



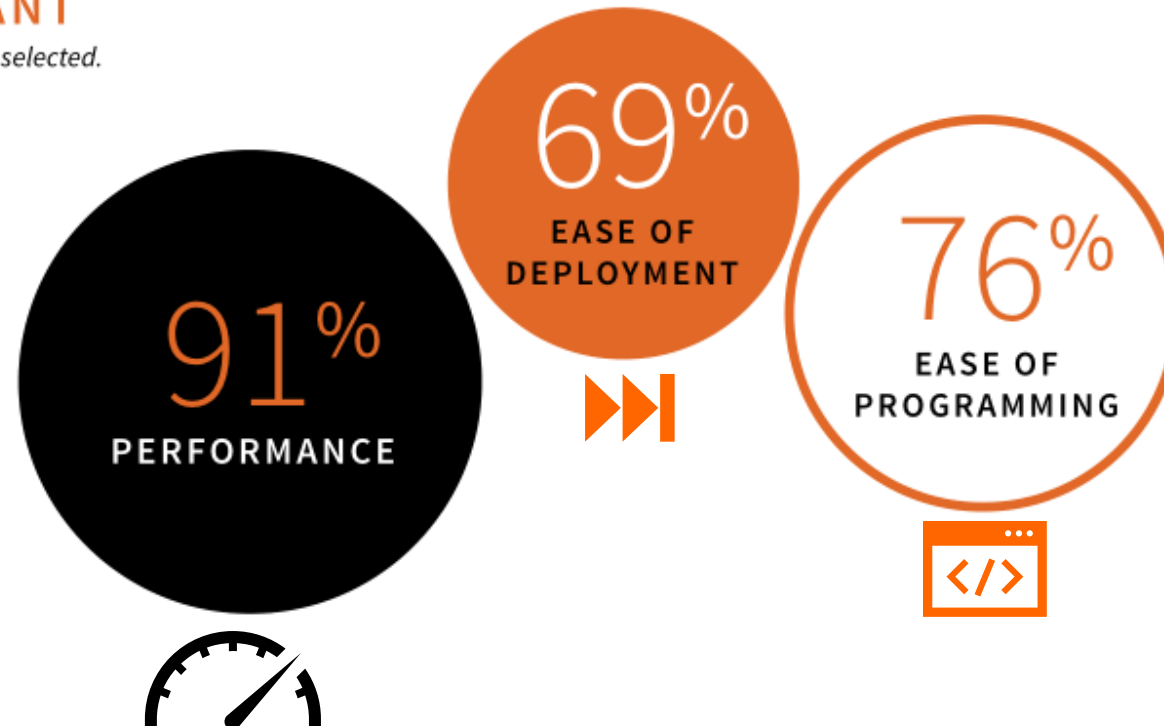
FPGA deployment and scaling made simple

Chris Kachris
chris@inaccel.com

What software developers want

% OF RESPONDENTS WHO CONSIDERED THE FEATURE
VERY IMPORTANT

More than one feature could be selected.

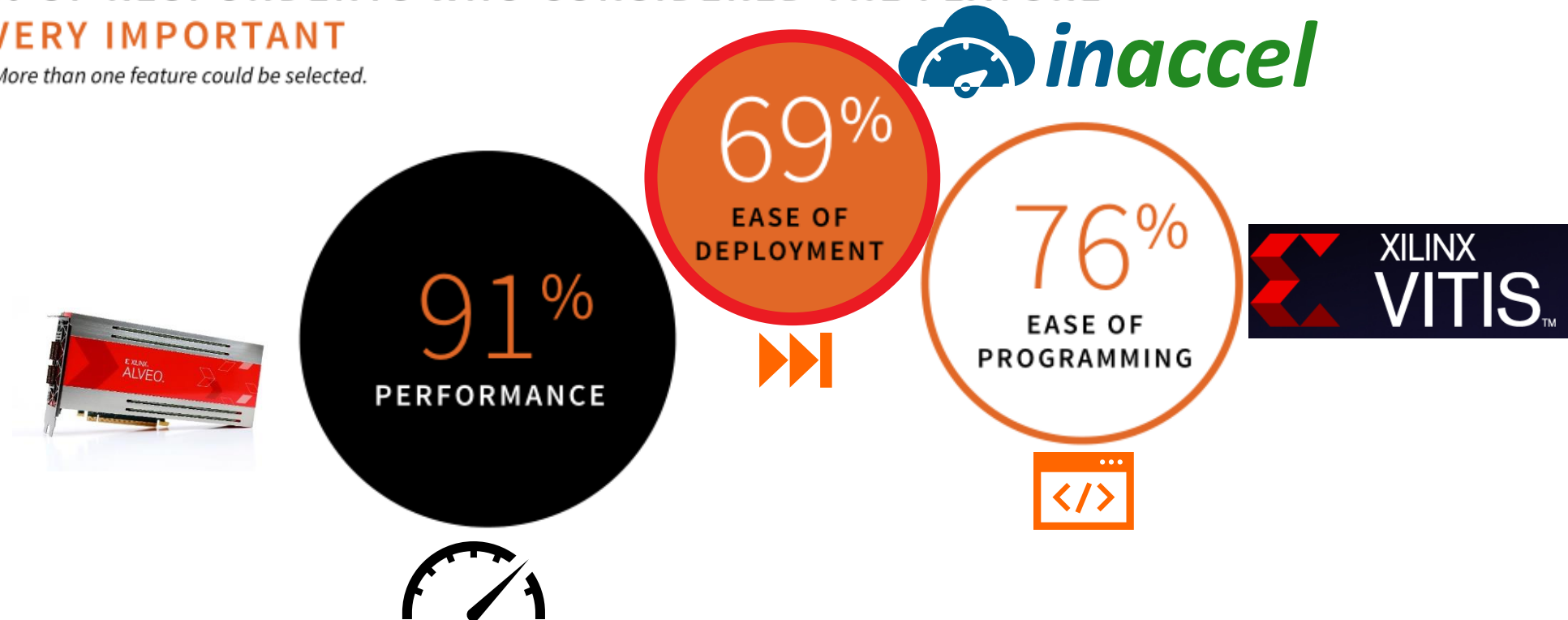


Source: Databricks, Apache Spark Survey 2016, Report

What software developers want

% OF RESPONDENTS WHO CONSIDERED THE FEATURE
VERY IMPORTANT

More than one feature could be selected.



Source: Databricks, Apache Spark Survey 2016, Report

> Help companies **speedup** their applications by using **accelerators** (FPGAs) seamlessly



> How?

Unique InAccel FPGA orchestrator

Automated deployment, scaling and management of FPGA clusters



Main challenges of FPGAs



> Hard to develop accelerators

Solved



> Hard to scale-out (kernels/cards)



> Hard to share (process/users/apps)







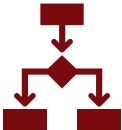


> Hard to deploy/invoke kernels



InAccel Products (Accelerators as IP)



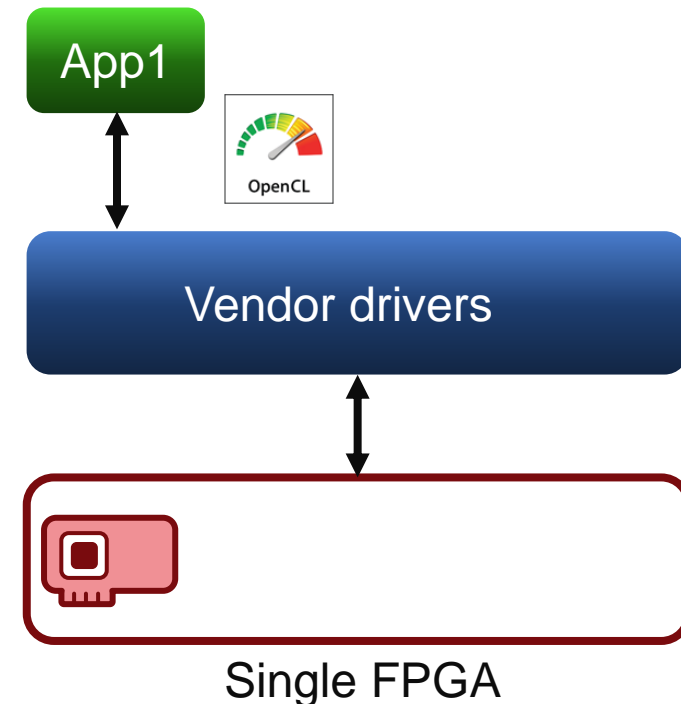
		 Speedup	 Cost reduction
	• Logistic Regression	15x	4x
	• K-means Clustering	14x	4x
	• Naïve-Bayes	5x	2x
	• FAISS (Similarity search)	2x	1.5x
	• Xgboost	6x	2x

GitHub

<https://github.com/inaccel>

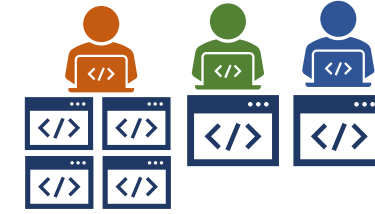
Current limitations for FPGA deployment

- > Currently only **one application** can talk to a single FPGA accelerator through **OpenCL**
- > Application can talk to a **single** FPGA.
- > Complex device sharing
 - From multiple threads/processes
 - Even from the same thread
- > Explicit allocation of the resources (memory/compute units)
- > User need to specify which FPGA to use (device ID, etc.)

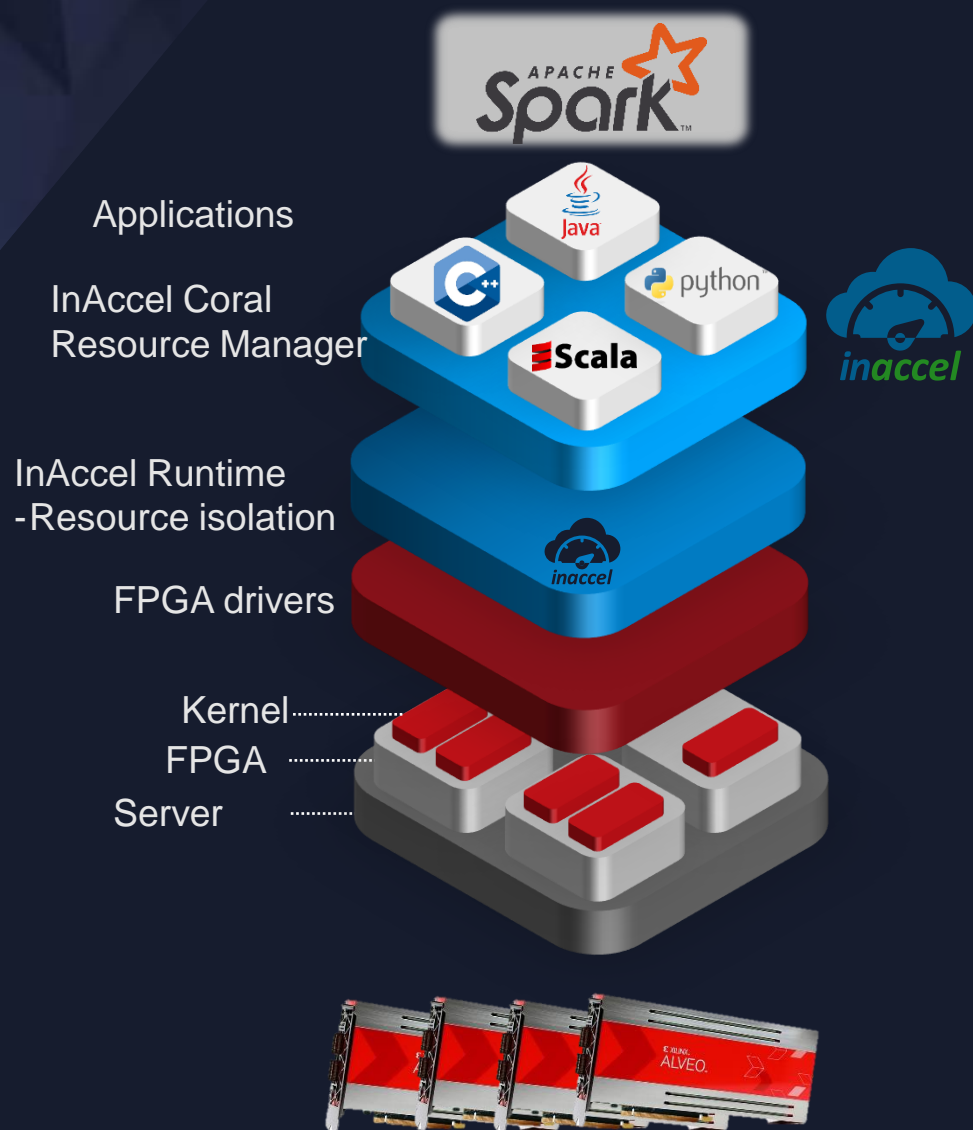


From single instance to data centers

- > Easy deployment
- > Instant scaling
- > Seamless sharing
- > Multiple-users
- > Multiple applications
- > Isolation
- > Privacy



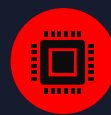
Scalable Orchestrator for FPGA clusters



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



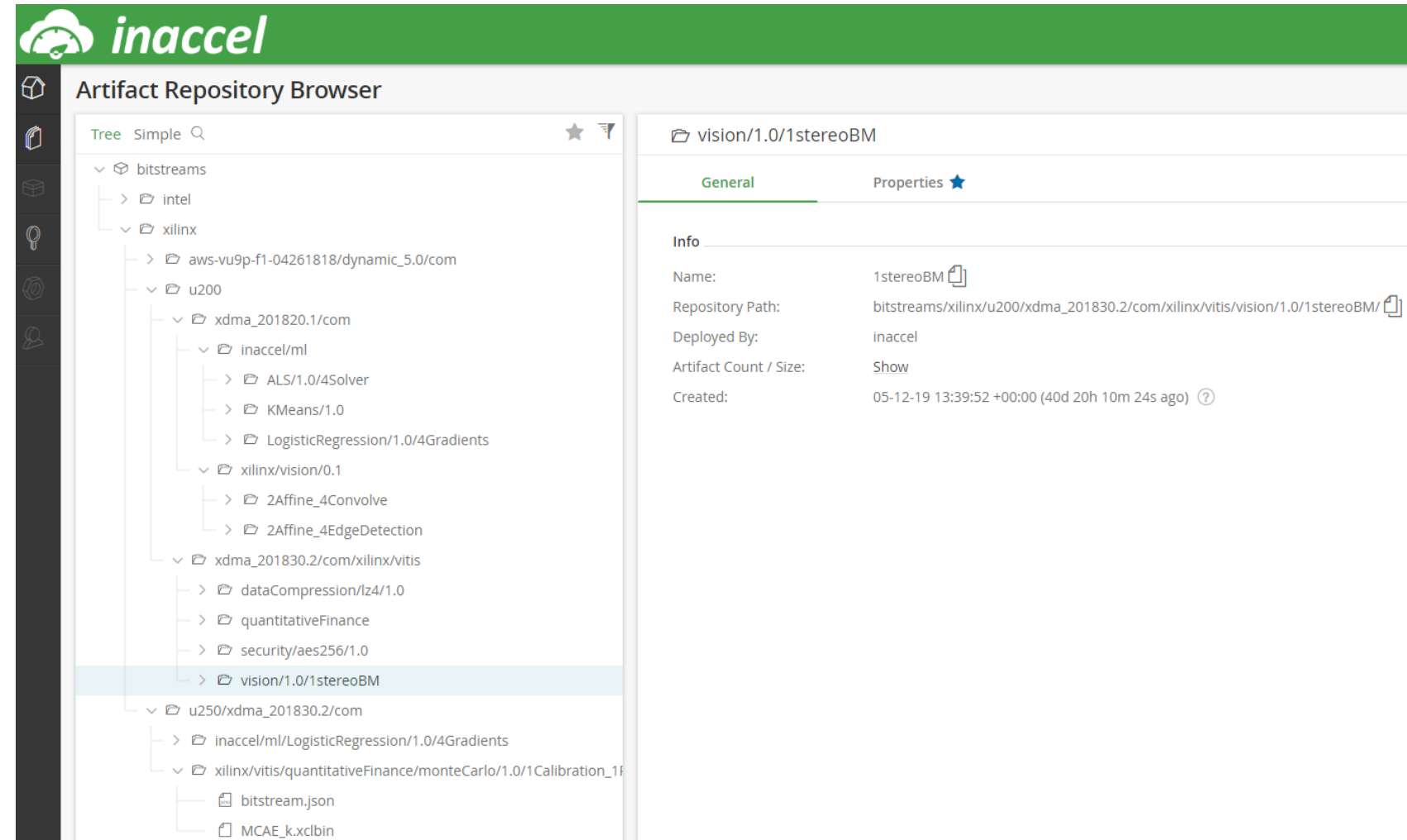
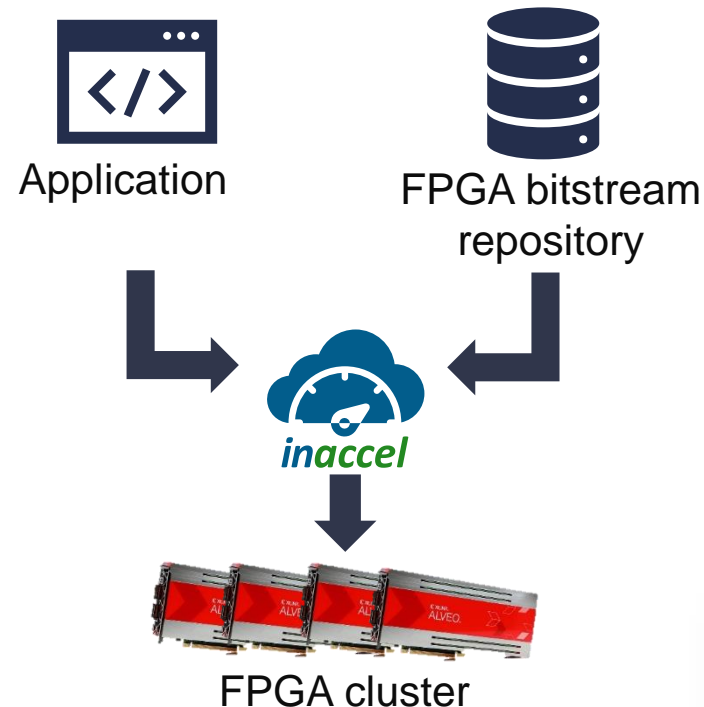
Seamless **sharing** of the FPGA cluster from multiple threads/processes/applications/users



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

FPGA bitstream repository

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

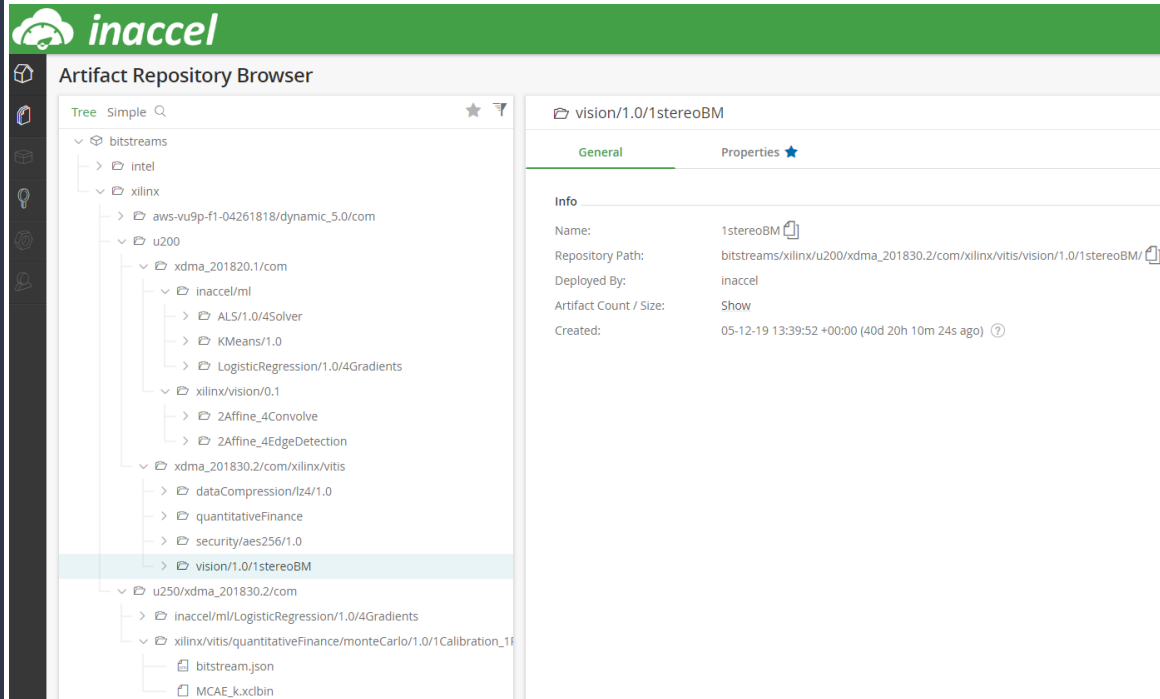


```
inaccel bitstream install [command options]
```

Single command

<https://store.inaccel.com>

FPGA repository - artifact



> Artifact: JSON file + bitstream

- >> Vendor
- >> FPGA cards
- >> Version
- >> Kernels
- >> Arguments

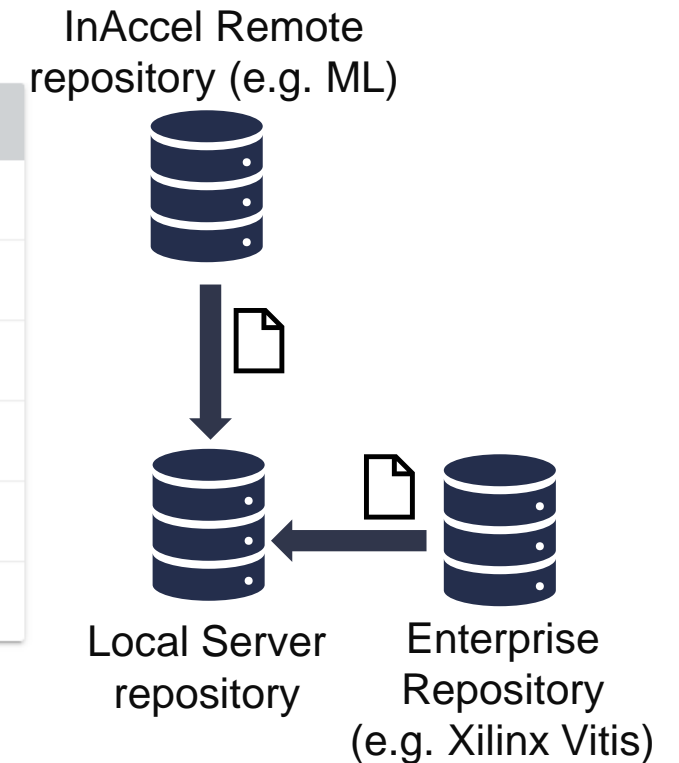
bitstream.json

```
1 {
2   "name": "krnl_stereopipeline.xclbin",
3   "bitstreamId": "com.xilinx.vitis.vision",
4   "version": "1.0",
5   "description": "Xilinx Vitis Stereo Block Matching Accelerator",
6   "platform": {
7     "vendor": "xilinx",
8     "name": "u200",
9     "version": "xdma_201830.2"
10  },
11  "kernels": [{
12    "name": [
13      "stereopipeline_accel"
14    ],
15    "kernelId": "stereoBM",
16    "arguments": [{
17      "type": "ap_uint<32>*",
18      "name": "img_L",
19      "access": "r"
20    },
21    {
22      "type": "ap_uint<32>*",
23      "name": "img_R",
24      "access": "r"
25    },
26    {
27      "type": "ap_uint<32>*",
28      "name": "img_disp",
```

FPGA repository CLI

- > Simple CLI interface for the local repository
- > Local repository can fetch artifacts from **external** links (e.g. Xilinx or InAccel repository) with **proprietary** artifacts

Command	Description
<code>inaccel bitstream decrypt</code>	Decrypt an FPGA binary
<code>inaccel bitstream encrypt</code>	Encrypt an FPGA binary
<code>inaccel bitstream parse</code>	Parse FPGA binary build-metadata
<code>inaccel bitstream install</code>	Install a bitstream to the local or a remote repository, from a local or a remote source
<code>inaccel bitstream list</code>	List all the bitstreams or detailed information for specific bitstreams in the local or a remote repository
<code>inaccel bitstream remove</code>	Remove one or more bitstreams from the local or a remote repository



Compatible with any Xilinx Vitis libraries



Generality

> Build your own repository of accelerators.

InAccel provides a stack of cores including Machine Learning. You can combine all these libraries, along with your own ones, seamlessly in the same application.

> Test it with any Xilinx Vitis/SDAccel example

InAccel Documentation
Overview
Accelerators ▾
FPGA Resource Manager ^
Getting Started
Deploying Accelerators
API Docs ▾
Programming Guides
[Examples](#)
Seamless Integration ▾
About ▾

automatically scale your accelerated solutions using in-house or 3rd-party accelerators, from high-level programming languages.

1. Prerequisites

- Docker Community Edition (CE)
- FPGA Runtime (Intel or Xilinx)
- Git | OpenJDK (Java Development Kit) 8 | Maven
- OpenCV 3.4.2

Amazon F1

Find **InAccel Default Image**, equipped with all the required development tools (and many more), available free on EC2 Community AMIs section.

2. Download InAccel

```
git clone https://bitbucket.org/inaccel/release.git inaccel && cd inaccel
```

Computer Vision

Acknowledgements:

The Computer vision accelerators (FPGA bitstreams/kernels) used for the purposes of this demonstration are written by developers at Xilinx.

Resources:

- SDAccel Examples - Vision

<https://docs.inaccel.com/latest/manager/examples/>

Simple invoking, deployment

No need for OpenCL

```
std::string binaryFile = argv[1];
size_t vector_size_bytes = sizeof(int) * DATA_SIZE;
cl_int err;
cl::Context context;
cl::Kernel krnl_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, krnl_vector_add = cl::Kernel(program, "vadd", &err));
        valid_device++;
    }
}
```

No need to allocate buffers
No need to specify bitstreams
No need to program specific device



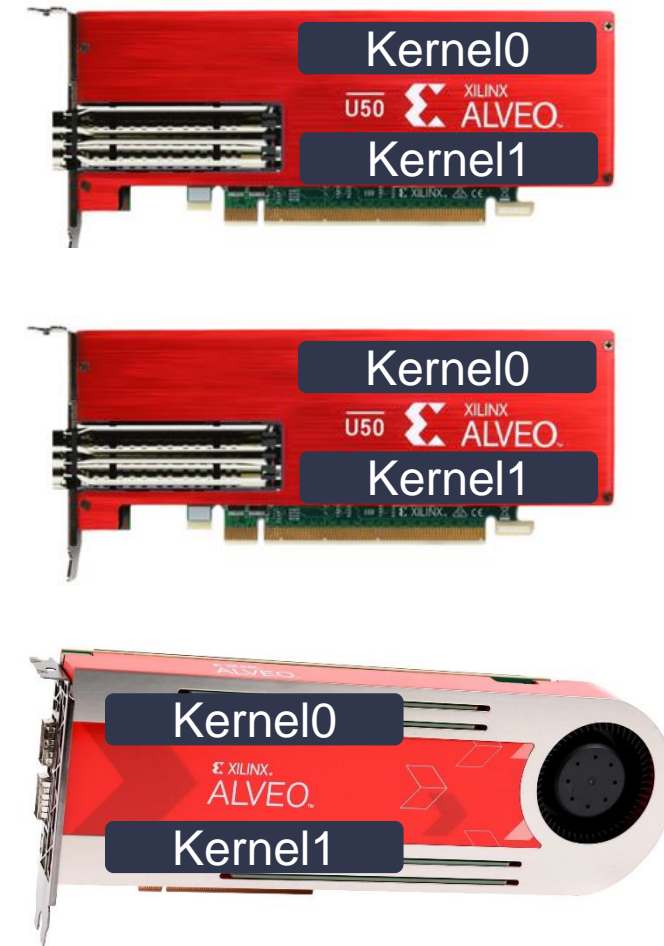
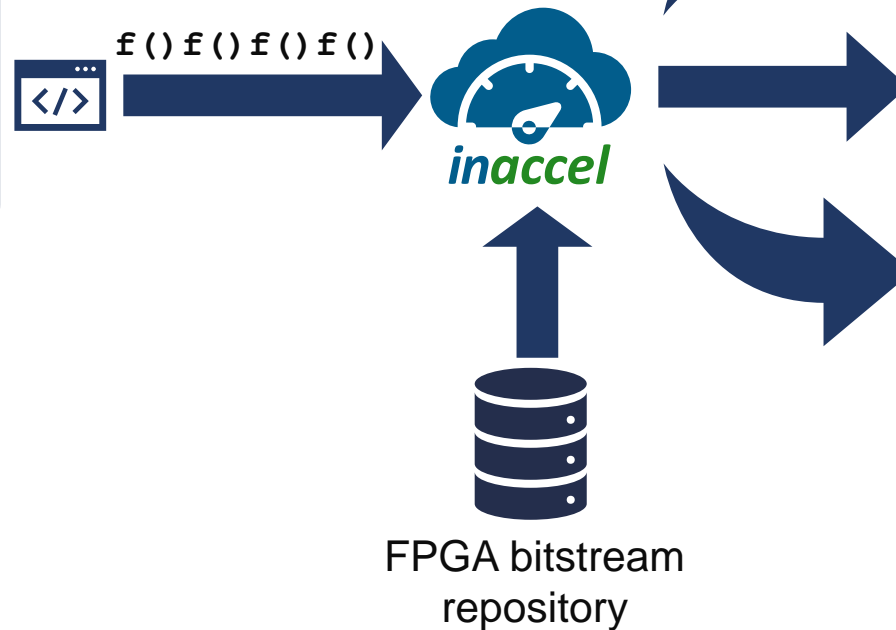
```
inaccel::Request add_req {"com.inaccel.math.vector.addition"};
add_req.Arg(a).Arg(b).Arg(c).Arg(size);
inaccel::Coral::Submit(add_req);
```

- Much simpler invoking
- Software-alike function invoking
- No need for OpenCL directives
- Same API for C/C++, Java, Python
- Native API

Instant scaling

```
inaccel coral start [command options]
```

Distribution of multi-thread applications to multiple clusters
With a single command



Example on scaling to 2 FPGA using the resource manager for logistic regression

```
+++1 ACTIVE FPGA+++

${SPARK_HOME}/bin/run-example \
> --inaccel \
> --jars ${INACCEL_SPARK_EXAMPLES} \
> inaccel.ml.LogisticRegressionExample \
* LogisticRegression Application *
  * LogisticRegression Training *
- Entering InAccel FPGA acceleration layer -
- LogisticRegression -
! Dataset transformation duration: 34.268114474 sec
! Model estimation duration: 250.664884735 sec
- Leaving inaccel FPGA acceleration layer -
! Time running {LabelIndexer - FeaturesScaler - LogisticRegression} Pipeline fit: 402.965841544 sec
* LogisticRegression Testing *
# accuracy: 0.8674716049382716

+++2 ACTIVE FGAs+++

${SPARK_HOME}/bin/run-example \
> --inaccel \
> --jars ${INACCEL_SPARK_EXAMPLES} \
> inaccel.ml.LogisticRegressionExample \
* LogisticRegression Application *
  * LogisticRegression Training *
- Entering InAccel FPGA acceleration layer -
- LogisticRegression -
! Dataset transformation duration: 33.949856611 sec
! Model estimation duration: 134.08420418 sec
- Leaving inaccel FPGA acceleration layer -
! Time running {LabelIndexer - FeaturesScaler - LogisticRegression} Pipeline fit: 285.278539904 sec
* LogisticRegression Testing *
# accuracy: 0.8674717283950617
```

1.86x speedup using 2 FGAs
simply by changing a line

Zero code changes

```
inaccel start --fpga=alveo:0
```



You specify how many
FGAs you want to use

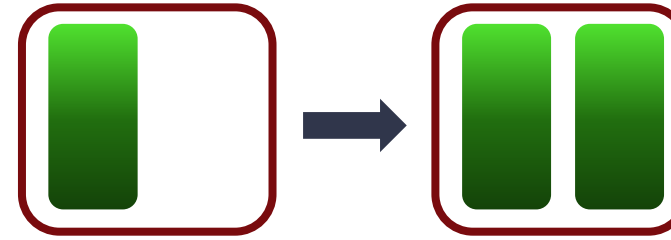
or

```
inaccel start --fpga=all
```

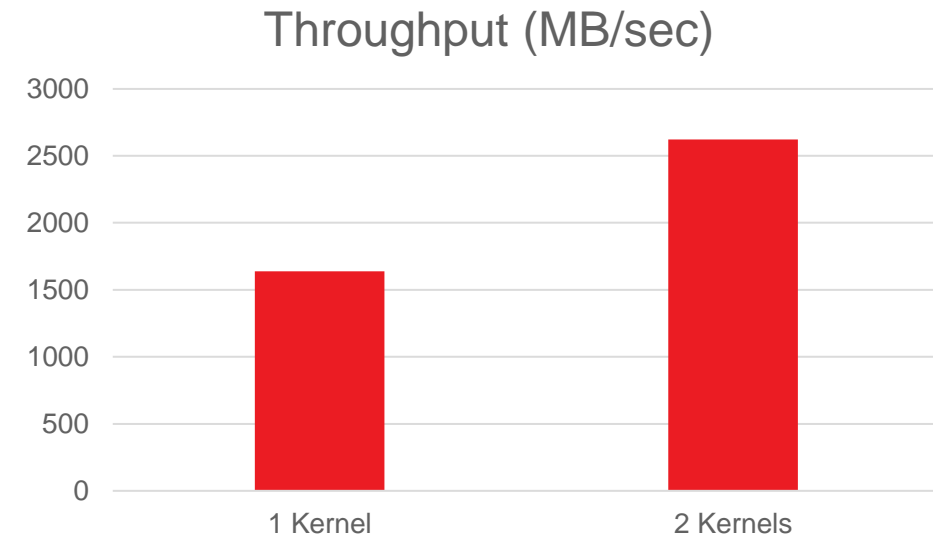

Instant Scalability

> Instant scalability to

- >> Multiple kernels
- >> Multiple FPGAs in the same server
- >> Multiple servers (using Kubernetes)



```
inaccel start --fpga=xilinx:0,xilinx:1
```



Lower than 2x speedup is due to the lower clock rate and PCI conflicts

Zero overhead, Improved Throughput



Zero overhead



Improved Performance



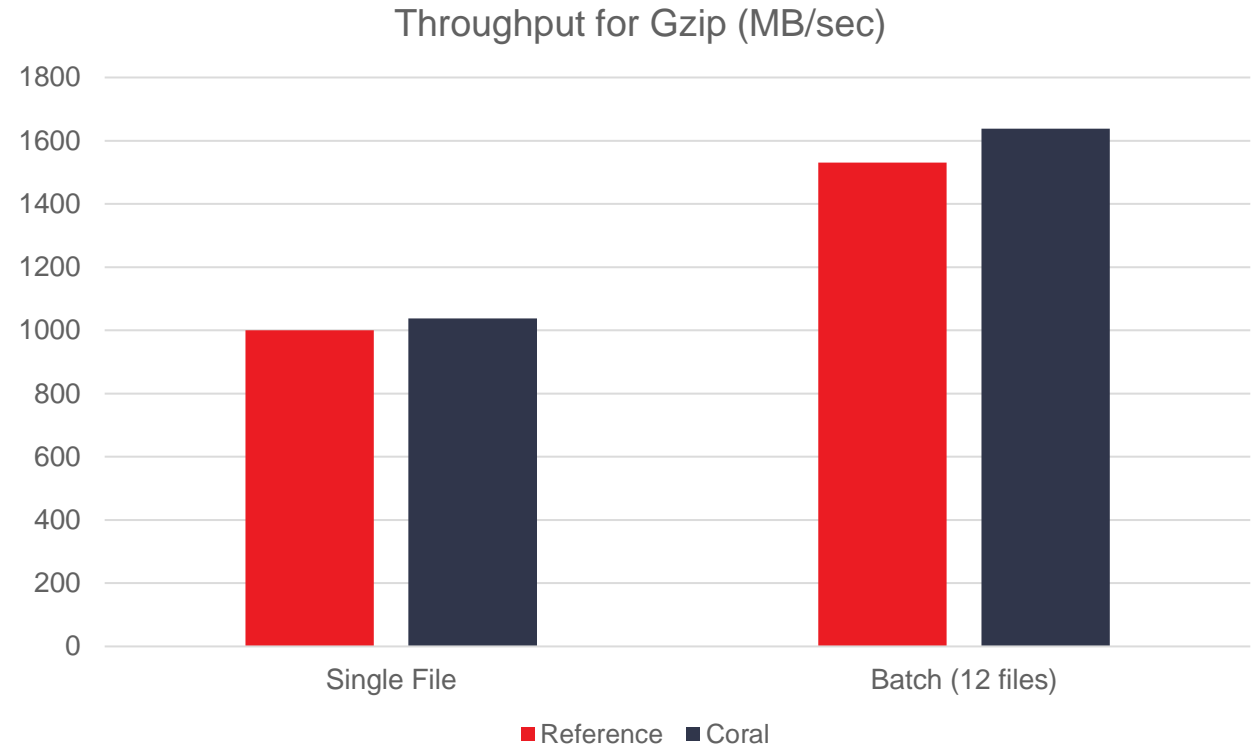
Instant scalability



Fully virtualization



Simpler programming



Optimized Pipeline

- > No need for OpenCL event to synchronize pipeline execution
- > Instant pipeline using InAccel FPGA manager => Optimized pipeline
 - >> Higher Throughput
 - >> Simplified code (no need for OpenCL events, barriers)
 - >> Useful for multiple kernels, multiple threads/applications

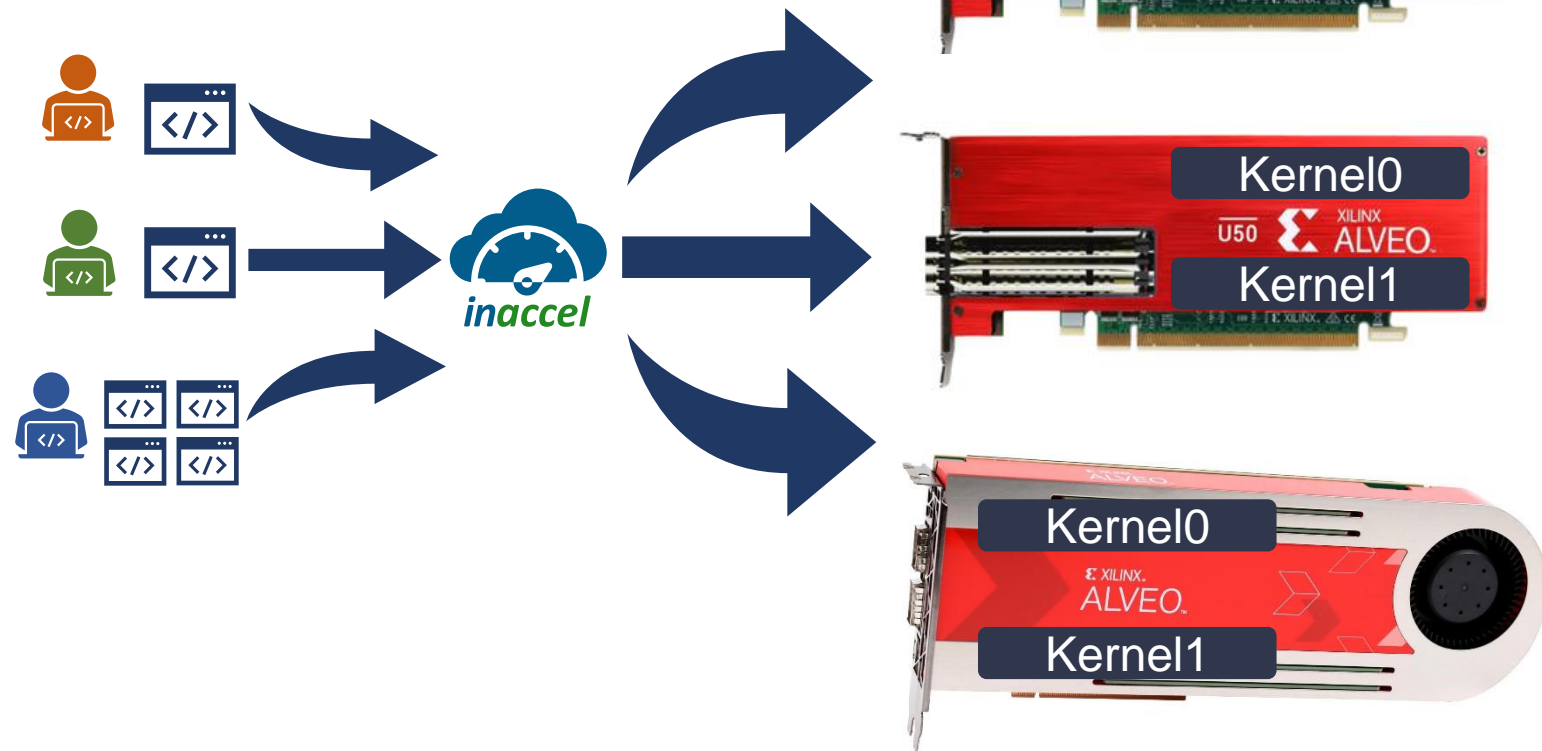


Seamless sharing of the resources

Resources sharing
from multiple
applications

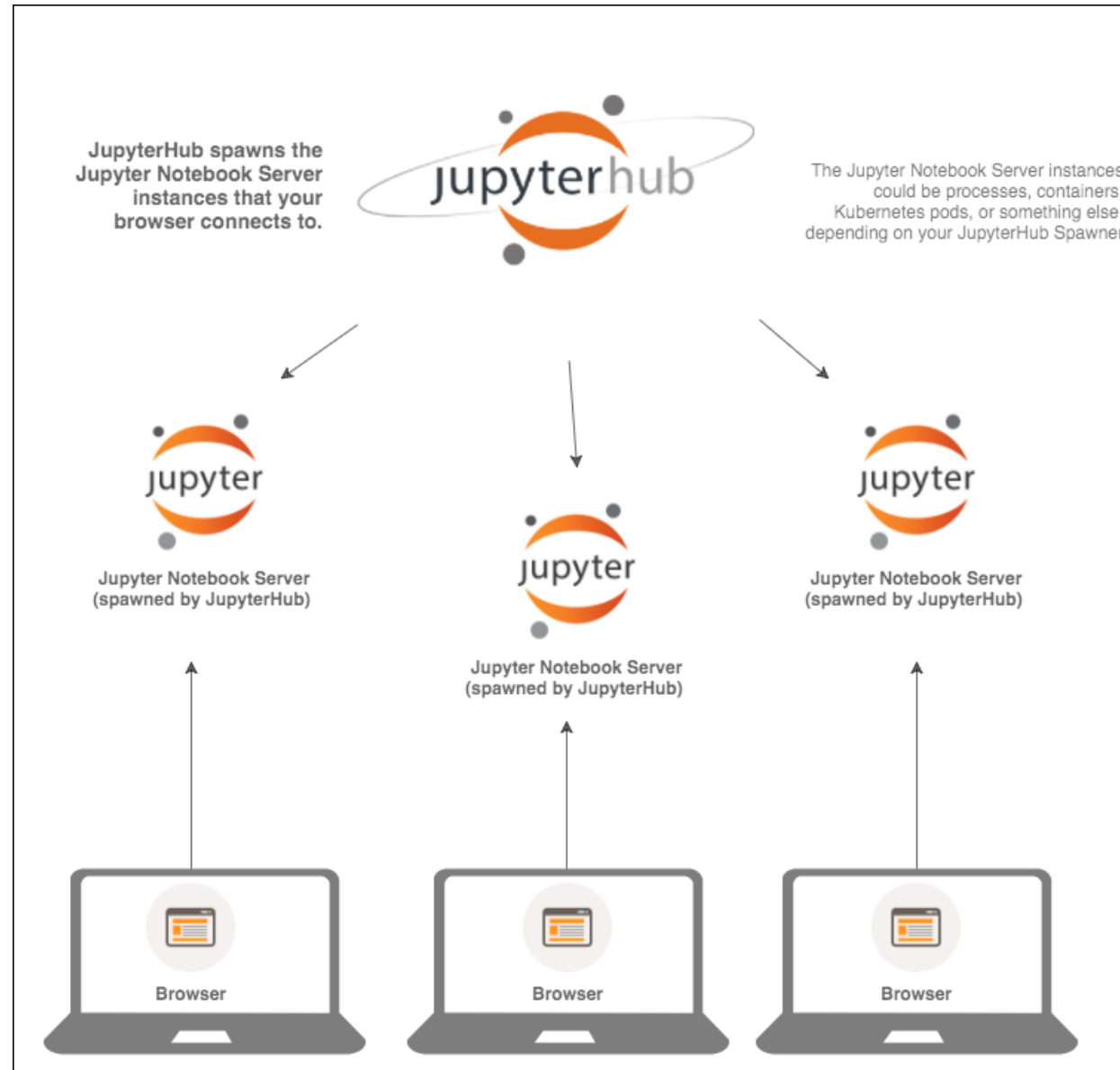
Utilization from:

- Multiple **threads**
- Multiple **processes**
- Multiple **applications**
- Multiple **users**
- Multiple **containers**



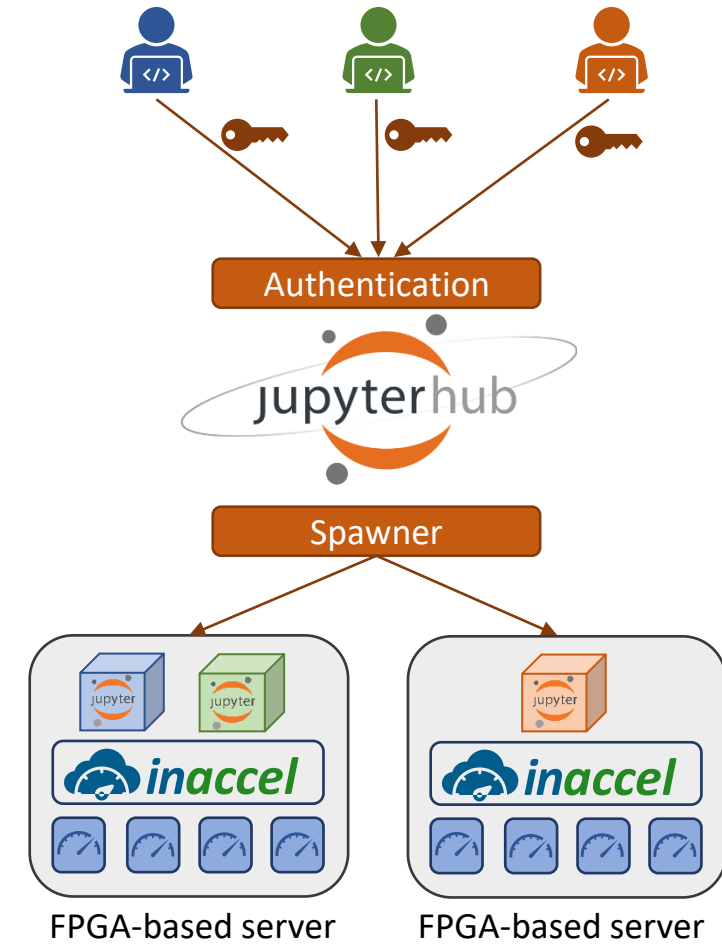
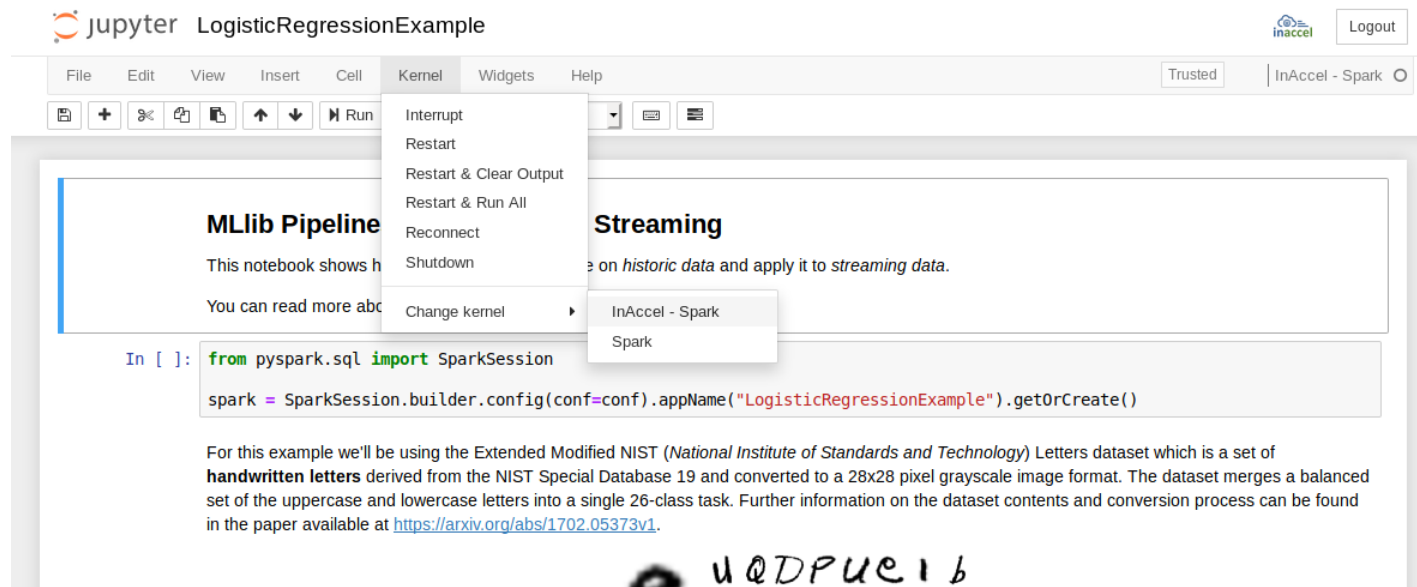
Jupyter - JupyterHub

- > **Deploy and run your FPGA-accelerated applications using Jupyter Notebooks**
- > **InAccel manager allows the instant deployment of FPGAs through JupyterHub**



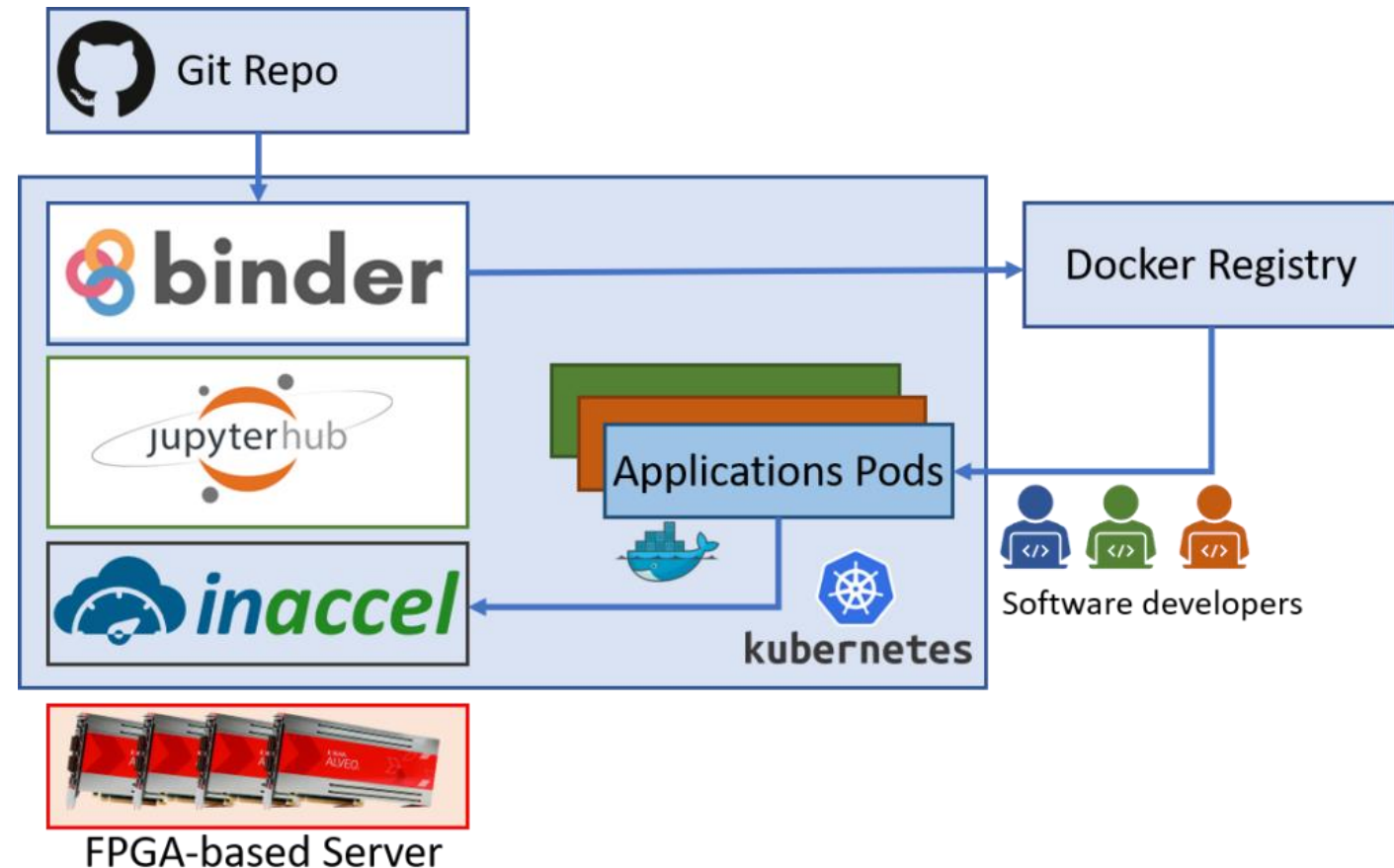
JupyterHub on FPGAs

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



BinderHub on FPGAs

- > BinderHub enables an end user to easily specify a desired computing environment from a Git repo
- > InAccel integration allows users to accelerate their applications from git repo using remote FPGA-based accelerators



<https://inaccel.com/deploying-fpgas-from-git-repos-instantly-using-binderhub/>

Unique FPGA monitoring tool

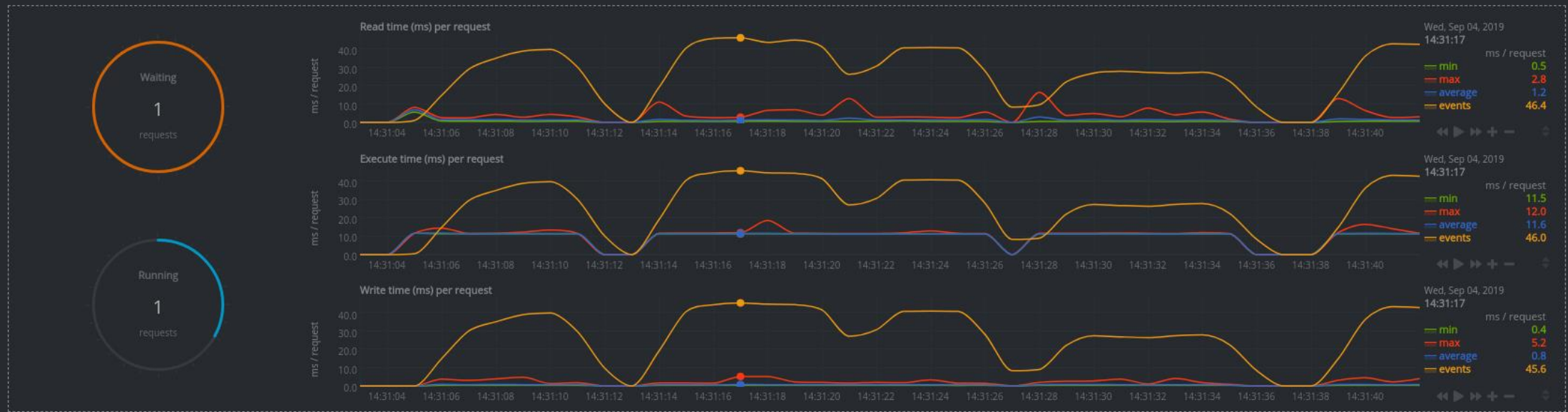


Xilinx 0 Xilinx 1

- com.inaccel.ml.logisticregression.gradients

com.inaccel.ml.logisticregression.gradients (1225630506)

Success: 3151 requests Failure: 0 requests

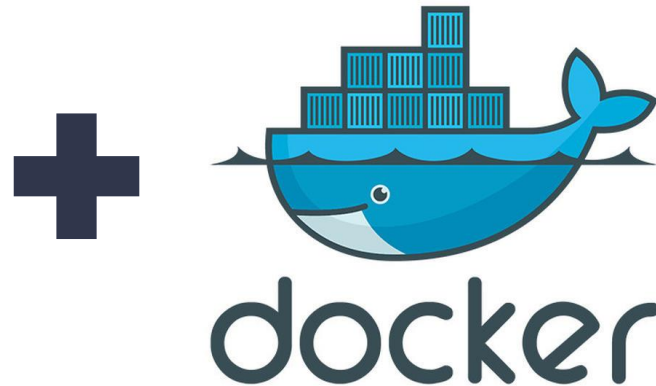
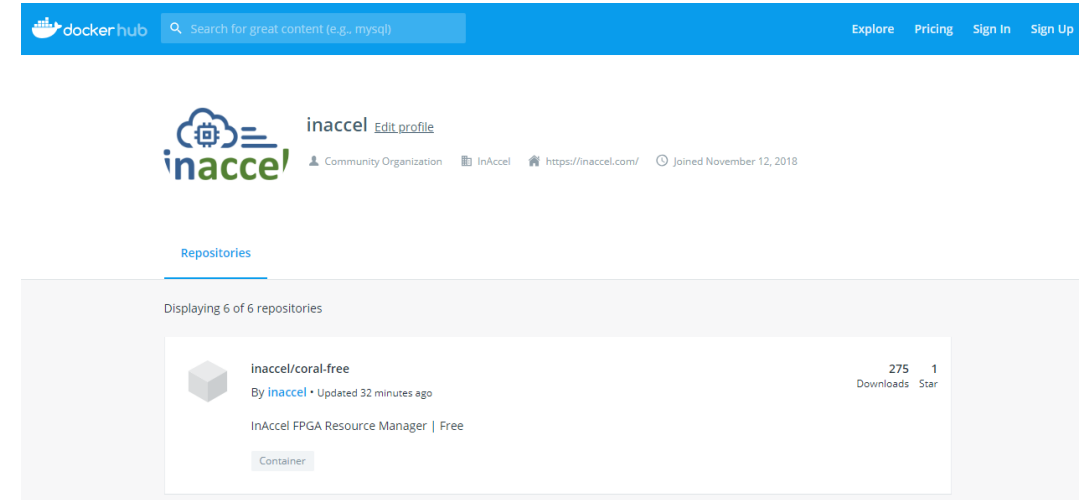


- > **Utilization of the kernels**
 - >> Time to read from DDR
 - >> Time to execute
 - >> Time to write back in DDR
- > **Useful for performance optimization**
- > **Support for multiple kernels, multiple FPGA cards**
- > **Integrated with Coral FPGA manager**

FPGA Manager deployment

Easy to Deploy

- > Launch a container with InAccel's **Docker** image or even deploy it as a daemonset on a **Kubernetes** cluster and enjoy acceleration services at the drop of a hat.
- > <https://hub.docker.com/u/inaccel/>



- Easy deployment
- Easy scalability
- Easy integration

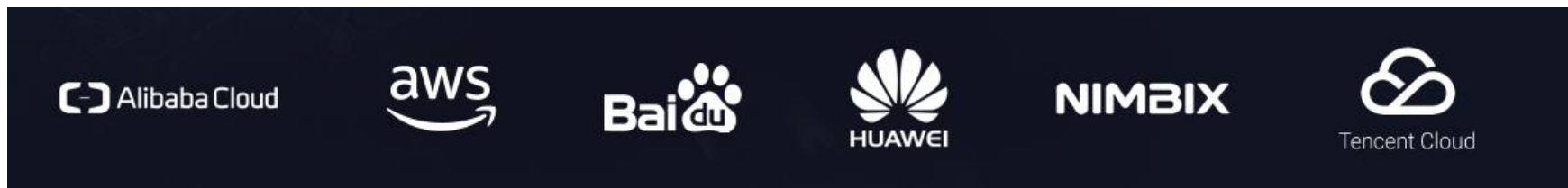
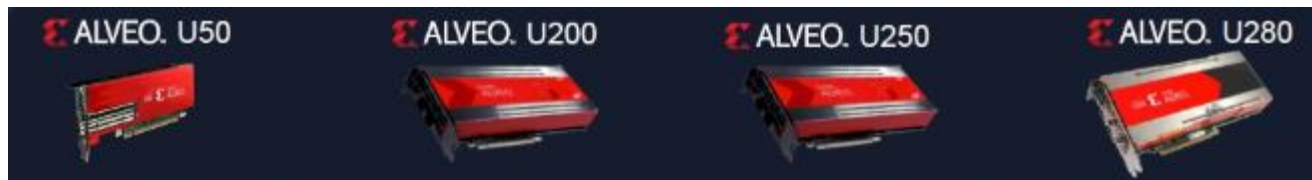
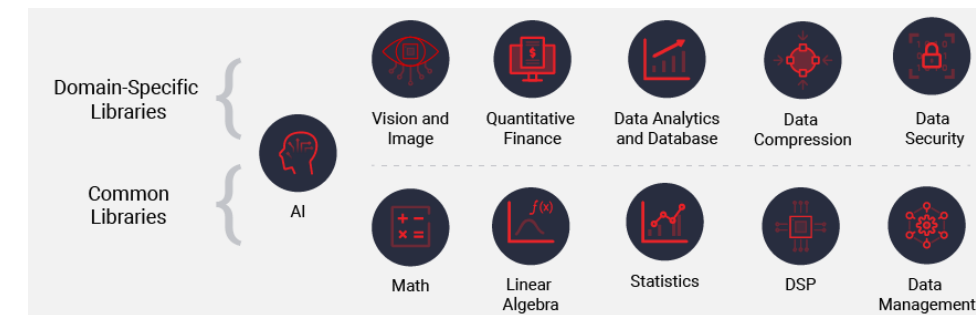
Compatibility with Xilinx Ecosystem

> FPGA repository for the Vitis Library

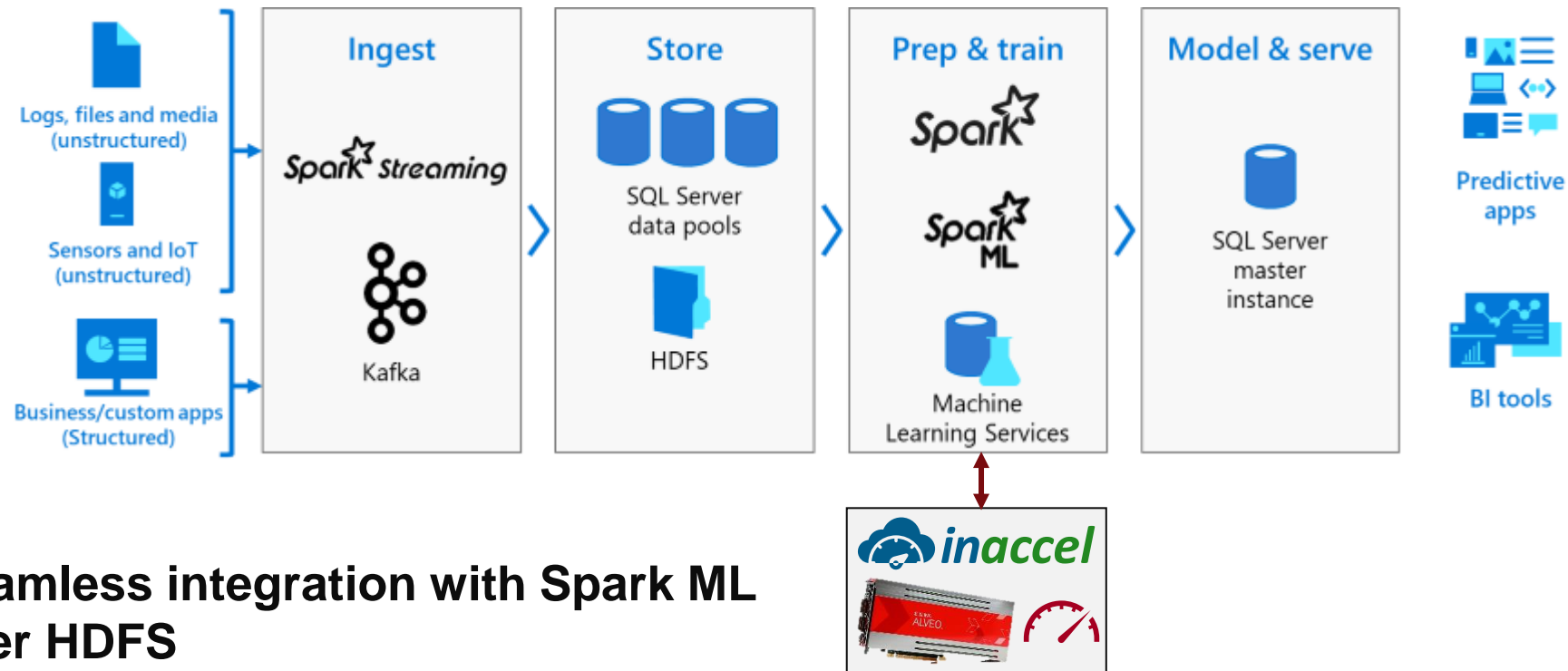
- >> Pre-compiled bitstreams for all Vitis Library for Alveo family cards (U50, U200, U250, U280)



> Scaling of Vitis libraries to multiple kernels, multiple Alveo cards, AWS f1.4x, f1.16x



Accelerated ML on top of SQL 2019



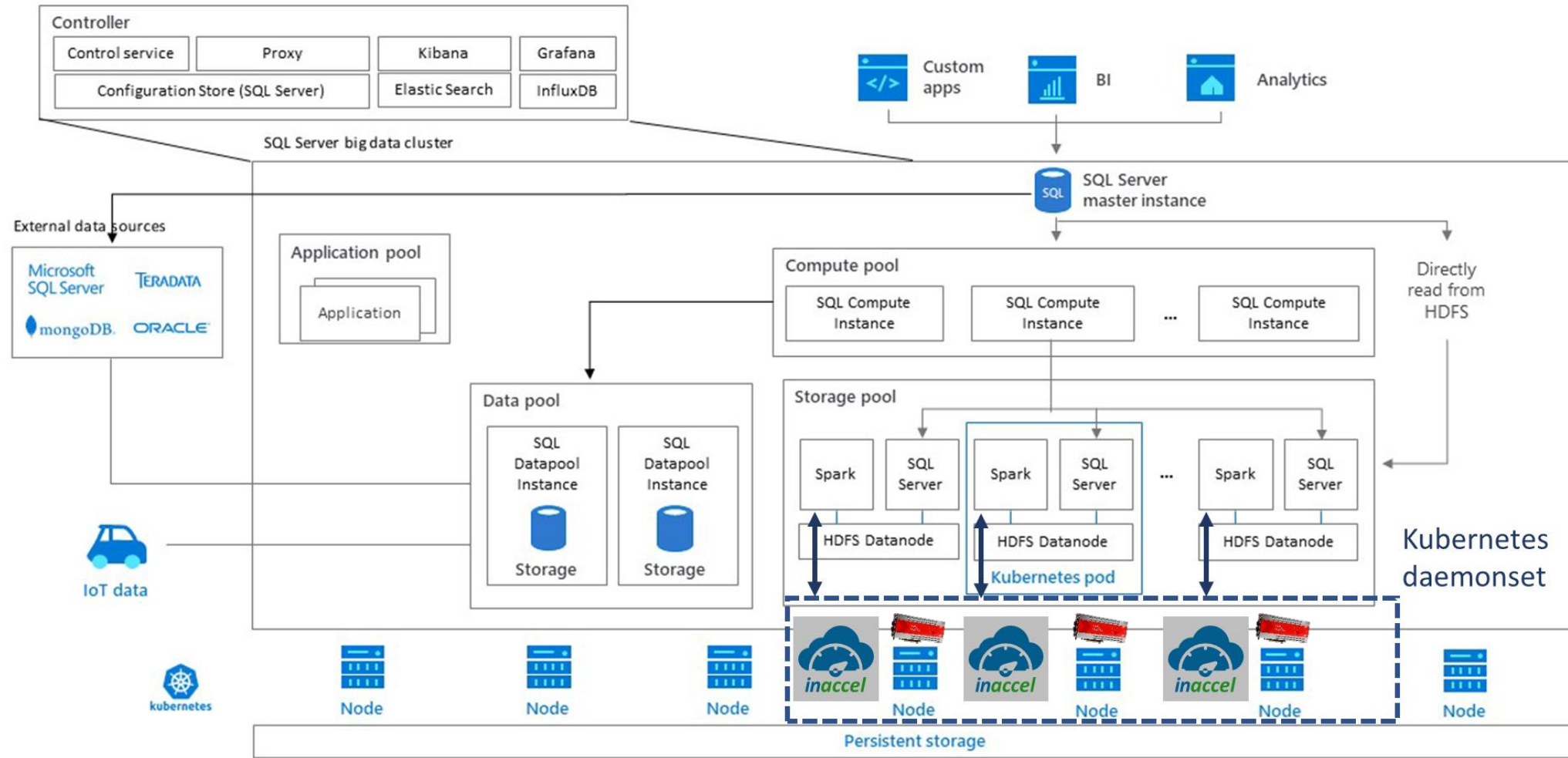
> Seamless integration with Spark ML over HDFS

- >> Zero Code changes
- >> Higher Performance
- >> Lower OpEx

https://www.youtube.com/watch?v=qu7ct_4CknM

<https://medium.com/@inaccel/accelerated-spark-ml-using-fpgas-on-top-of-microsoft-sql-server-2019-big-data-cluster-88acc130d780>

Integration with SQL Big Data Cluster 2019



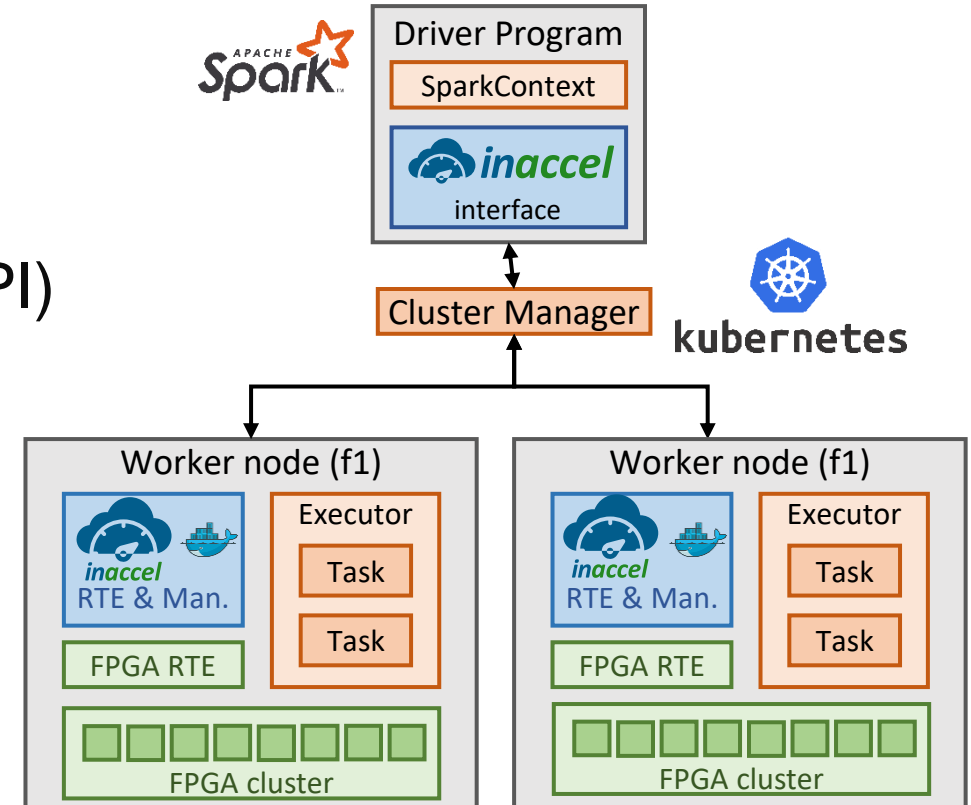
<https://medium.com/@inaccel/accelerated-spark-ml-using-fpgas-on-top-of-microsoft-sql-server-2019-big-data-cluster-88acc130d780>

InAccel's Coral manager integrated with Spark



> Integrated solution that allows

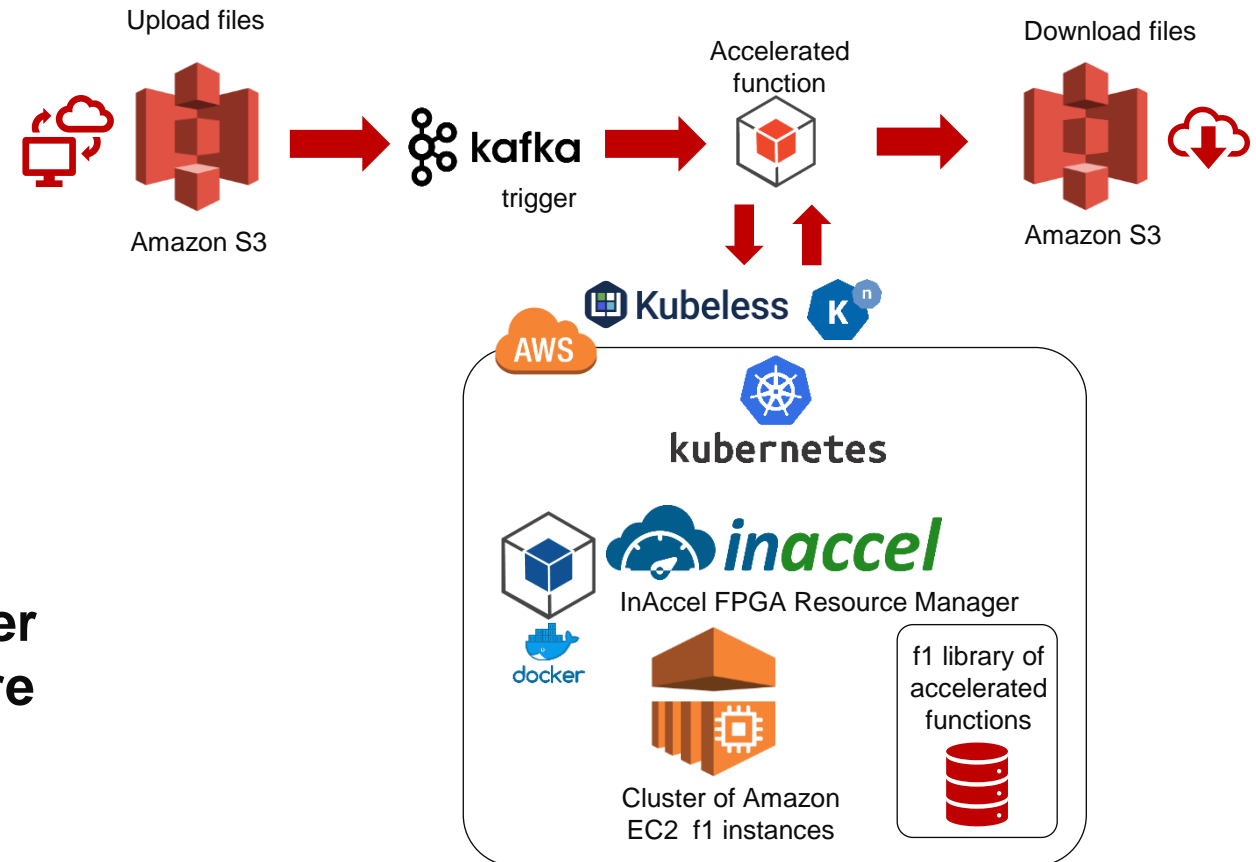
- >> Scale Up (1, 2, or 8 FPGAs per node)
- >> Scale Out to multiple nodes (using Spark API)
- >> Seamless integration
- >> Docker-based deployment
- >> No-need to specify which FPGA to use



Serverless deployment

- > Integrated framework for serverless deployment
- > Compatible with Kubernetes
- > 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>



- > The InAccel Coral runtime specification aims to specify the configuration, and execution interface for the efficient management of any accelerator-based hardware resource.
- > **Hook Developers**
 - > Hook developers can extend the functionality of a compliant runtime by hooking into an accelerator's lifecycle with an external application. Example use cases include sophisticated hardware configuration, advanced logging, IP licensing (hardware identification - authentication - decryption), etc.
- > **Platform Developers**
 - > Platform developers can build runtime implementations that expose diverse hardware resources and system configuration, containing low-level OS and hardware-specific details, on a particular platform.

<https://github.com/InAccel/runtime>

Pricing model (enterprise version)

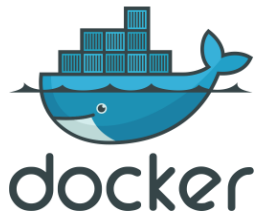
FPGA Resource manager

- > Pricing model per node (server)
- > Each node can have 1 to 8 FPGAs



FPGA Resource manager	20 servers or less	More than 20 servers
Monthly	\$300/servers	\$250/server
Yearly	\$3,000/server	\$2,500/server

Successful Use cases, Integrations



Partnerships





Making FPGA-based acceleration easy

info@inaccel.com

www.inaccel.com

