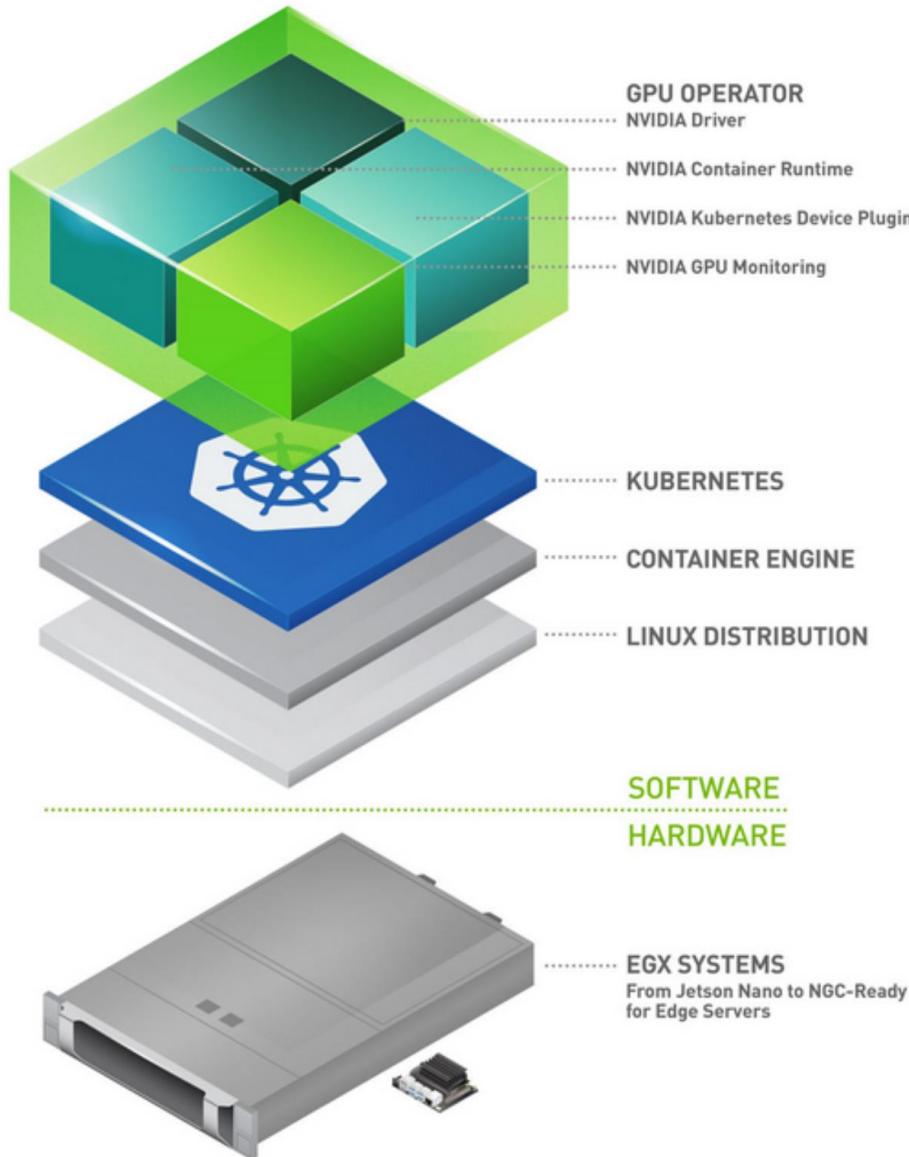
Main challenges on FPGA Deployment



- > What prevents the wide deployment of FPGA on Data centers/clusters



GPUs in the cloud



> Full ecosystem for easy deployment



kubernetes

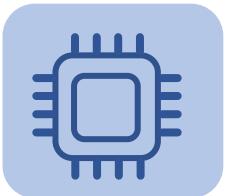
MicroK8s



CPU – GPU - FPGAs

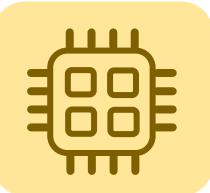


CPU



```
int threads = 100;
int id = 100;
#pragma omp parallel
{
    threads = omp_get_num_threads()
    id = omp_get_thread_num()
    std::cout << "hello from", id ;
}
return 0;
```

GPU



```
int main(void)
{
    int N = 1<<20;
    float *x, *y, *d_x, *d_y;
    x = (float*)malloc(N*sizeof(float));
    y = (float*)malloc(N*sizeof(float));

    cudaMalloc(&d_x, N*sizeof(float));
    cudaMalloc(&d_y, N*sizeof(float));

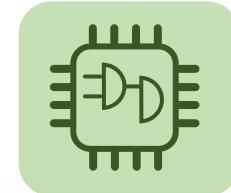
    for (int i = 0; i < N; i++) {
        x[i] = 1.0f;
        y[i] = 2.0f;
    }

    cudaMemcpy(..., ...);
    cudaMemcpyHostToDevice);
    cudaMemcpy(..., ...);
    cudaMemcpyHostToDevice);

    saxpy<<< (N+255) / 256, 256>>>...;

    cudaMemcpy(y, d_y, N*sizeof(float),
    cudaMemcpyDeviceToHost);
```

FPGA



```
std::string binaryFile = argv[1];
size_t vector_size_bytes = sizeof(int) * DATA_SIZE;
cl_int err;
cl::Context context;
cl::Kernel kml_vector_add;
cl::CommandQueue q;
// Allocate Memory in Host Memory
// When creating a buffer with user pointer (CL_MEM_USE_HOST_PTR), under the hood user ptr
// is used if it is properly aligned. when not aligned, runtime had no choice but to create
// its own host side buffer. So it is recommended to use this allocator if user wish to
// create buffer using CL_MEM_USE_HOST_PTR to align user buffer to page boundary. It will
// ensure that user buffer is used when user create Buffer/Mem object with CL_MEM_USE_HOST_PTR
std::vector<int>, aligned_allocator<int>> source_in1(DATA_SIZE);
std::vector<int>, aligned_allocator<int>> source_in2(DATA_SIZE);
std::vector<int>, aligned_allocator<int>> source_hw_results(DATA_SIZE);
std::vector<int>, aligned_allocator<int>> source_sw_results(DATA_SIZE);

// Create the test data
std::generate(source_in1.begin(), source_in1.end(), std::rand);
std::generate(source_in2.begin(), source_in2.end(), std::rand);
for (int i = 0; i < DATA_SIZE; i++) {
    source_sw_results[i] = source_in1[i] + source_in2[i];
    source_hw_results[i] = 0;
}

// OPENCL HOST CODE AREA START
// get_xil_devices() is a utility API which will find the xilinx
// platforms and will return list of devices connected to xilinx platform
auto devices = xcl::get_xil_devices();
// read_binary_file() is a utility API which will load the binary file
// and will return the pointer to file buffer.
auto fileBuf = xcl::read_binary_file(binaryFile);
cl::Program bins{{fileBuf.data(), fileBuf.size()}};
int valid_device = 0;
for (unsigned int i = 0; i < devices.size(); i++) {
    auto device = devices[i];
    // Creating Context and Command Queue for selected device
    OCL_CHECK(err, context = cl::Context({device}, NULL, NULL, &err));
    OCL_CHECK(err,
        q = cl::CommandQueue(
            context, {device}, CL_QUEUE_PROFILING_ENABLE, &err));
    std::cout << "Trying to program device[" << i
        << "]: " << device.getInfo(CL_DEVICE_NAME) << std::endl;
    OCL_CHECK(err,
        cl::Program program(context, {device}, bins, NULL, &err));
    if (err != CL_SUCCESS) {
        std::cout << "Failed to program device[" << i
            << "] with xclbin file!\n";
    } else {
        std::cout << "Device[" << i << "] program successful!\n";
        OCL_CHECK(err, kml_vector_add = cl::Kernel(program, "vadd", &err));
        valid_device++;
    }
}
```

FPGA device

Bitstreams

Memory Allocation

Transfer

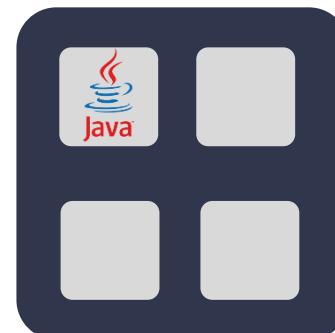
Challenges on FPGAs – Deployment



> How can I **deploy** my FPGA accelerator easy?



> Without having to specify on host code about bitstreams, FPGA card, memory management, memory transfers



Challenges on FPGAs – Scaling



> How can I instantly **scale-out** my applications to multiple FPGAs?



> Manually distribution on workload on different FPGAs.

- >> Error-prone
- >> Complex
- >> Not scalable



Challenges on FPGAs – Resource management



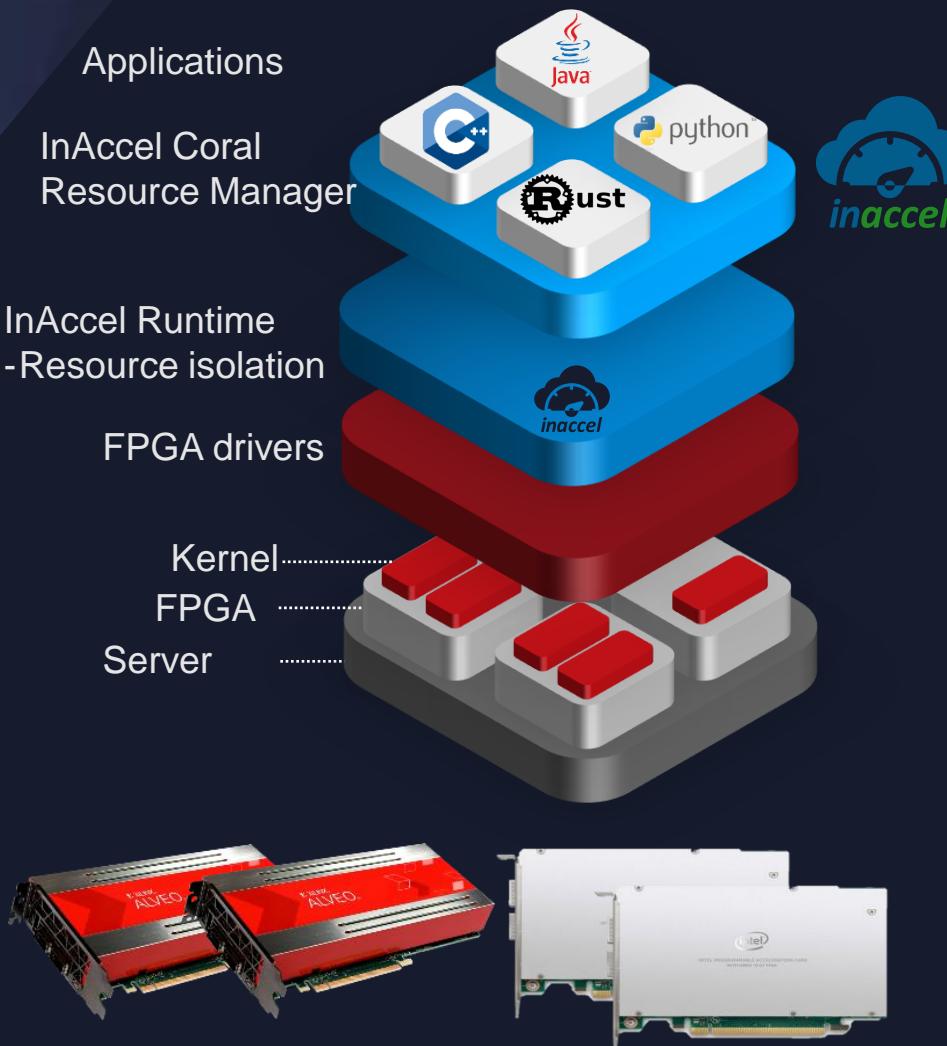
> How can **multiple** users or applications **share** my FPGA cluster?



- > Currently only a single application can control the FPGA configuration
- > Hard to share FPGA resource among users/threads/processes



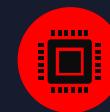
Scalable Orchestrator for FPGA clusters



Automated Deployment, Scaling and Management of FPGA clusters



Seamless invoking from C/C++, Python, Java and Scala. No need for OpenCL



Automatic configuration and management of the FPGA **bitstreams** and **memory**



Seamless **resource management** of the FPGA cluster from multiple threads/processes/applications/users



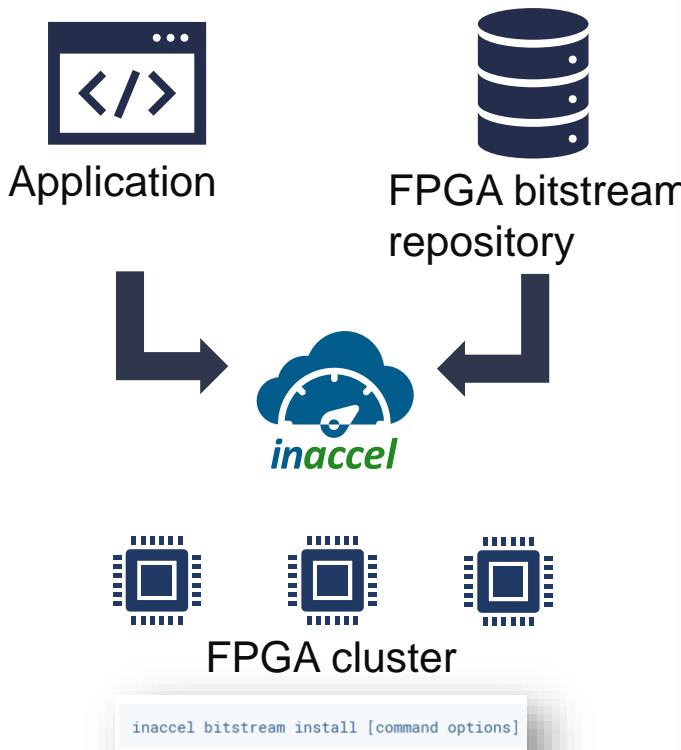
Fully **scalable**: Scale-up (multiple FPGAs per node) and Scale-out (multiple FPGA-based servers over Spark)

Bitstream repository



- > **FPGA Resource Manager is integrated with a bitstream repository that is used to store FPGA bitstreams**

<https://store.inaccel.com>



A screenshot of the inaccel Artifact Repository Browser. The interface shows a tree view of bitstreams stored in a repository. The tree structure includes categories like bitstreams, intel, xilinx, u200, u250/xdma_201830.2, com, and u280. Under the xilinx category, there are sub-folders for aws-vu9p-f1/dynamic_5.0/com, aws-vu9p-f1-04261818/dynamic_5.0/com, and xdma_201820.1/com, among others. The xilinx/vitis folder under u200 contains sub-folders for inaccel/math/vector/0.1/2addition_2subtraction, dataCompression/lz4/1.0, quantitativeFinance, security/aes256/1.0, and vision/1.0/1stereoBM. The com/inaccel/xilinx/vitis folder under u250/xdma_201830.2 contains sub-folders for quantitativeFinance/monteCarlo/1.0/1Calibration_1Pre and vision. The xilinx/com/researchlabs folder under u280 contains sub-folders for xdma_201910.1/com/inaccel/math/vector/0.1/2addition_2subtraction and xdma_201920.3/com/inaccel/xilinx/vitis/vision. On the right side of the interface, there is a detailed view of the "xilinx/vitis/vision" entry, showing its General properties. The properties include Name: vision, Repository Path: bitstreams/xilinx/u280/xdma_201920.3/com/xilinx/vitis/vision/, Deployed By: xilinx, Artifact Count / Size: Show, and Created: 09-03-20 10:37:17 +00:00 (77d 1h 31m 45s ago).

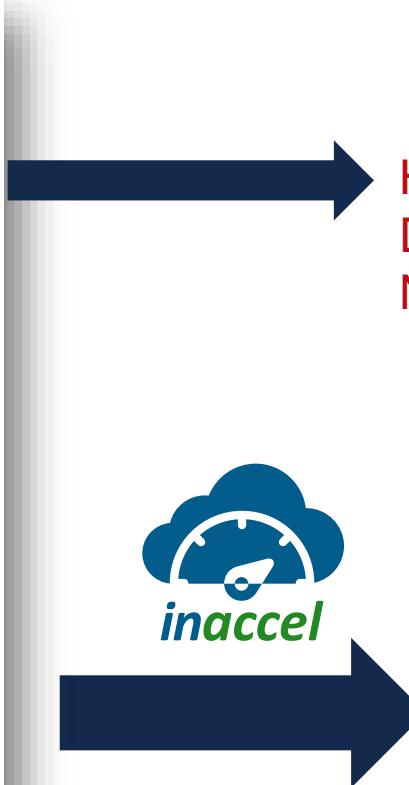
Simple deployment – InAccel API



```
std::string binaryFile = argv[1];
size_t vector_size_bytes = sizeof(int) * DATA_SIZE;
cl_int err;
cl::Context context;
cl::Kernel krn1_vector_add;
cl::CommandQueue q;
// Allocate Memory in Host Memory
// When creating a buffer with user pointer (CL_MEM_USE_HOST_PTR), under the hood user ptr
// is used if it is properly aligned. when not aligned, runtime had no choice but to create
// its own host side buffer. So it is recommended to use this allocator if user wish to
// create buffer using CL_MEM_USE_HOST_PTR to align user buffer to page boundary. It will
// ensure that user buffer is used when user create Buffer/Mem object with CL_MEM_USE_HOST_PTR
std::vector<int, aligned_allocator<int>> source_in1(DATA_SIZE);
std::vector<int, aligned_allocator<int>> source_in2(DATA_SIZE);
std::vector<int, aligned_allocator<int>> source_hw_results(DATA_SIZE);
std::vector<int, aligned_allocator<int>> source_sw_results(DATA_SIZE);

// Create the test data
std::generate(source_in1.begin(), source_in1.end(), std::rand);
std::generate(source_in2.begin(), source_in2.end(), std::rand);
for (int i = 0; i < DATA_SIZE; i++) {
    source_sw_results[i] = source_in1[i] + source_in2[i];
    source_hw_results[i] = 0;
}

// OPENCL HOST CODE AREA START
// get_xil_devices() is a utility API which will find the xilinx
// platforms and will return list of devices connected to Xilinx platform
auto devices = xcl::get_xil_devices();
// read_binary_file() is a utility API which will load the binaryFile
// and will return the pointer to file buffer.
auto fileBuf = xcl::read_binary_file(binaryFile);
cl::Program::binaries bins{{fileBuf.data(), fileBuf.size()}};
int valid_device = 0;
for (unsigned int i = 0; i < devices.size(); i++) {
    auto device = devices[i];
    // Creating Context and Command Queue for selected Device
    OCL_CHECK(err, context = cl::Context({device}, NULL, NULL, NULL, &err));
    OCL_CHECK(err,
        q = cl::CommandQueue(
            context, {device}, CL_QUEUE_PROFILING_ENABLE, &err));
    std::cout << "Trying to program device[" << i
        << "]": << device.getInfo<CL_DEVICE_NAME>() << std::endl;
    OCL_CHECK(err,
        cl::Program program(context, {device}, bins, NULL, &err));
    if (err != CL_SUCCESS) {
        std::cout << "Failed to program device[" << i
            << "] with xclbin file!\n";
    } else {
        std::cout << "Device[" << i << "]: program successful!\n";
        OCL_CHECK(err, krn1_vector_add = cl::Kernel(program, "vadd", &err));
        valid_device++;
    }
}
```



Host-side buffers only
Decouple applications from bitstreams
No platform-dependent device configurations

```
inaccel::request vadd("vector.addition");
vadd.arg(a).arg(b).arg(c).arg(size);
inaccel::submit(vadd).get();
```

- Simple programming using InAccel Coral API
- Asynchronous accelerator invocation
- No OpenCL directives
- Unified API in C/C++, Java, Python and Rust

<https://setup.inaccel.com/coral-api/#using-the-api>

C++ invoking



CPU only

```
unsigned int nbytes    = (width*height);

// Input and output buffers (Y,U,V)
YUVImage srcImage;
YUVImage dstImage;
srcImage.yChannel = (unsigned char *)malloc(nbytes);
srcImage.uChannel = (unsigned char *)malloc(nbytes);
srcImage.vChannel = (unsigned char *)malloc(nbytes);
dstImage.yChannel = (unsigned char *)malloc(nbytes);
dstImage.uChannel = (unsigned char *)malloc(nbytes);
dstImage.vChannel = (unsigned char *)malloc(nbytes);

// Create output buffers for reference results
unsigned char *y_ref = (unsigned char *)malloc(nbytes);
unsigned char *u_ref = (unsigned char *)malloc(nbytes);
unsigned char *v_ref = (unsigned char *)malloc(nbytes);

unsigned numRunsSW = comparePerf?numRuns:1;

#pragma omp parallel for num_threads(3)
for(unsigned int n=0; n<numRunsSW; n++)
{
    // Compute reference results
    Filter2D(filterCoeffs[filterType], factor, bias, width, height, srcImage.yChannel, y_ref);
    Filter2D(filterCoeffs[filterType], factor, bias, width, height, srcImage.uChannel, u_ref);
    Filter2D(filterCoeffs[filterType], factor, bias, width, height, srcImage.vChannel, v_ref);
}
```

Reference ConvFilter

InAccel

```
unsigned int nbytes    = (width*height);

// Input and output buffers (Y,U,V)
YUVImage srcImage;
YUVImage dstImage;
srcImage.yChannel = (unsigned char *)inaccel_alloc(nbytes);
srcImage.uChannel = (unsigned char *)inaccel_alloc(nbytes);
srcImage.vChannel = (unsigned char *)inaccel_alloc(nbytes);
dstImage.yChannel = (unsigned char *)inaccel_alloc(nbytes);
dstImage.uChannel = (unsigned char *)inaccel_alloc(nbytes);
dstImage.vChannel = (unsigned char *)inaccel_alloc(nbytes);

// Create output buffers for reference results
unsigned char *y_ref = (unsigned char *)inaccel_alloc(nbytes);
unsigned char *u_ref = (unsigned char *)inaccel_alloc(nbytes);
unsigned char *v_ref = (unsigned char *)inaccel_alloc(nbytes);

unsigned numRunsSW = comparePerf?numRuns:1;

#pragma omp parallel for num_threads(3)
for(unsigned int n=0; n<numRunsSW; n++)
{
    // Compute reference results
    Filter2D(filterCoeffs[filterType], factor, bias, width, height, srcImage.yChannel, y_ref);
    Filter2D(filterCoeffs[filterType], factor, bias, width, height, srcImage.uChannel, u_ref);
    Filter2D(filterCoeffs[filterType], factor, bias, width, height, srcImage.vChannel, v_ref);
}
```

InAccel ConvFilter

Ready to use accelerators



Instant evaluation of accelerations

No need for synthesis, P&R

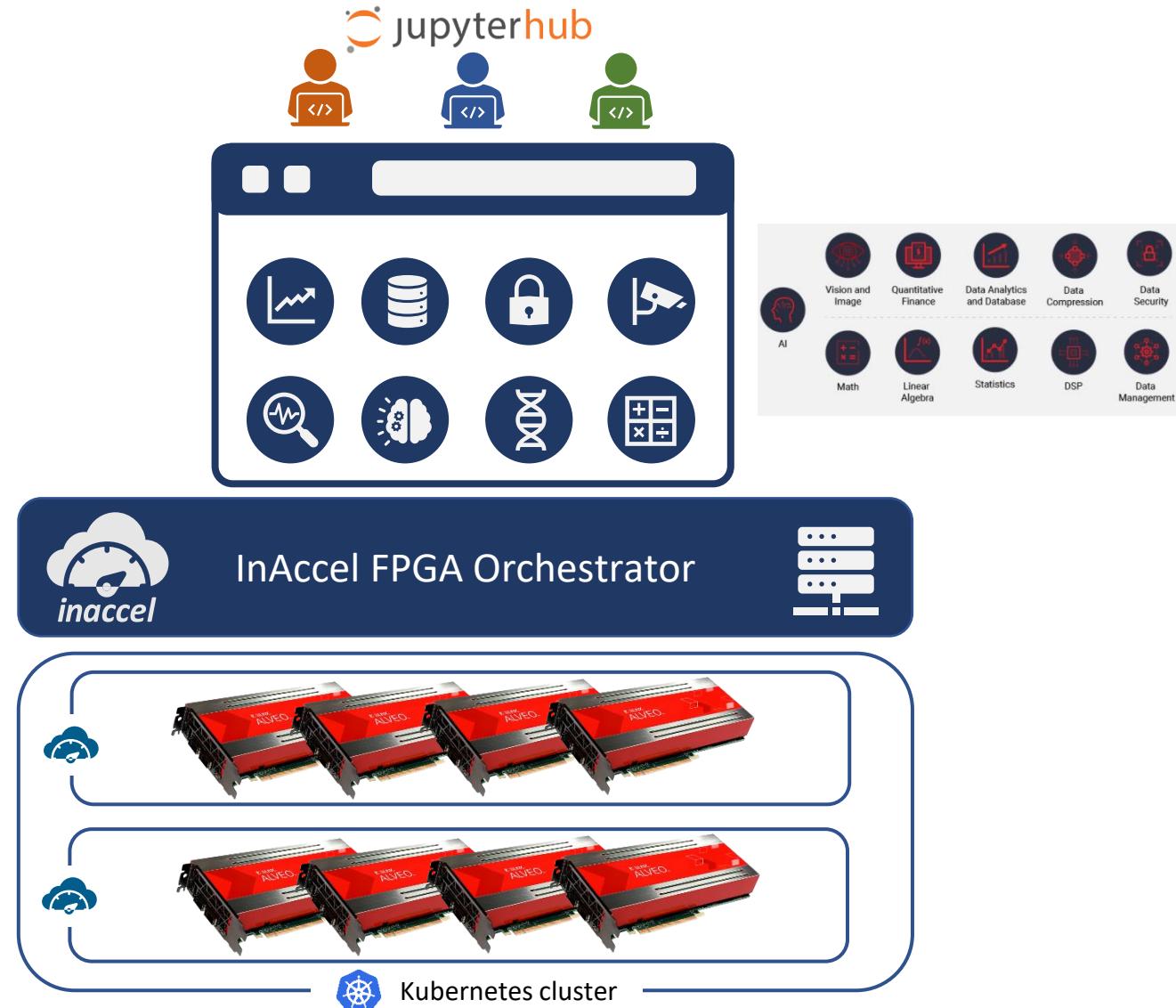
- No need for bitstreams
 - No need for OpenCL
 - No need for configuration/boards
 - No need for an account



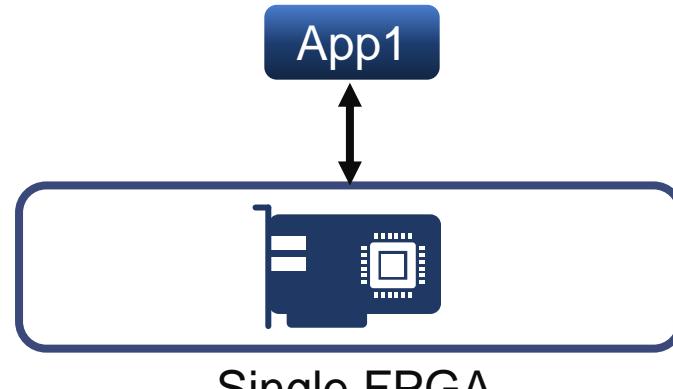
Multi-tenant Vitis deployment



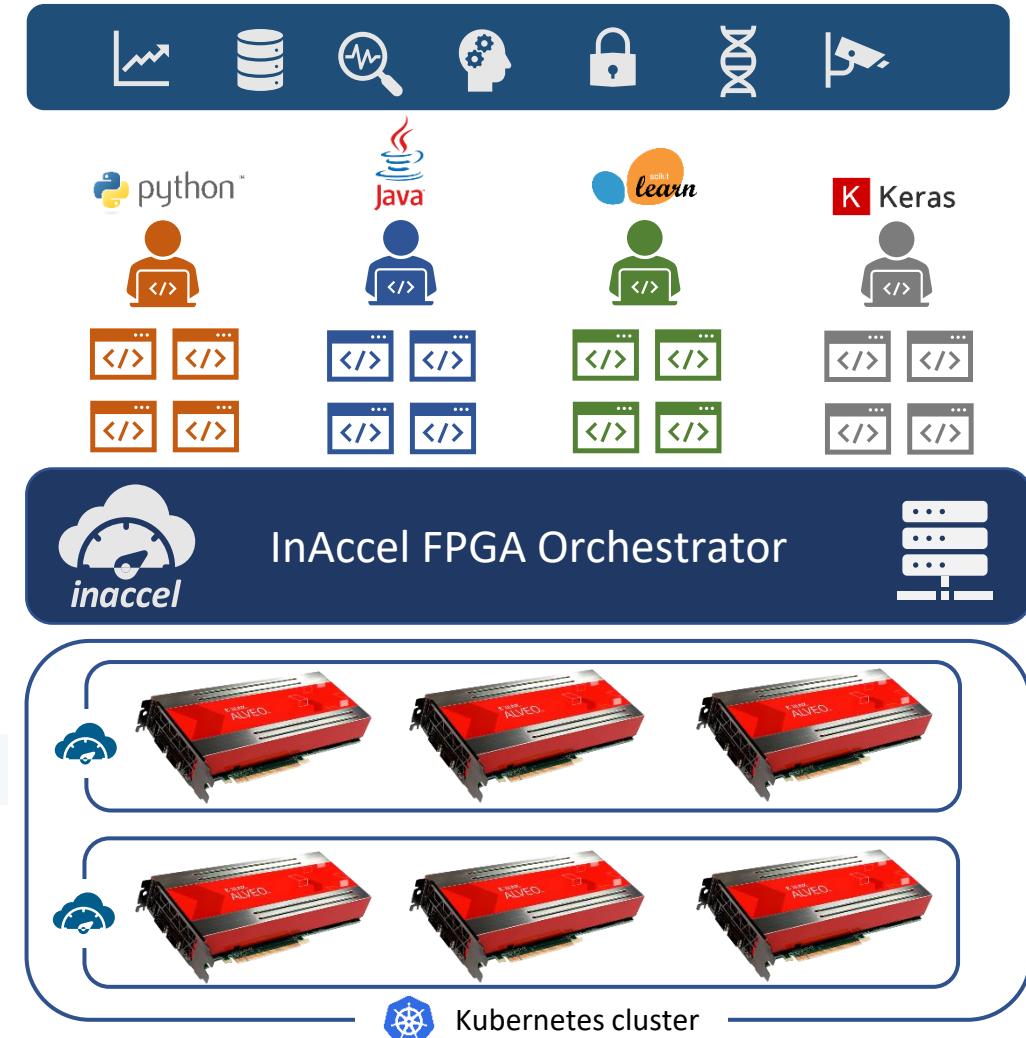
- > Run Vitis from browser
- > Fully compatible with any Vitis library
- > Multi-tenant, multiple applications
- > Scalable deployment



From single node to scalable deployment

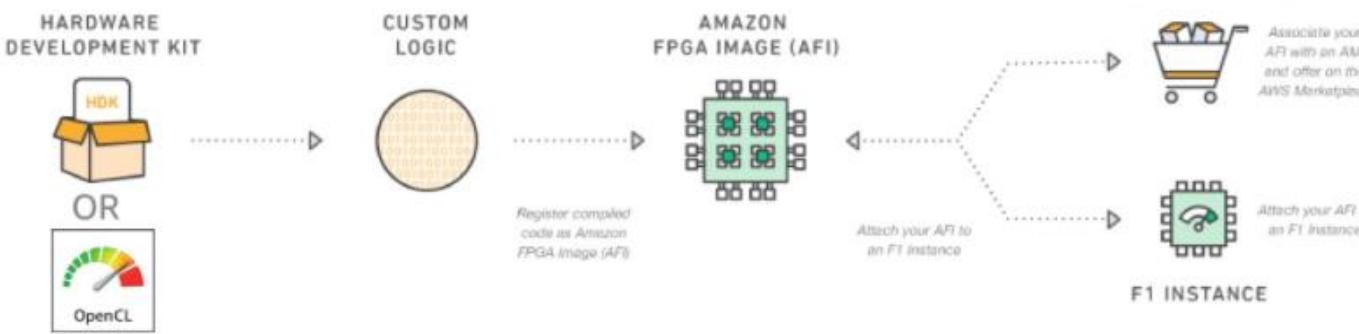


```
curl -sS https://setup.inaccel.com/repo | sh -s install
```



Deploy FPGAs on cloud

- > Several steps
- > Prior knowledge on FPGAs
 - >> Bitstream
 - >> Memory management
 - >> Communication
 - >> Challenges: Bitstream version, Firmware, SDK



How To Create an Amazon FPGA Image (AFI) From One of The CL Examples: Step-by-Step Guide

Fast path to running CL Examples on FPGA Instance

For developers that want to skip the development flow and start running the examples on the FPGA instance. You can skip steps 1 through 3 if you are not interested in the development process. Step 4 through 6 will show you how to use one of the predesigned AFI examples. By using the public AFIs, developers can skip the build flow steps and jump to step 4. Public AFIs are available for each example and can be found in the example/README.

Step 1. Pick one of the examples and start in the example directory

It is recommended that you complete this step-by-step guide using HDK hello world example. Next use this same guide to develop using the cl_dram_dma. When you're ready, copy one of the examples provided and modify the design files, scripts and constraints directory.

```
$ cd $HDK_DIR/cl/examples/cl_hello_world # you can change cl_hello_world to cl_dram_dma, cl_uram_example or cl_hello_world_vhd1  
$ export CL_DIR=$pwd
```

Setting up the CL DIR environment variable is crucial as the build scripts rely on that value. Each example follows the recommended directory structure to match the expected structure for HDK simulation and build scripts.

Step 2. Build the CL

This checklist should be consulted before you start the build process.

NOTE This step requires you to have Xilinx Vivado Tools and Licenses installed

```
$ vivado -mode batch # Verify Vivado is installed.
```

Executing the `aws_build_dcp_from_cl.sh` script will perform the entire implementation process converting the CL design into a completed Design Checkpoint that meets timing and placement constraints of the target FPGA. The output is a tarball file comprising the DCP file, and other log/manifest files, formatted as `YY_MM_DD-HhMm-Developer_CL.tar`. This file would be submitted to AWS to create an AFI. By default the build script will use Clock Group A Recipe A0 which uses a main clock of 125 MHz.

```
$ cd $CL_DIR/build/scripts  
$ ./aws_build_dcp_from_cl.sh
```

In order to use a 250 MHz main clock the developer can specify the A1 Clock Group A Recipe as in the following example:

```
$ cd $CL_DIR/build/scripts  
$ ./aws_build_dcp_from_cl.sh -clock_recipe_a A1
```

Other clock recipes can be specified as well. More details on the [Clock Group Recipes Table](#) and how to specify different recipes can be found in the following README.

NOTE: The DCP generation can take up to several hours to complete, hence the `aws_build_dcp_from_cl.sh` will run the main build process (`vivado`) in within a `nohup` context. This will allow the build to continue running even if the SSH session is terminated half way through the run.

To be notified via e-mail when the build completes:

1. Set up notification via SNS:

```
$ pip install --user --upgrade boto3 # boto3 package is required by the notify_via sns script  
$ export EMAIL=your_email@example.com  
$ $AWS_FPGA_REPO_DIR/shared/bin/scripts/notify_via_sns.py
```

2. Check your e-mail address and confirm subscription

3. When calling `aws_build_dcp_from_cl.sh`, add on the `-notify` switch
4. Once your build is complete, an e-mail will be sent to you stating "Your build is done."
5. For each example the known warnings are documented in `warnings.txt` file located in the `$CL_DIR/build/scripts` directory `cl_hello_world` `warnings cl_dram_dma` `warnings cl_uram_example` `warnings`

Step 3. Submit the Design Checkpoint to AWS to Create the AFI

To submit the DCP, create an S3 bucket for submitting the design and upload the tarball file into that bucket. You need to prepare the following information:

1. Name of the logic design (Optional).
2. Generic description of the logic design (Optional).
3. Location of the tarball file object in S3.
4. Location of an S3 directory where AWS would write back logs of the AFI creation.
5. AWS region where the AFI will be created. Use `copy-fpga-image` API to copy an AFI to a different region.

To upload your tarball file to S3, you can use any of the tools supported by S3.

Deployment of FPGA on Kubernetes – before



1. Install vendor-specific **FPGA drivers** and deployment shells on every node
2. Deploy the Intel/Xilinx **FPGA Device Plugin**
3. Develop OpenCL-based applications, that contain **platform-dependent** code
4. Build "fat" container images which **include large bitstream** files
5. Run **Kubernetes tasks** which are hard to maintain/upgrade

6. ... and still you need to **manually** perform **workload balancing** to distribute acceleration tasks along the requested FPGA resources

Kubernetes deployment - before

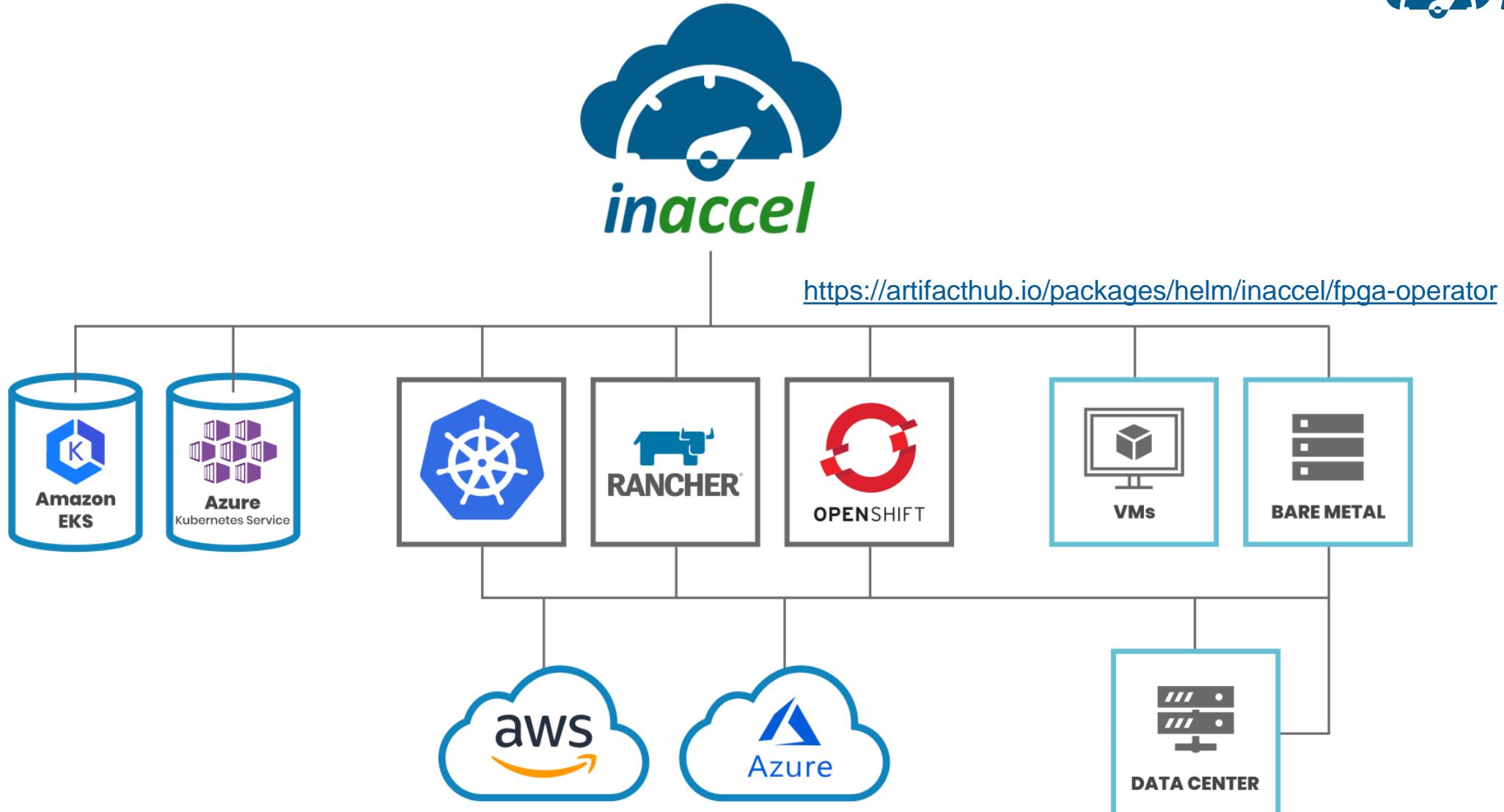


The screenshot shows a series of web pages from <https://developer.xilinx.com> related to "Using Alveo in a Kubernetes Environment".

- Top Left Tab:** "Using Alveo in a Kubernetes Environment". This page provides an overview of using Alveo in a Kubernetes environment, mentioning the setup of two servers (one with two U200 cards and one with one U200 card) and the installation of the Alveo Device Plugin.
- Top Right Tab:** A diagram titled "Using Alveo in a Kubernetes Environment" showing a network topology with an Internet connection, a router, a switch, and two server nodes. One node is labeled "Kubernetes" and the other "Alveo".
- Middle Left Tab:** "Using Alveo in a Kubernetes Environment". This page contains detailed steps for setting up the environment, including installing CentOS 7.6, XRT 2019.2, and the Alveo Device Plugin, and configuring a Kubernetes cluster with two nodes.
- Middle Right Tab:** A code snippet showing a Kubernetes pod definition for an Alveo resource named "mypod". It includes a command to run a shell script that prints "Hello World".
- Bottom Left Tab:** "Using Alveo in a Kubernetes Environment". This page shows log output from a pod named "kubemaster" and a "mypod" pod, detailing their creation and running status.
- Bottom Right Tab:** A code snippet showing a Kubernetes batch job named "my_zlib" with five parallel instances, each requiring two U200 cards. It includes a command to run a shell script named "verify_zlib.sh".

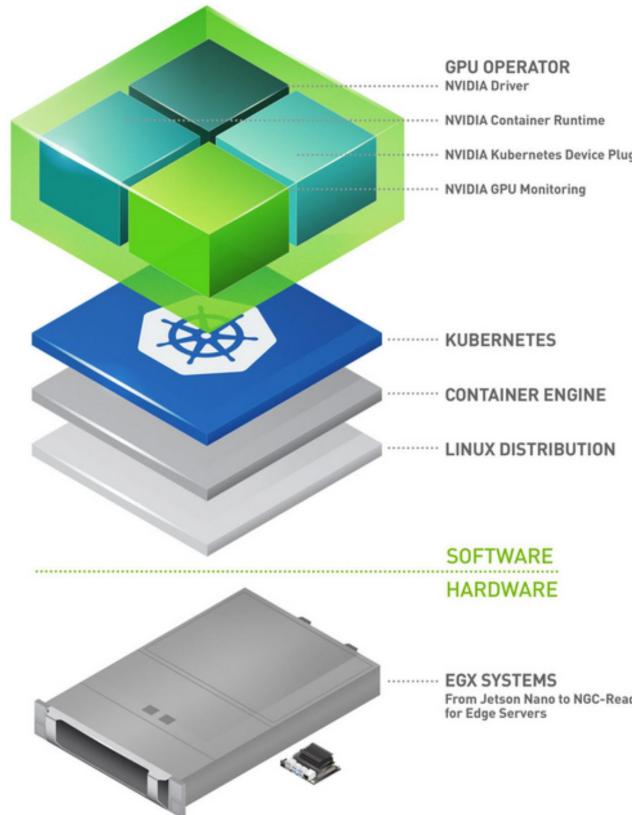
<https://developer.xilinx.com/en/articles/using-alveo-in-a-kubernetes-environment.html>

Unique InAccel FPGA Operator



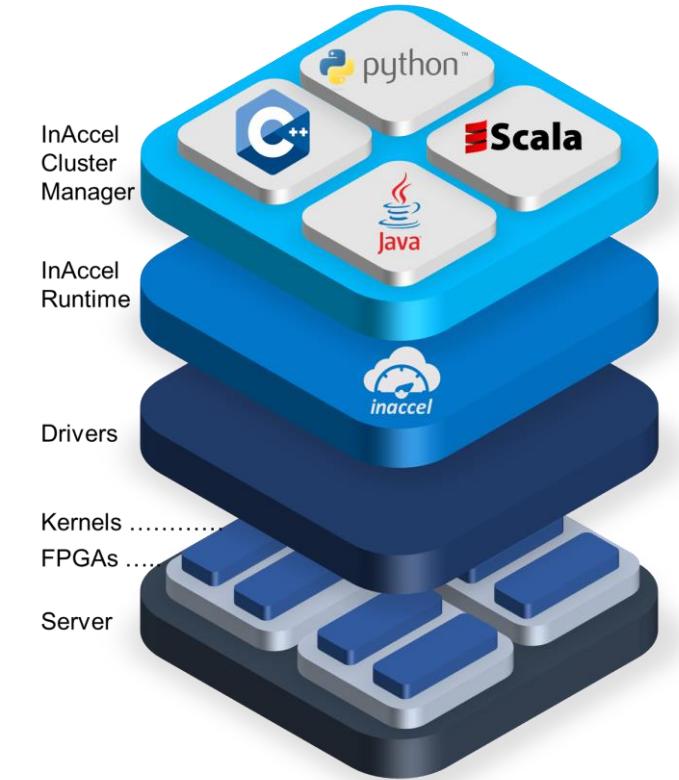
The InAccel FPGA operator manages FPGA resources in a Kubernetes cluster and automates tasks related to bootstrapping FPGA nodes.

Simplifying FPGA deployment in Kubernetes



NVIDIA - Full stack for GPUs

<https://developer.nvidia.com/blog/nvidia-gpu-operator-simplifying-gpu-management-in-kubernetes/>



Full stack for FPGAs (vendor agnostic)

FPGA deployment is a single step



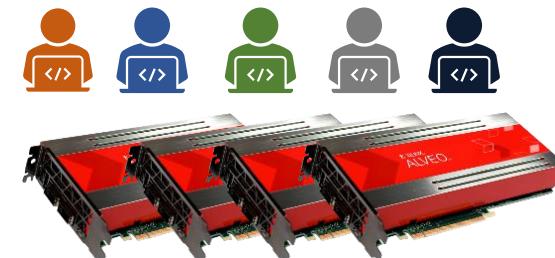
- 1. Spawn a K8s cluster**
- 2. Deploy InAccel FPGA operator**

```
1. helm repo add inaccel https://setup.inaccel.com/helm  
2. helm install my-fpga-operator inaccel/fpga-operator
```

FPGA drivers/runtime, InAccel Coral Resource manager + Monitor

- 3. Run your application targeting FPGA resources**

- > **Multiple users**
- > **Auto-scaling**
- > **Easy resource management**



Cloud deployment on Kubernetes



> FPGA deployment on EKS cluster using Rancher UI and InAccel FPGA Operator

A screenshot of the Rancher UI interface. The left sidebar shows various cluster resources like Namespaces, Nodes, Workload, and DaemonSets. The main panel displays the 'DaemonSet: fpga-operator' details. It shows the namespace as 'kube-system' and an age of '14 mins'. The image used is 'inaccel/coral:2.0'. Below this, a chart titled 'Pods by State' shows 2 Active, 0 Transitioning, 0 Warning, and 0 Error pods. A table then lists the individual pods: 'fpga-operator-6xrmf' and 'fpga-operator-sdtjj', both running on node 'ip-192-168-135-176.ec2.internal' and using the 'inaccel/coral:2.0' image.

State	Name	Node	Image
Running	fpga-operator-6xrmf	ip-192-168-135-176.ec2.internal	inaccel/coral:2.0
Running	fpga-operator-sdtjj	ip-192-168-87-154.ec2.internal	inaccel/coral:2.0

<https://www.youtube.com/watch?v=lqhIkX7oLBs>

FPGA on Kubernetes using InAccel



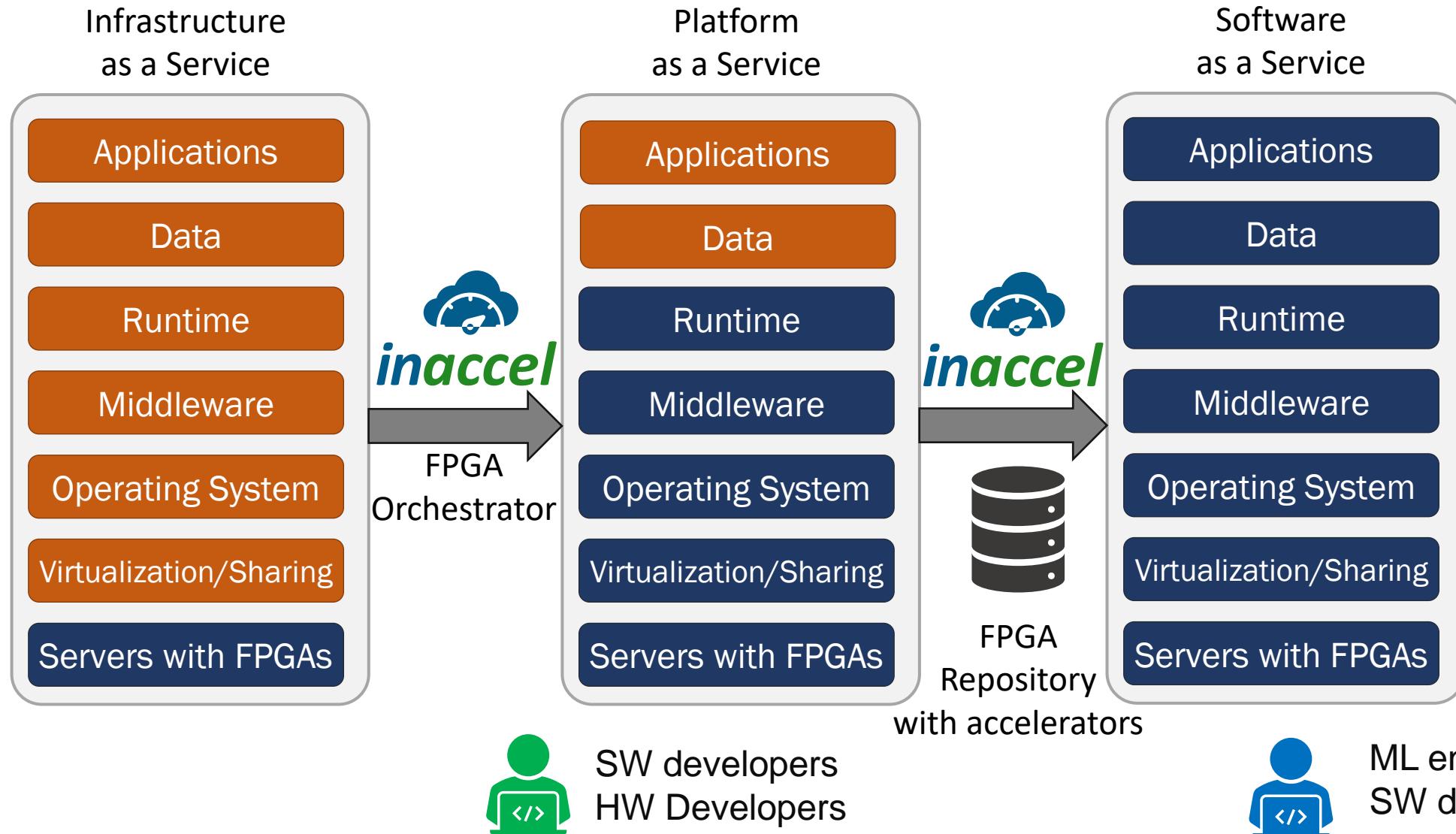
Kubernetes on FPGAs

This snippet explains how to run FPGAs in Kubernetes clusters like a pro

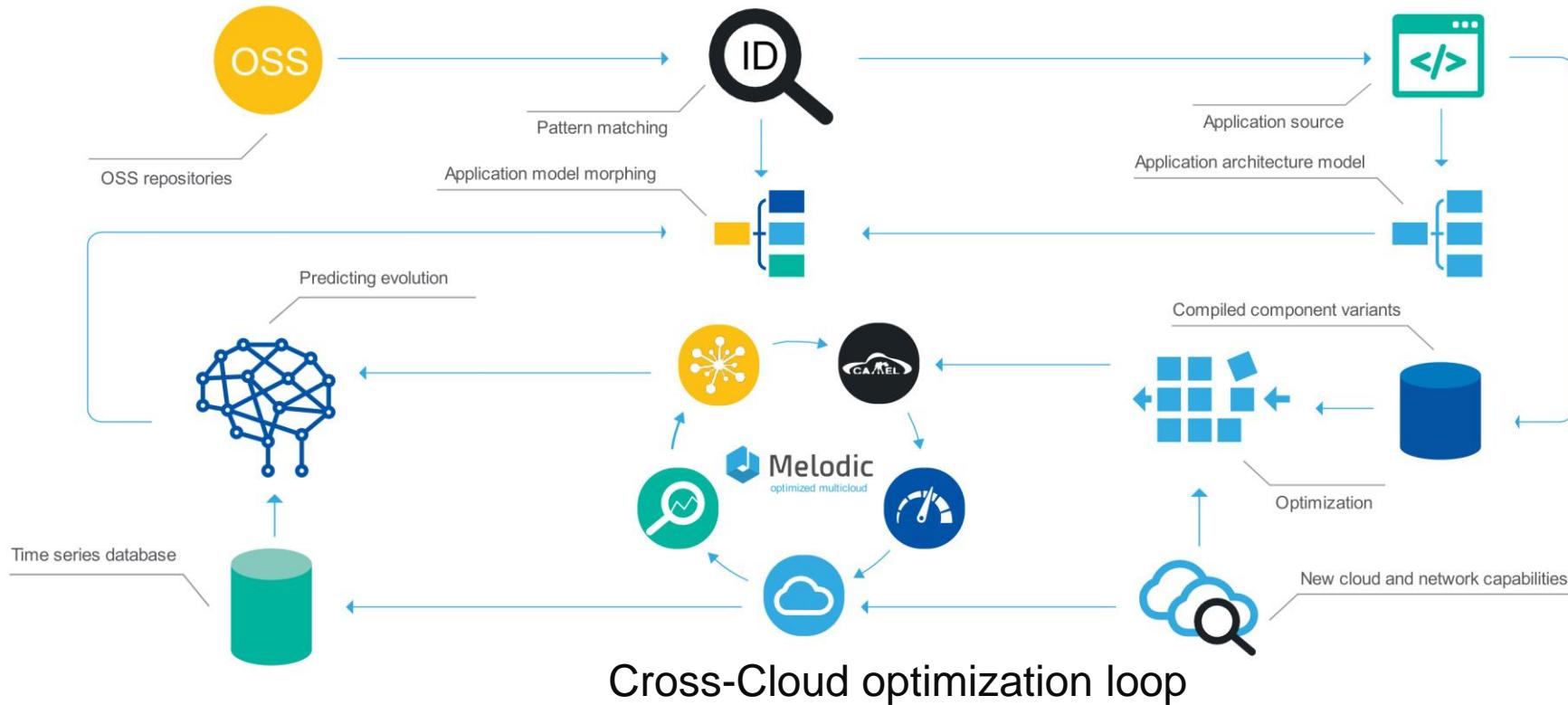
```
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  containers:
  - name: example
    image: inaccel/jupyter:lab
    ports:
    - containerPort: 8888
```

<https://www.youtube.com/watch?v=E94YTh4mm1g>

PaaS and SaaS for FPGA clusters



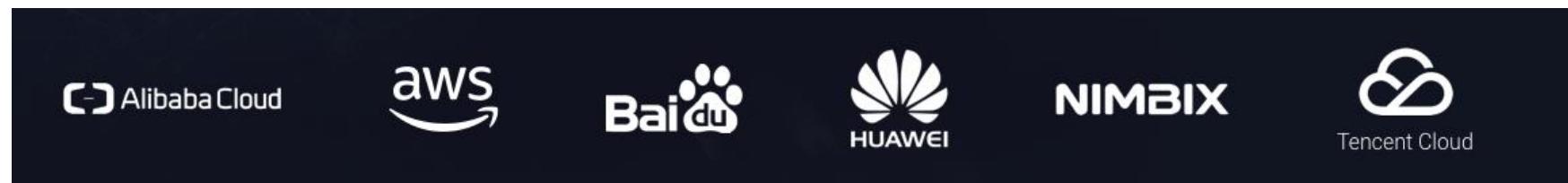
H2020: Multi-cloud deployment



How hardware accelerators can be deployed on multi-cloud environment?



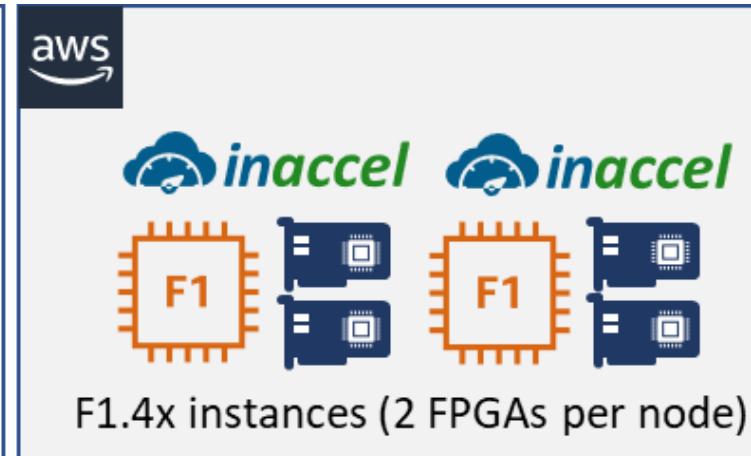
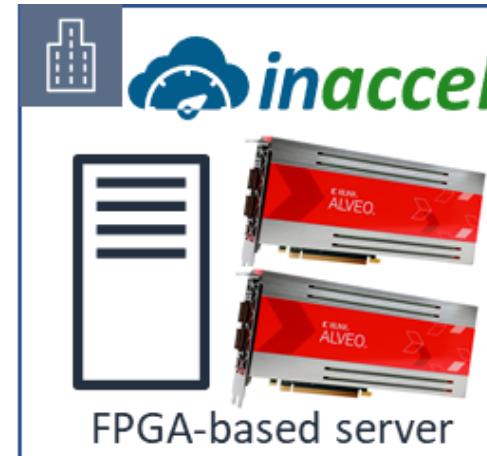
<https://www.morphemic.cloud/>



More Challenges



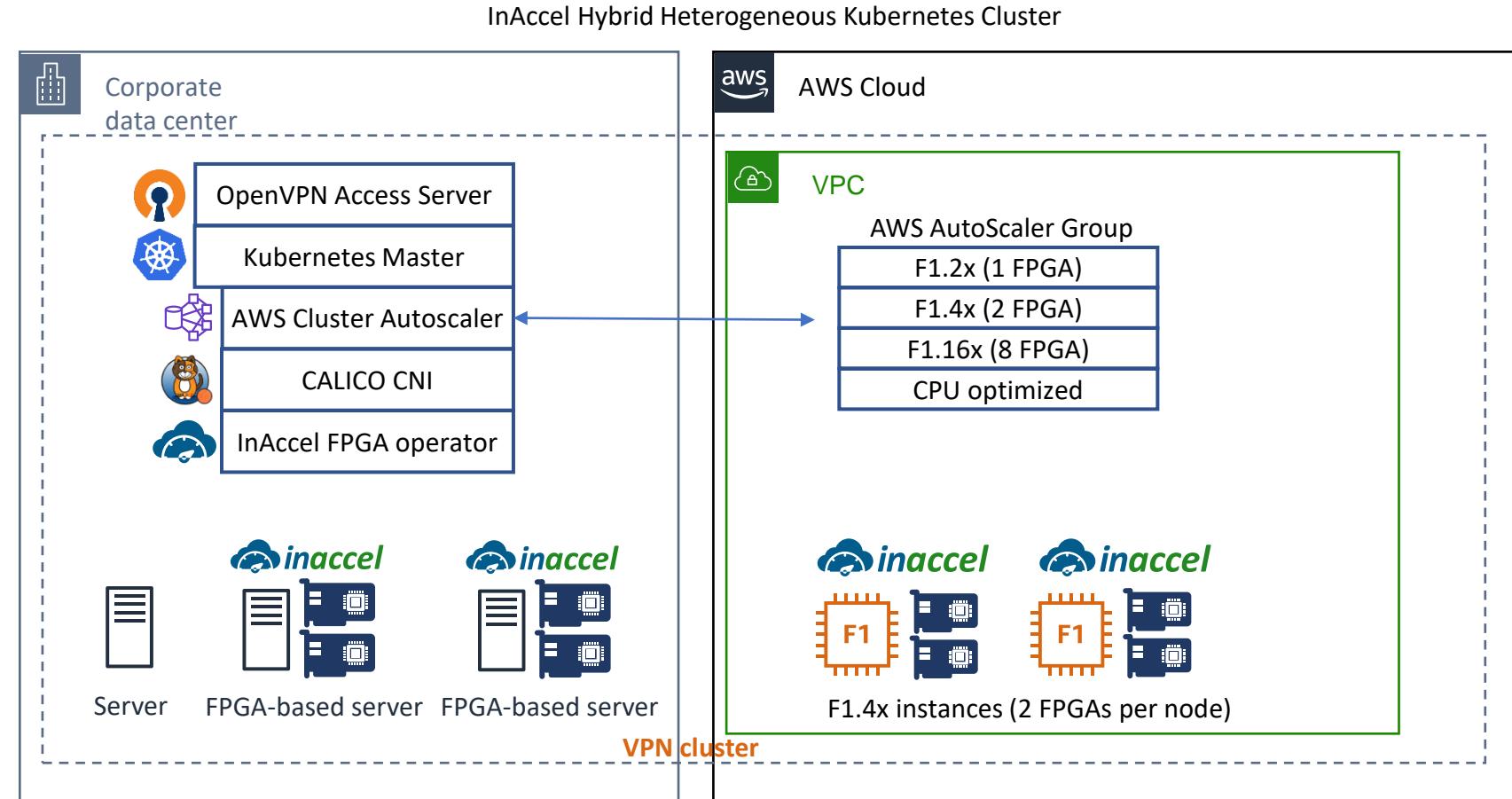
- > How can **scale-out** my application on-prem and on cloud?



Auto-scalable deployment



- > Starting on prem
- > Moving to the cloud
 - >> Automatically
 - >> Instantly



Auto-scalable FPGA deployment



Setup the Master node

1. Initialize the Kubernetes control-plane. Use the VPN IP, that the OpenVPN Access Server has assigned to that node (e.g 172.27.224.1), as the IP address the API Server will advertise it's listening on.

```
sudo kubeadm init \  
  --apiserver-advertise-address=172.27.224.1 \  
  --kubernetes-version stable-1.18
```

To make `helm` and `kubectl` work for your non-root user, use the commands from the `kubeadm init` output.

2. Deploy **Calico** network policy engine for Kubernetes.

```
kubectl apply -f https://docs.projectcalico.org/v3.14/manifests/calico.yaml
```

3. Deploy **Cluster Autoscaler** for AWS.

```
helm repo add stable https://kubernetes-charts.storage.googleapis.com  
helm install cluster-autoscaler stable/cluster-autoscaler \  
  --set autoDiscovery.clusterName=InAccel \  
  --set awsAccessKeyId=<your-aws-access-key-id> \  
  --set awsRegion=us-east-1 \  
  --set awsSecretAccessKey=<your-aws-secret-access-key> \  
  --set cloudProvider=aws
```

4. Deploy InAccel FPGA Operator.

```
helm repo add inaccel https://setup.inaccel.com/helm  
helm install inaccel inaccel/fpga-operator \  
  --set license=<your-license> \  
  --set nodeSelector.inaccel/fpga(enabled
```

Setup the local Worker nodes

<https://docs.inaccel.com/labs/auto-scaling-aws/>

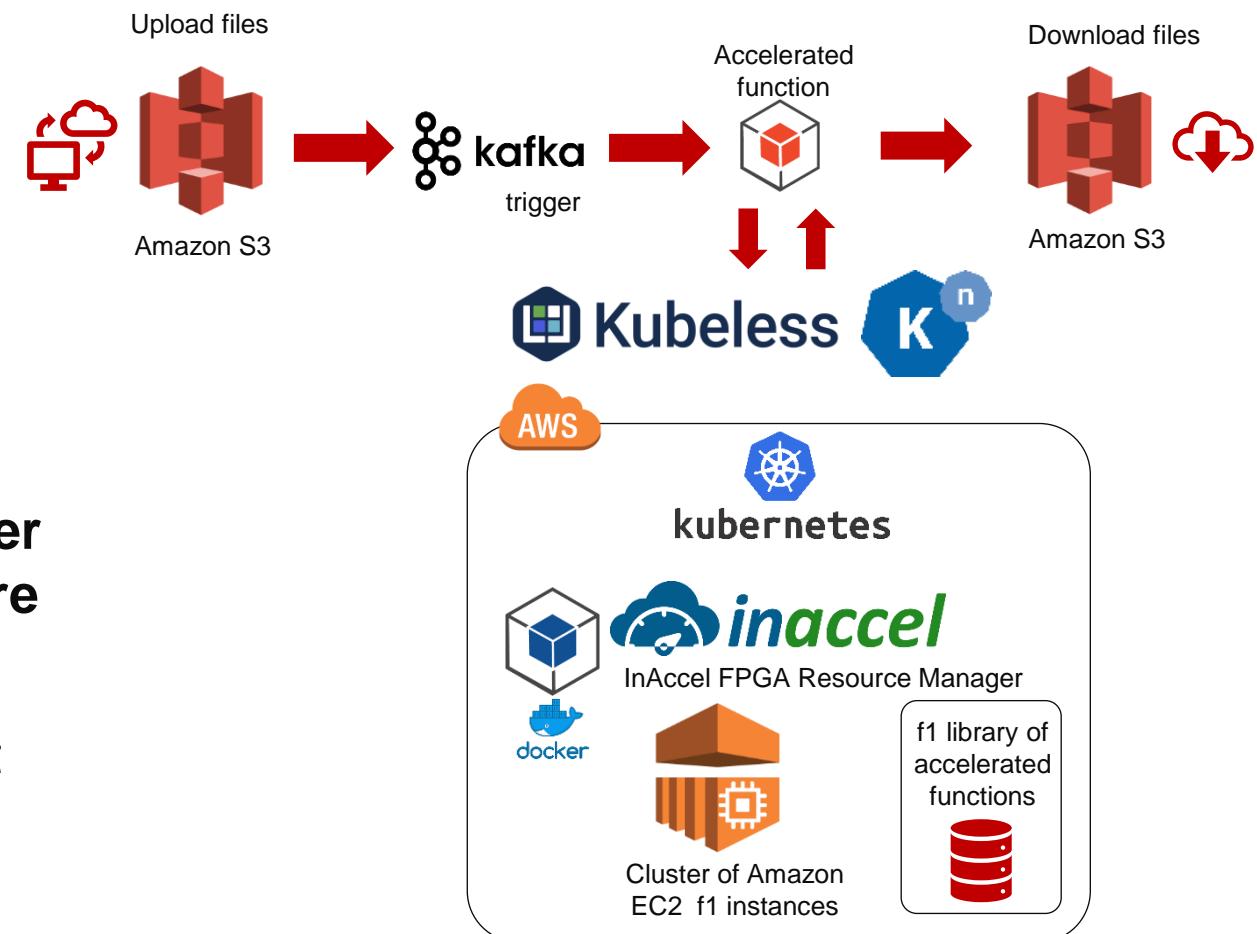


<https://www.youtube.com/watch?v=CVVvYvXY4w5w>

Serverless deployment

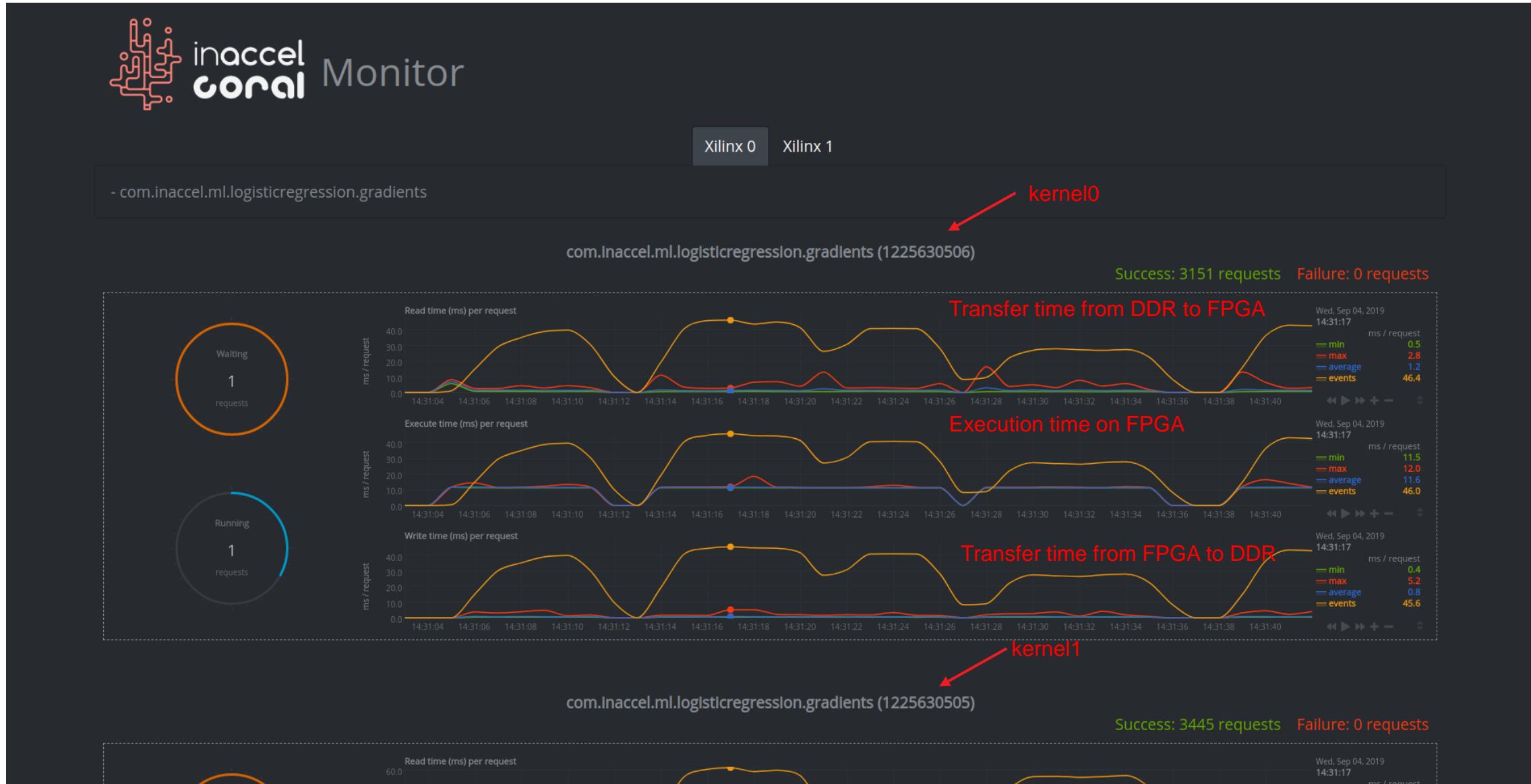


- > Integrated framework for serverless deployment
- > Compatible with Kubeless, Knative
- > Users only have to **upload the images** on the S3 bucket and then InAccel's FPGA Manager **automatically deploy the cluster of FPGAs**, process the data and then **store back the results** on the S3 bucket.
- > Users do not have to know anything about the FPGA execution.



<https://medium.com/@inaccel/fpgas-goes-serverless-on-kubernetes-55c1d39c5e30>

Insight into the FPGA utilization



Keras Deployment on Alveo cards



- > Easy deployment of Keras applications



```
pip install inaccel-keras
```

```
import time

from inaccel.keras.applications.resnet50 import ResNet50
from inaccel.keras.preprocessing.image import ImageDataGenerator

model = ResNet50(weights='imagenet')

data = ImageDataGenerator(dtype='int8')
images = data.flow_from_directory('imagenet/', target_size=(224, 224), class_mode=None, batch_size=64)

begin = time.monotonic()
preds = model.predict(images, workers=16)
end = time.monotonic()

print('Duration for', len(preds), 'images: %.3f sec' % (end - begin))
print('FPS: %.3f' % (len(preds) / (end - begin)))
```

2897 fps on U250



<https://docs.inaccel.com/project/keras/>

Quantized ResNet50 on multiple Alveo cards



1 Application => 2 Alveo



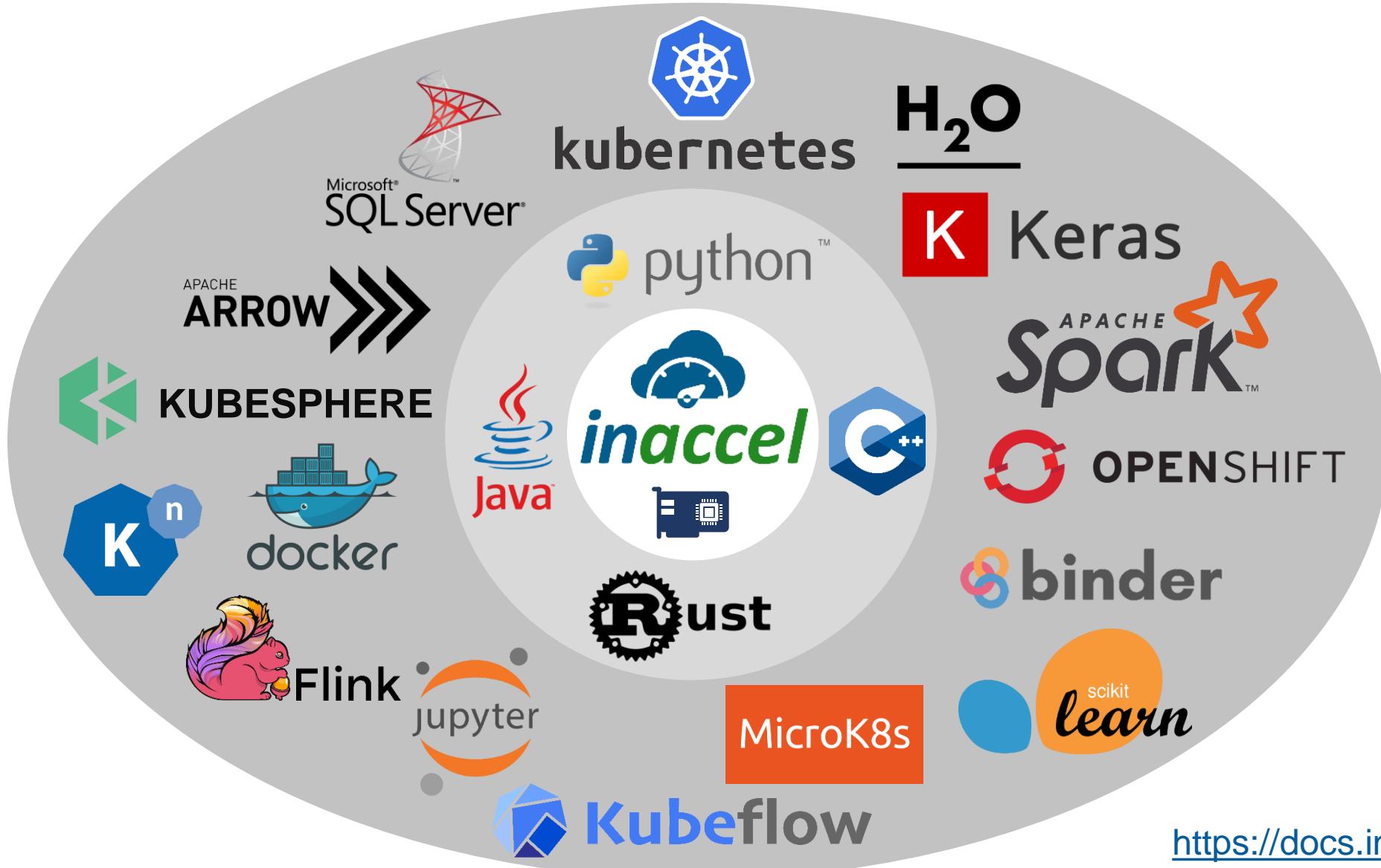
2 Applications => 1 Alveo



2 Applications => 2 Alveo



Successful Use cases, Integrations

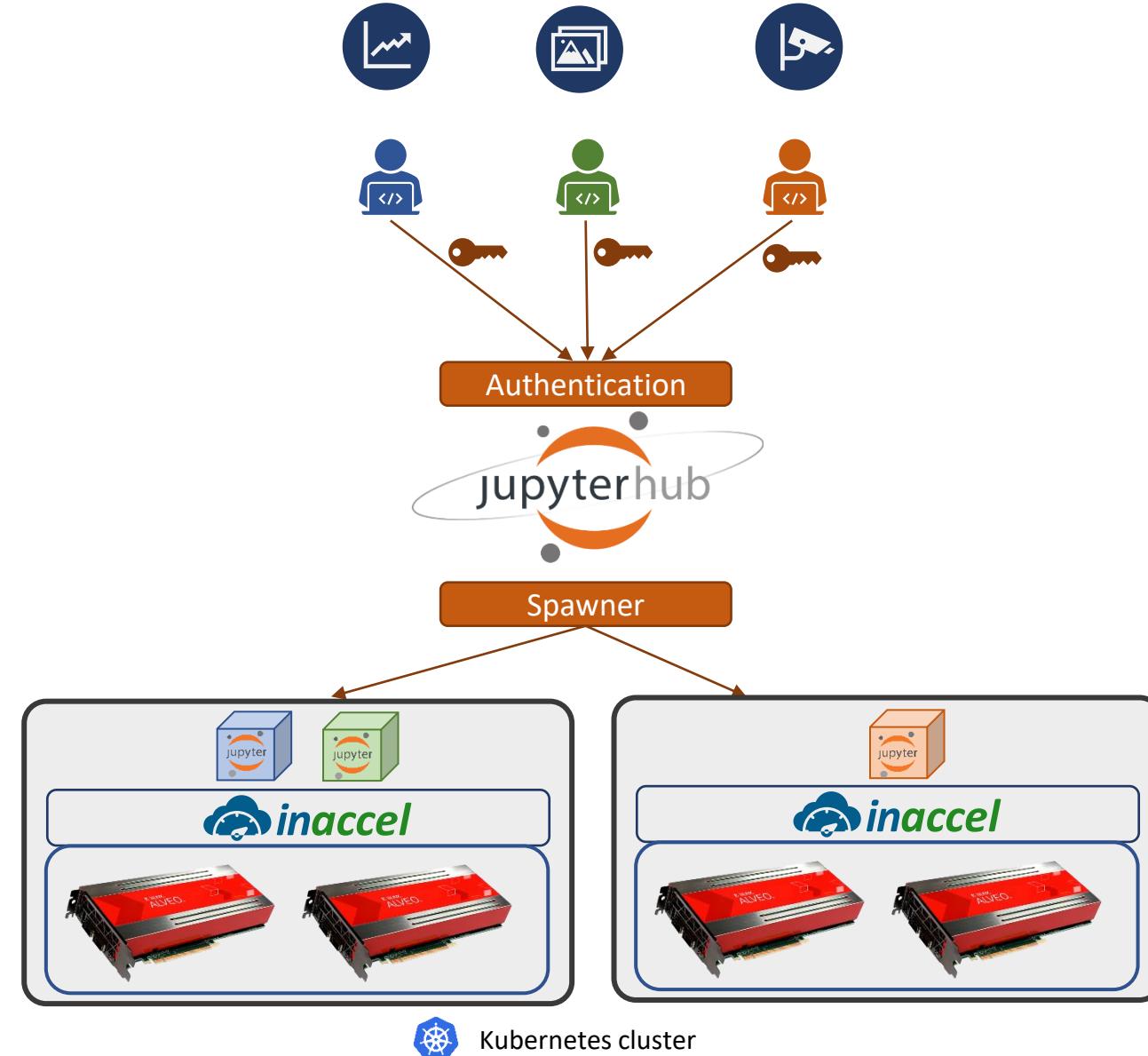


<https://docs.inaccel.com/>

JupyterHub on FPGAs



- > Instant acceleration of Jupyter Notebooks with zero code-changes
- > Offload the most computational intensive tasks on FPGA-based servers



Applications



Machine Learning

A photograph of a vast field of sunflowers stretching towards a horizon under a bright blue sky filled with large, white, fluffy clouds. The foreground is filled with the yellow flowers and green leaves of the sunflowers.

<https://blogs.intel.com/psg/inaccels-accelerated-ml-suite-boasts-spark-ml-performance-by-as-much-as-7x-on-fpga-based-alibaba-cloud-f1-instances/>



Quantitative Finance

<https://blogs.intel.com/psg/flumaion-accelerates-quantitative-financial-calculations/>



Genomics

Solution Brief

FPGA
Genomic Analytics

Acceleration of Sequence Alignment and Variant Calling for Genomic Analytics Using Intel® FPGAs

Introduction

Genomic Analytics aligns a selected genome to a reference genome to detect point mutations in the selected genome as compared to the reference genome. This technique is fundamental to the diagnosis and care of rare, inherited diseases as well as common diseases such as cancer. The ability to quickly analyze genomes is progressing towards personalized medicine, which will require the storage of many, many human genomes. There are 3 billion nucleotide base pairs in a human genome, so the challenge is how to store and process all of these genomes in a genome's lifetime.

As far as Coronavirus has been around the world, the genetic sequences of known viruses have been shared on GISAID, an online global platform for genomic data. One Coronavirus genome sequence contains 29k to 32k bases in the RNA strand. The SARS-CoV-2 virus is the most recent addition to GISAID. What does this tell us about the virus, named SARS-CoV-2, is spreading and evolving. But because they are so similar, it is difficult to see changes in cases and few see tell-tale differences, they are easy to overestimate.

Virologist Eeva Broberg of the Centre for Disease Prevention and Control¹ states that "there is no evidence that the SARS-CoV-2 virus has mutated significantly in Italy than at an undetected spread from Beijing".² This statement underscores the importance of using high performance computing to analyze genomic mutations.

"The very first SARS-CoV-2 sequence is every important, answered the ever basic question about the disease: what pathogen is causing it? The genetics that helped identify the pathogen were also used to track the disease's spread, had crossed into the human population just once. If the sequence could tell more, very little."
 The first SARS-CoV-2 sequence was published in January 2020. Since then, SARS-CoV-2 accumulates on average of about one to two mutations per month. Using high performance computing to analyze the mutations can help scientists make connections between cases, and gauge whether there might be unexpected mutations in the virus. This information can be used to predict where the virus might spread, better protect the population as the virus evolves and migrates in different regions of the world.

Scientists will be occurring the genomic diversity of these viral genome sequences for signs that the virus is getting more dangerous. Caution is warranted. An analysis by the National Science Review in China in March 2020 found that by March 2020 in the National Science Review argued that they fell into one of two distinct types, named S and L, and are distinguished by two mutations. Because the S and L variants have different mutation rates, the researchers in China and their authors concluded that the type L genome has evolved to become more aggressive variants.

Genomic scientists and researchers within different groups around the world have been trying to uncover genetic determinants of susceptibility, severity, and transmission of COVID-19. In the United States, the CDC is tracking the progression of the pandemic, disease caused by the SARS-CoV-2 coronavirus.

Using genomic analytics on the sequences of human viruses to track mutations is a key to the efficient processing of huge amounts of data. Fast genome sequencing is a

<https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/solution-sheets/genomic-analytics-using-intel-fpga.pdf>



Deep Learning

Solution Brief

Scalable Deep Learning Inference Accelerator using FPGAs

Authors:

Iannis Stamatis
Elias Koromilas
Chris Kachris
InAccel

Jing Lu
Xu Tianci
Inspur

Natalia Polakova
Intel

If you are responsible for building, testing, or deploying deep learning inference models:

- As a **business strategist or executive**: You will benefit from understand how to select the right technologies for deep learning to successfully generate increase the performance of your products and reduce the cost significantly.
- As a **technology decision-maker**: You will learn how to incorporate a cost-effective deep learning inference framework into your organization and back at the same time enjoy:
 - Higher performance
 - Lower latency
 - Lower cost
 - Lower energy consumption
 - Instantaneous deployment
 - Multitenant deployment

Scalable deployment of Inspur Deep Learning Inference Accelerator on multi-tenant Intel FPGA cluster

Executive Summary

Inference refers to the process of using a trained machine learning algorithm to make a prediction. After a neural network is trained, it is deployed as an inference engine to do predictions in production.

T2F is an open-source deep learning inference accelerator based on FPGA computing platform, developed by Inspur AI & HPC. A wide range of general purpose deep learning networks can be supported. Models from popular deep learning frameworks such as PyTorch, TensorFlow, and Caffe can be loaded into T2F easily by tools.

In this Solution Brief we show how InAccel orchestrator can be used to manage multiple InAccel scalable deployment of T2F to a cluster of FPGA.

We show how InAccel's orchestrator allows **easy deployment, scaling, resource management, and task scheduling** for FPGAs making it easier than ever, the deployment and the utilization of FPGA for Deep Learning inference.

<https://inaccel.com/wp-content/uploads/Inspur-Solution-Brief-Inference.pdf>



Video Analytics

Solution Brief

Accelerated Face Detection on a cluster of FPGAs using InAccel orchestrator

Scalable deployment of Face Detection on a cluster of Xilinx Alveo cards using InAccel orchestrator

Authors:

Ioannis Stamatatos
Elias Koromilas
Chris Kachris
InAccel

If you are responsible for building, testing, or deploying face detection or video analytics:

- As a business strategist or executive: You will better understand how to use InAccel's technologies for deep learning and face detection to successfully generate increase the performance of your system and reduce the cost.
- As a technology decision-maker: You will learn how to incorporate a cost-effective deep learning framework into your technology stack and at the same time enjoy:
 - Higher performance
 - Higher performance
 - Lower latency
 - Lower cost
 - Lower energy consumption
 - Instant Scalable deployment
 - Multitenant deployment

Executive Summary

Face Detection is the process of using a specific function in an image or a video frame to identify a face.

Face Detection is used in many application like security, entertainment, retail and other markets.

In this Solution Brief we show how InAccel orchestrator can be integrated with a widely used Face detection to allow multi-tenant scalable deployment of face detection on a cluster of Xilinx Alveo cards.

We show how InAccel's orchestrator allows easy deployment, scaling, resource management, and task scheduling for FPGAs making it easier than ever, the deployment and the utilization of FPGA for Face Detection. The same framework can be applied to any other video-analytics application.



https://inaccel.com/wp-content/uploads/Face-detection_inaccel.pdf

Data Science platforms



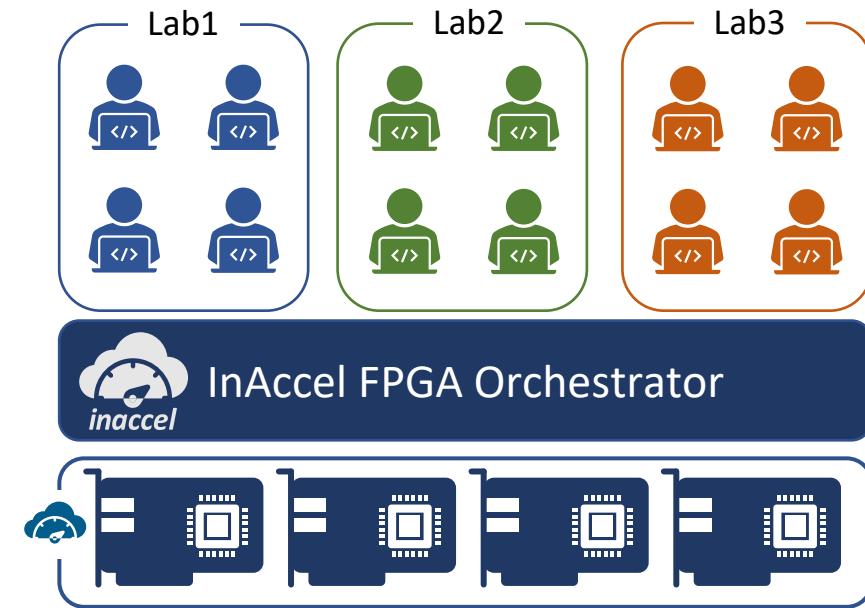
	GPU	FPGA
inaccel		XILINX intel
Azure Notebooks	NVIDIA	
Amazon SageMaker	NVIDIA	
colab	NVIDIA	
DataCamp	NVIDIA	
kaggle	NVIDIA	



Universities



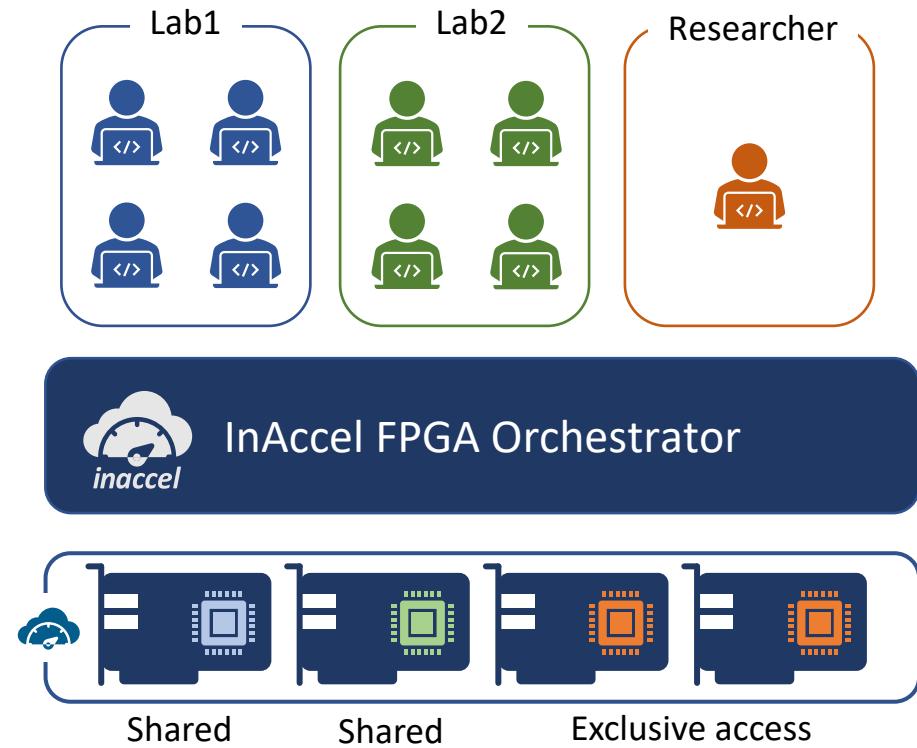
- > **How do you allow multiple students to share the available FPGAs?**
- > Many universities have limited number of FPGA cards that want to share with multiple students.
- > InAccel FPGA orchestrator allows multiple students to share one or more FPGAs seamlessly.
- > It allows students to just invoke the function that want to accelerate and InAccel FPGA manager performs the serialization and the scheduling of the functions to the available FPGA resources.



Universities



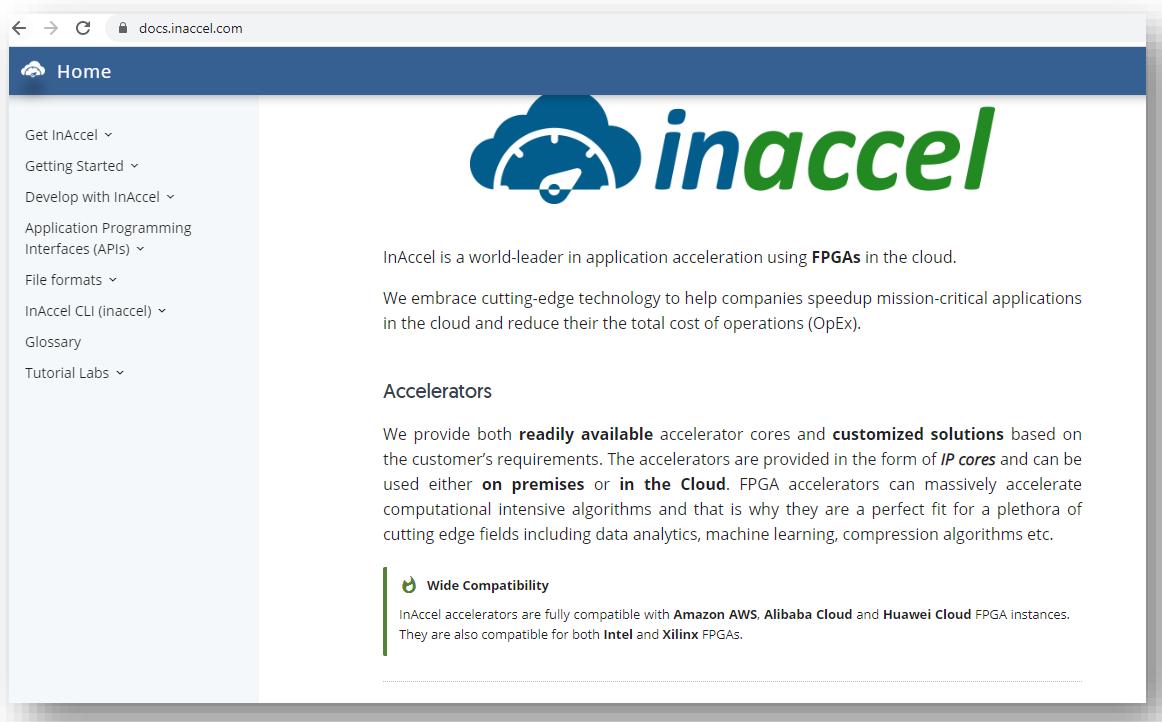
- > **But the researchers want exclusive access**
- > InAccel orchestrator allows to select which FPGA cards will be available for multiple students and which FPGAs can be allocated exclusively to researchers and Ph.D. students (so they can get accurate measurements for their papers).
- > The FPGAs that are shared with multiple students will perform on a best-effort approach (InAccel manager performs the serialization of the requested access) while the researchers have exclusive access to the FPGAs with zero overhead.



Test it on your prem or on your browser



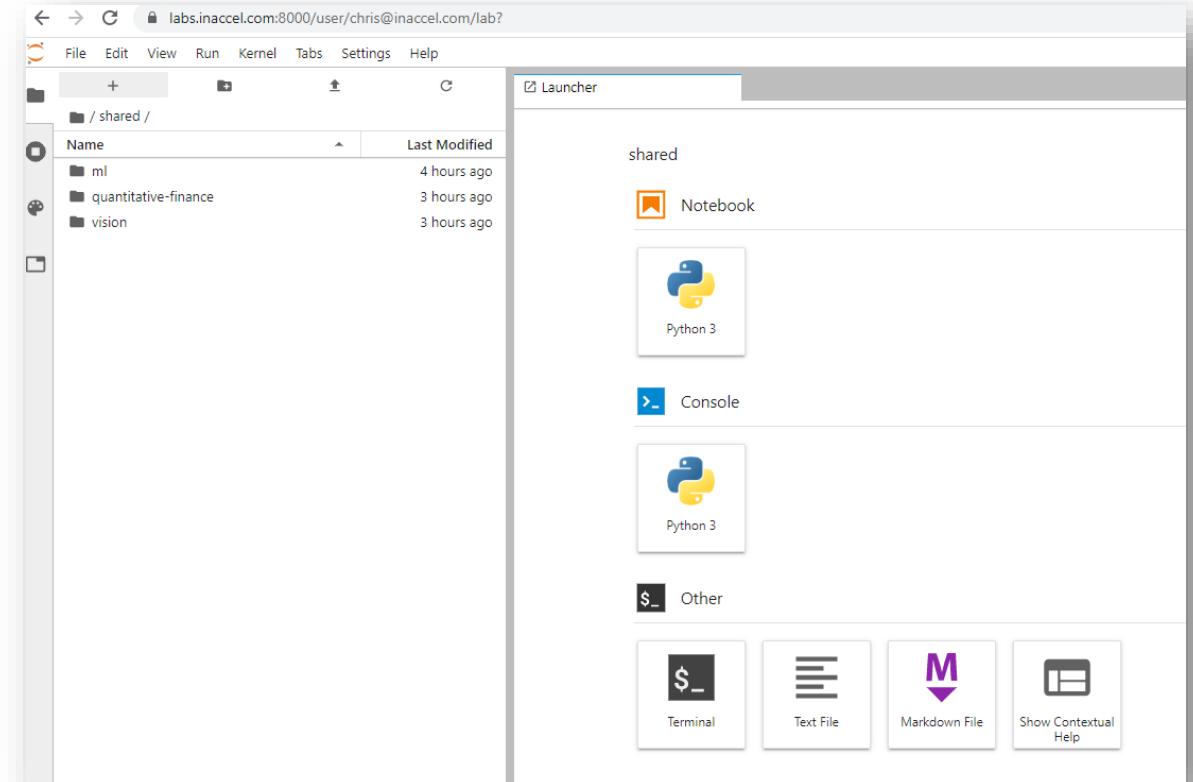
On-prem



A screenshot of a web browser displaying the InAccel documentation website at docs.inaccel.com. The page features a large InAccel logo at the top right. A sidebar on the left contains links to "Get InAccel", "Getting Started", "Develop with InAccel", "Application Programming Interfaces (APIs)", "File formats", "InAccel CLI (inaccel)", "Glossary", and "Tutorial Labs". The main content area includes a brief introduction, a section titled "Accelerators" with a detailed description of their offerings, and a "Wide Compatibility" sidebar stating they are compatible with Amazon AWS, Alibaba Cloud, and Huawei Cloud, as well as Intel and Xilinx FPGAs.

<https://docs.inaccel.com/>

Online - Browser



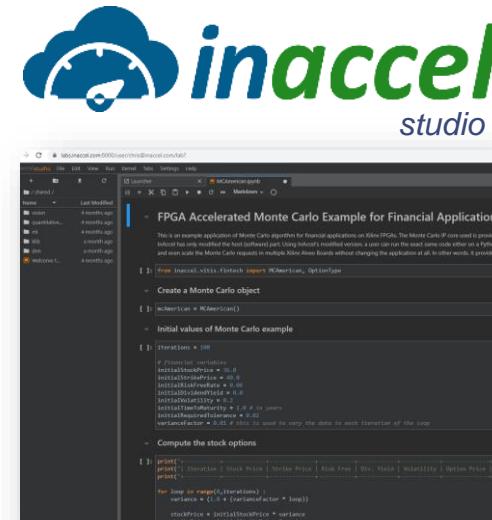
A screenshot of a web browser displaying the InAccel online development studio at <https://studio.inaccel.com>. The interface has a "Launcher" on the right side with options for "Notebook" (Python 3), "Console" (Python 3), "Terminal", "Text File", "Markdown File", and "Show Contextual Help". On the left, there's a file browser showing a folder named "shared/" containing subfolders "ml", "quantitative-finance", and "vision". The main workspace shows a code editor with Python code related to "quantitative-finance".

<https://studio.inaccel.com>

InAccel solutions



<https://inaccel.com/fpga-manager/>



<https://studio.inaccel.com>



<https://store.inaccel.com>

InAccel, Inc. Corporate overview

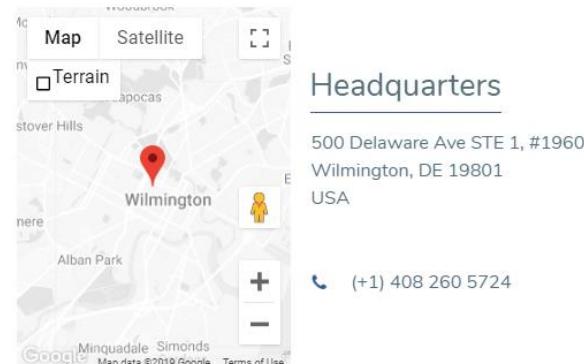


- > Founded in January 2018
- > Registered in Delaware, USA

> Membership:



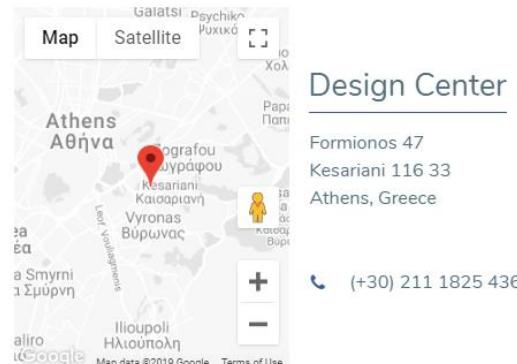
Registered
Technology
Partner



Headquarters

500 Delaware Ave STE 1, #1960
Wilmington, DE 19801
USA

(+1) 408 260 5724



Design Center

Formionos 47
Kesariani 116 33
Athens, Greece

(+30) 211 1825 436



MORPHEMIC



This project has received funding from the European Union's Horizon 2020 Research and Innovation program under grant agreement No. 871643.



inaccel

Application Acceleration, seamlessly

www.inaccel.com

info@inaccel.com

USA:

500 Delaware Ave STE 1, #1960
Wilmington, DE 19801
USA

Europe (Design Center):

Formionos 47
Kesariani 116 33
Athens, Greece