Building a Raspberry Pi Cluster

Part III —OpenMPI, Python, and Parallel Jobs

Garrett Mills · Follow

10 min read · Apr 29, 2019

This is Part III in my series on building a small-scale HPC cluster. Be sure to check out Part I and Part II.

In the first two parts, we set up our Pi cluster with the SLURM scheduler and ran some test jobs using R. We also looked at how to schedule many small jobs using SLURM. We also installed software the easy way by running the package manager install command on all of the nodes simultaneously.

In this part, we're going to set up OpenMPI, install Python the "better" way, and take a look at running some jobs in parallel to make use of the multiple cluster nodes.

Part 1: Installing OpenMPI

https://www.open-mpi.org/

OpenMPI is an open-source implementation of the Message Passing Interface concept. An MPI is a software that connects processes running across multiple computers and allows them to communicate as they run. This is what allows a single script to run a job spread across multiple cluster nodes.

We're going to install OpenMPI the easy way, as we did with R. While it is possible to install it using the "better" way (spoiler alert: compile from source), it's more difficult to get it to play nicely with SLURM.

We want it to play nicely because SLURM will auto-configure the environment when a job is running so that OpenMPI has access to all the resources SLURM has allocated the job. This saves us a *lot* of headache and setup for each job.

1.1 — Install OpenMPI

To install OpenMPI, SSH into the head node of the cluster, and use `srun` to install OpenMPI on each of the nodes:

```
$ sudo su -
# srun --nodes=3 apt install openmpi-bin openmpi-common libopenmpi3
libopenmpi-dev -y
```

(Obviously, er.)

1.2 — Test it

Believe it or not, that's all it took to get OpenMPI up and running on our cluster. Now, we're going to create a very basic hello-world program to test it out.

1.2.1 — Create a program.

We're going to create a C program that creates an MPI cluster with the resources SLURM allocates to our job. Then, it's going to call a simple print command on each process.

Create the file `/clusterfs/hello_mpi.c` with the following contents:

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char** argv){
    int node;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &node);

    printf("Hello World from Node %d!\n", node);

    MPI_Finalize();
}
```

Here, we include the `mpi.h` library provided by OpenMPI. Then, in the main function, we initialize the MPI cluster, get the number of the node that the current process will be running on, print a message, and close the MPI cluster.

1.2.2 — Compile the program.

We need to compile our C program to run it on the cluster. However, unlike with a normal C program, we won't just use `gcc` like you might expect. Instead, OpenMPI provides a compiler that will automatically link the MPI libraries.

Because we need to use the compiler provided by OpenMPI, we're going to grab a shell instance from one of the nodes:

```
login1$
node1$ cd /clusterfs
node1$ mpicc hello_mpi.c
node1$ ls
a.out* hello_mpi.c
node1$ exit
```

The `a.out` file is the compiled program that will be run by the cluster.

1.2.3 — Create a submission script.

Now, we will create the submission script that runs our program on the cluster. Create the file `/clusterfs/sub_mpi.sh`:

```
#!/bin/bash

cd $SLURM_SUBMIT_DIR

# Print the node that starts the process
echo "Master node: $(hostname)"

# Run our program using OpenMPI.
# OpenMPI will automatically discover resources from SLURM.
mpirun a.out
```

1.2.4 — Run the job.

Run the job by submitting it to SLURM and requesting a couple of nodes and processes:

```
$ cd /clusterfs
$ sbatch --nodes=3 --ntasks-per-node=2 sub_mpi.sh
Submitted batch job 1211
```

This tells SLURM to get 3 nodes and 2 cores on each of those nodes. If we have everything working properly, this should create an MPI cluster with 6 nodes. Assuming this works, we should see some output in our `slurm-XXX.out` file:

```
Master
Hello World from Node 0!
Hello World from Node 1!
Hello World from Node 2!
Hello World from Node 3!
Hello World from Node 4!
Hello World from Node 5!
```

Part 2: Installing Python (the “better” way)

Okay, so for a while now, I’ve been alluding to a “better” way to install cluster software. Let’s talk about that. Up until now, when we’ve installed software on the cluster, we’ve essentially did it individually on each node. While this works, it quickly becomes inefficient. Instead of duplicating effort trying to make sure the same software versions and environment is available on every single node, wouldn’t it be great if we could install software centrally for all nodes?

Well, luckily a new feature in the modern Linux operating system allows us to do just that: compile from source! (/s) Rather than install software through the individual package managers of each node, we can compile it from source and configure it to be installed to a directory in the shared storage. Because the architecture of our nodes is identical, they can all run the software from shared storage.

This is useful because it means that we only have to maintain a single installation of a piece of software and its configuration. On the downside, compiling from source is a *lot* slower than installing pre-built packages. It’s also more difficult to update. Trade-offs.

In this section, we’re going to install Python3 from source and use it across our different nodes.

2.0 — Prerequisites

In order for the Python build to complete successfully, we need to make sure that we have the libraries it requires installed on one of the nodes. We’ll only install these on one node and we’ll make sure to only build Python on that node:

```
$ srun
node1$ sudo apt install -y build-essential python-dev python-
setuptools python-pip python-smbus libncursesw5-dev libgdbm-dev
libc6-dev zlib1g-dev libsqlite3-dev tk-dev libssl-dev openssl
libffi-dev
```

Hooo boy. That's a fair number of dependencies. While you can technically build Python itself without running this step, we want to be able to access Pip and a number of other extra tools provided with Python. These tools will only compile if their dependencies are available.

Note that these dependencies don't need to be present to *use* our new Python install, just to compile it.

2.1 — Download Python

Let's grab a copy of the Python source files so we can build them. We're going to create a build directory in shared storage and extract the files there. You can find links to the latest version of Python here, but I'll be installing 3.7. Note that we want the "Gzipped source tarball" file:

```
$ cd /clusterfs && mkdir build && cd build
$ wget https://www.python.org/ftp/python/3.7.3/Python-3.7.3.tgz
$ tar xvzf Python-3.7.3.tgz
... tar output ...
$ cd Python-3.7.3
```

At this point, we should have the Python source extracted to the directory `/clusterfs/build/Python-3.7.3`.

2.2 — Configure Python

For those of you who have installed software from source before, what follows is pretty much a standard `configure;make;make install`, but we're going to change the prefix directory.

The first step in building Python is configuring the build to our environment. This is

done with the `./configure` command. Running this by itself will configure Python to install to th

; to pass it a custom fl ıge. Buckle up, because this may take a while:

```
$ mkdir /clusterfs/usr        # directory Python will install to
$ cd /clusterfs/build/Python-3.7.3
$ srun --nodelist=node1 bash  # configure will be run on node1
node1$ ./configure \
            --enable-optimizations \
            --prefix=/clusterfs/usr \
            --with-ensurepip=install
...configure output...
```

2.3 — Build Python

Now that we've configured Python to our environment, we need to actually compile the binaries and get them ready to run. We will do this with the `make` command. However, because Python is a fairly large program, and the RPi isn't exactly the biggest workhorse in the world, it will take a little while to compile.

So, rather than leave a terminal open the whole time Python compiles, we're going to use our shiny new scheduler! We can submit a job that will compile it and we can just wait for the job to finish. To do this, create a submission script in the Python source folder:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=4
#SBATCH --nodelist=node1

cd $SLURM_SUBMIT_DIR

make -j4
```

This script will request 4cores on `node1` and will run the make command on those cores. Make is the software tool that will compile Python for us. Now, just submit the job from the login node:

```
$ cd /c
$ sbatch sub_build_python.sh
Submitted batch job 1212
```

Now, we just wait for the job to finish running. It took about an hour for me on an RPi 3B+. You can view its progress using the `squeue` command, and by looking in the SLURM output file:

```
$ tail -f slurm-1212.out        # replace "1212" with the job ID
```

2.4 — Install Python

Lastly, we will install Python to the `/clusterfs/usr` directory we created. This will also take a while, though not as long as compiling. We can use the scheduler for this task. Create a submission script in the source directory:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=node1

cd $SLURM_SUBMIT_DIR

make install
```

However, we don't want just any old program to be able to modify or delete the Python install files. So, just like with any normal program, we're going to install Python as root so it cannot be modified by normal users. To do this, we'll submit the install job as a root user:

```
$ sudo su -
# cd /clusterfs/build/Python-3.7.3
# sbatch sub_install_python.sh
Submitted batch job 1213
```

Again, you d have a functional

2.5 — Test it out.

We should now be able to use our Python install from any of the nodes. As a basic first test, we can run a command on all of the nodes:

```
$ srun --nodes=3 /clusterfs/usr/bin/python3 -c "print('Hello')"
Hello
Hello
Hello
```

We should also have access to `pip`:

```
$ srun --nodes=1 /clusterfs/usr/bin/pip3 --version
pip 19.0.3 from /clusterfs/usr/lib/python3.7/site-packages/pip
(python 3.7)
```

The exact same Python installation should now be accessible from all the nodes. This is useful because, if you want to use some library for a job, you can install it once on this install, and all the nodes can make use of it. It's cleaner to maintain.

Part 3: A Python MPI Hello-World

Finally, to test out our new OpenMPI and Python installations, we're going to throw together a quick Python job that uses OpenMPI. To interface with OpenMPI in Python, we're going to be using a fantastic library called mpi4py.

For our demo, we're going to use one of the demo programs in the mpi4py repo. We're going to calculate the value of pi (the number) in parallel.

3.0 — Prerequisites

Before we can write our script, we need to install a few libraries. Namely, we will install the mpi4py library, and numpy. NumPy is a package that contains many useful structures and operations used for scientific computing in Python. We can

install these libraries through pip, using a batch job. Create the file `/clusterfs/calc-pi/sub_inst`


```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1

/clusterfs/usr/bin/pip3 install numpy mpi4py
```

Then, submit the job. We have to do this as root because it will be modifying our Python install:

```
$ cd /clusterfs/calc-pi
$ sudo su
# sbatch sub_install_pip.sh
Submitted batch job 1214
```

Now, we just wait for the job to complete. When it does, we should be able to use the mpi4py and numpy libraries:

```
$ srun bash
node1$ /clusterfs/usr/bin/python3
Python 3.7.3 (default, Mar 27 2019, 13:41:07)
[GCC 8.3.1 20190223 (Red Hat 8.3.1-2)] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import numpy
>>> from mpi4py import MPI
```

3.1 — Create the Python program.

As mentioned above, we're going to use one of the demo programs provided in the mpi4py repo. However, because we'll be running it through the scheduler, we need to modify it to not require any user input. Create the file `/clusterfs/calc-pi/calculate.py`:

This program will split the work of computing our approximation of pi out to however many processes we provide it. Then, it will print the computed value of pi, as well as the error from the stored value of pi.

3.2 — Create and submit the job.

We can run our job using the scheduler. We will request some number of cores from the cluster, and SLURM will pre-configure the MPI environment with those cores. Then, we just run our Python program using OpenMPI. Let's create the submission

file `/clusterfs/calc-pi/sub_calc_pi.sh`:


```
#!/bin/bash
#SBATCH --ntasks=6

cd $SLURM_SUBMIT_DIR

mpiexec -n 6 /clusterfs/usr/bin/python3 calculate.py
```

Here, we use the `--ntasks` flag. Where the `--ntasks-per-node` flag requests some number of cores for each node, the `--ntasks` flag requests a specific number of cores *total*. Because we are using MPI, we can have cores across machines. Therefore, we can just request the number of cores that we want. In this case, we ask for 6 cores.

To run the actual program, we use `mpiexec` and tell it we have 6 cores. We tell OpenMPI to execute our Python program using the version of Python we installed.

> *Note that you can adjust the number of cores to be higher/lower as you want. Just make sure you change the `mpiexec -n ##` flag to match.*

Finally, we can run the job:

```
$ cd /clusterfs/calc-pi
$ sbatch sub_calc_pi.sh
Submitted batch job 1215
```

3.3 — Success!

The calculation should only take a couple seconds on the cluster. When the job completes (remember — you can monitor it with `squeue`), we should see some output in the `slurm-####.out` file:

```
$ cd /clusterfs/calc-pi
$ cat slurm-1215.out
pi is approximately 3.1418009868930934, error is 0.0002083333033003
```

You can tw reasing the numbe ing the `calculate.py` file:


```
if myrank == 0:
    _n = 20         # change this number to control the intervals
    .f
```

Slurm Raspberry Pi Cluster Glmdev Python

For example, here's the calculation run on 500 intervals:

```
pi is approximately 3.1415929869231265, error is 0.0000003333333334
```

Conclusion

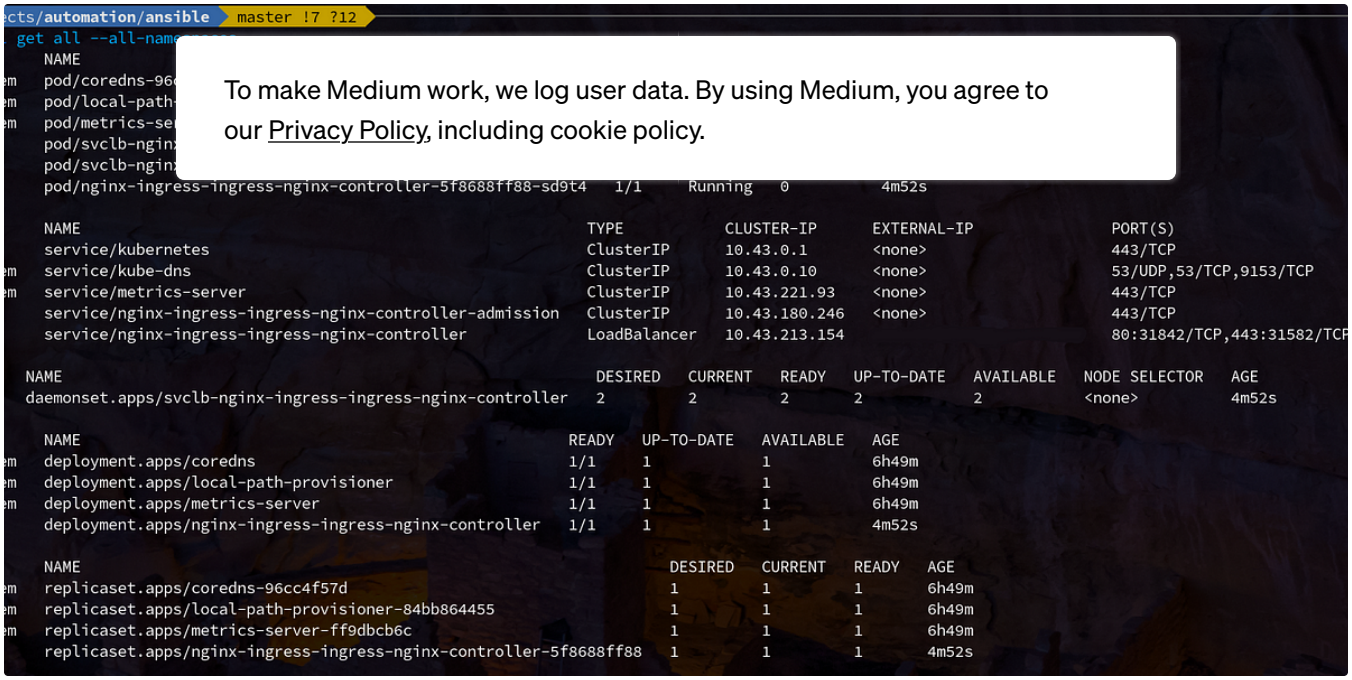
We w have a basically complete cluster. We can run jobs using the SLURM r; we discussed how to install software the lazy way and the better way; we i OpenMPI; and we ran some example programs that use it.

Written by Garrett Mills
350 Followers
Hi, there. I'm a software developer and speaker who likes to make things: https://garrettmills.dev/

Hopefully, your cluster is functional enough that you can add software and components to it to suit your projects. In the fourth and final installment of this series, we'll discuss a few maintenance niceties that are more related to managing installed software and users than the actual functionality of the cluster.

Happy Computing!

More from Garrett Mills
— Garrett Mills



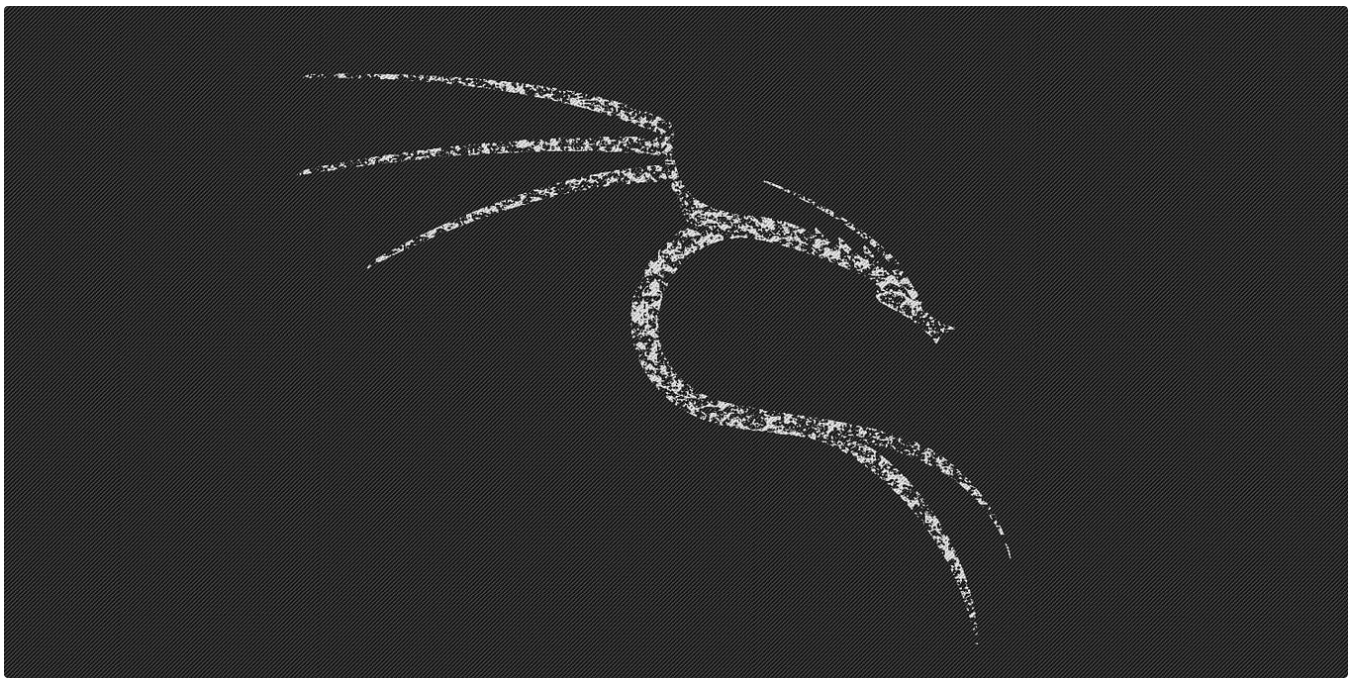
Garrett Mills in Better Programming

Rancher K3s: Kubernetes on Proxmox Containers

Using LXC containers and K3s to spin up a K8s cluster with NGINX Ingress Controller

9 min read · Apr 19, 2022

93 5



Garrett Mills

Installing

Linux. When [illegible] of code; however, today there are many modern Linux...

7 min read · Oct 4, 2015

98 4

```
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
glmdev*      up   infinite      3  alloc node[1-3]
glmdev@login1 ~/bobt> squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               995    glmdev sub-bobt   glmdev PD       0:00      1 (Resources)
               996    glmdev sub-bobt   glmdev PD       0:00      1 (Priority)
               997    glmdev sub-bobt   glmdev PD       0:00      1 (Priority)
               998    glmdev sub-bobt   glmdev PD       0:00      1 (Priority)
               983    glmdev sub-bobt   glmdev  R       0:09      1 node1
               984    glmdev sub-bobt   glmdev  R       0:09      1 node2
               985    glmdev sub-bobt   glmdev  R       0:09      1 node1
               986    glmdev sub-bobt   glmdev  R       0:09      1 node1
               987    glmdev sub-bobt   glmdev  R       0:08      1 node1
               988    glmdev sub-bobt   glmdev  R       0:08      1 node2
               989    glmdev sub-bobt   glmdev  R       0:08      1 node2
               990    glmdev sub-bobt   glmdev  R       0:08      1 node2
               991    glmdev sub-bobt   glmdev  R       0:08      1 node3
               992    glmdev sub-bobt   glmdev  R       0:08      1 node3
               993    glmdev sub-bobt   glmdev  R       0:08      1 node3
               994    glmdev sub-bobt   glmdev  R       0:08      1 node3
glmdev@login1 ~/bobt>
```

Garrett Mills

Building a Raspberry Pi Cluster

Part I—The Basics

12 min read · Nov 15, 2018

646 30

```
interfac
 /** @minLength 1 */
 username: string

 /** @minLength 1 */
 password: string

 rememberMe?: boolean
```

Garrett Mills in Better Programming

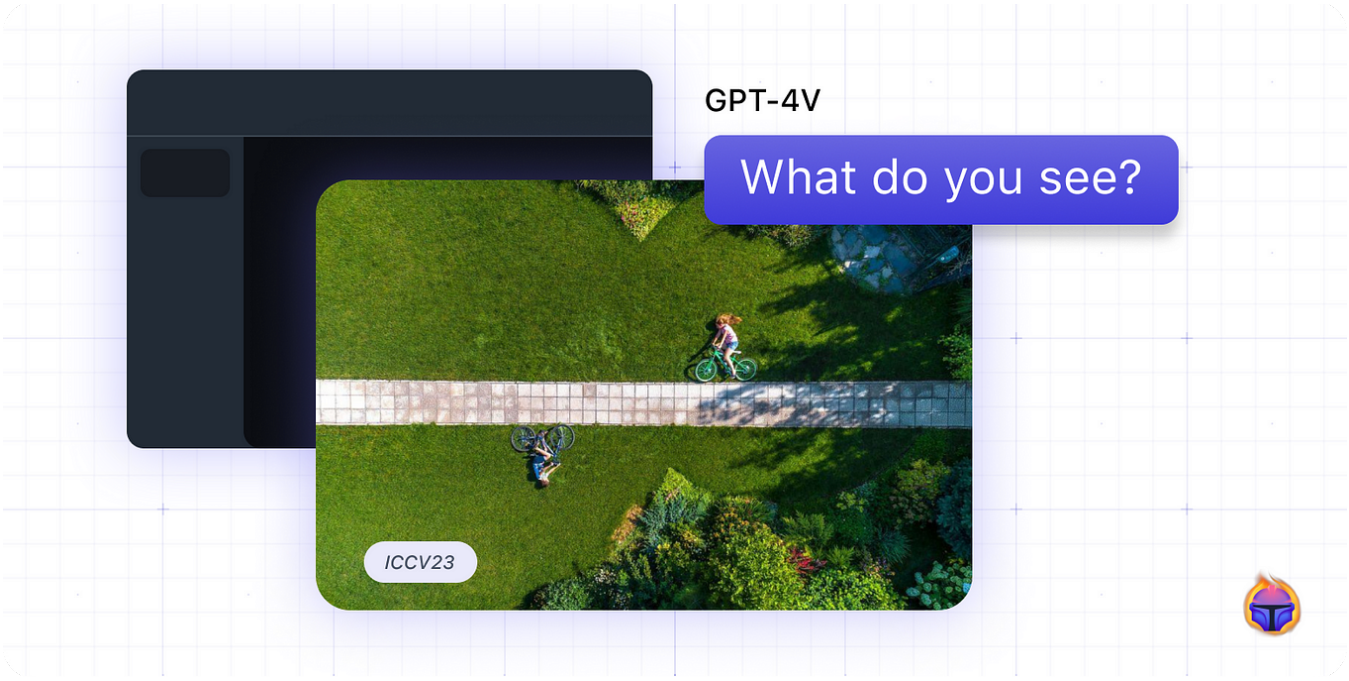
Runtime Data Validation from TypeScript Interfaces

How I (ab)used the TypeScript compiler to enable transparent runtime validation using Zod and TypeScript interfaces.

10 min read · Jan 14, 2022

68

Recommended from Medium



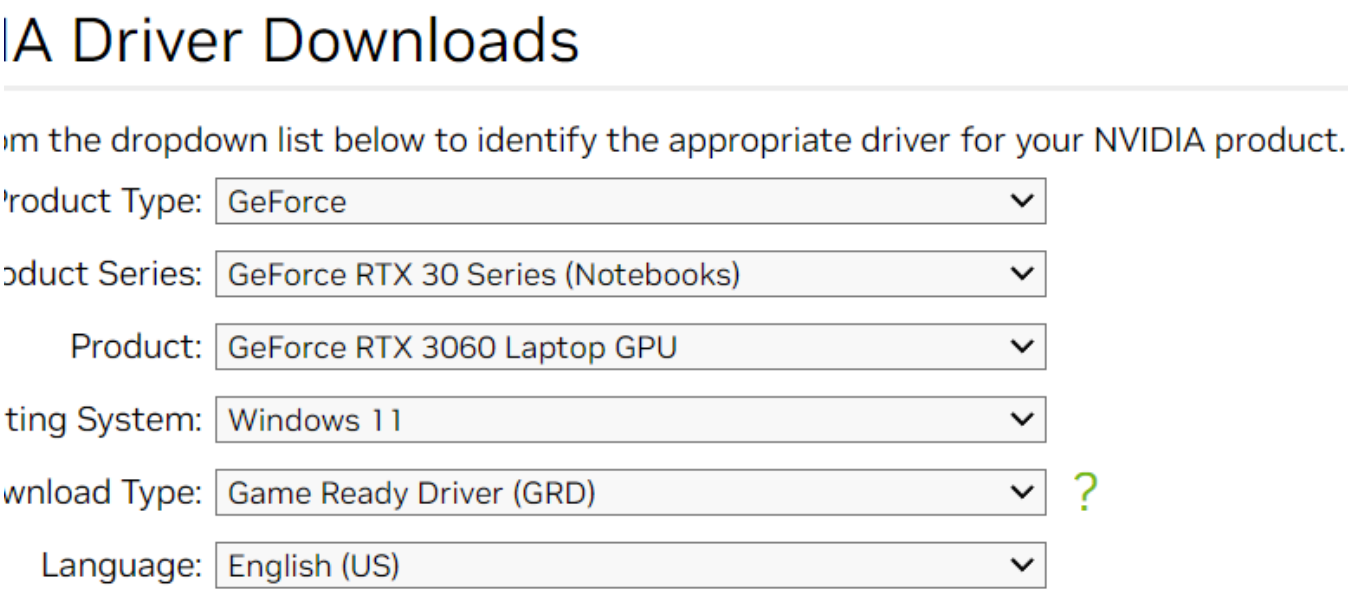
The Tenyks Blogger

Amid the

ICCV23 worl

5 min read · Oct 24

257 4



Abhishek Kumar Goyal

Setting up Tensorflow with CUDA for GPU on Windows 11

Machine Learning involves dealing with a humongous amount of data. Processing the data via various algorithms and applying different...

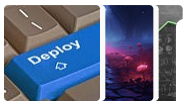
4 min read · Jul 9

130 2

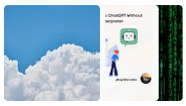
Lists

Coding & Development

11 stories · 282 saves

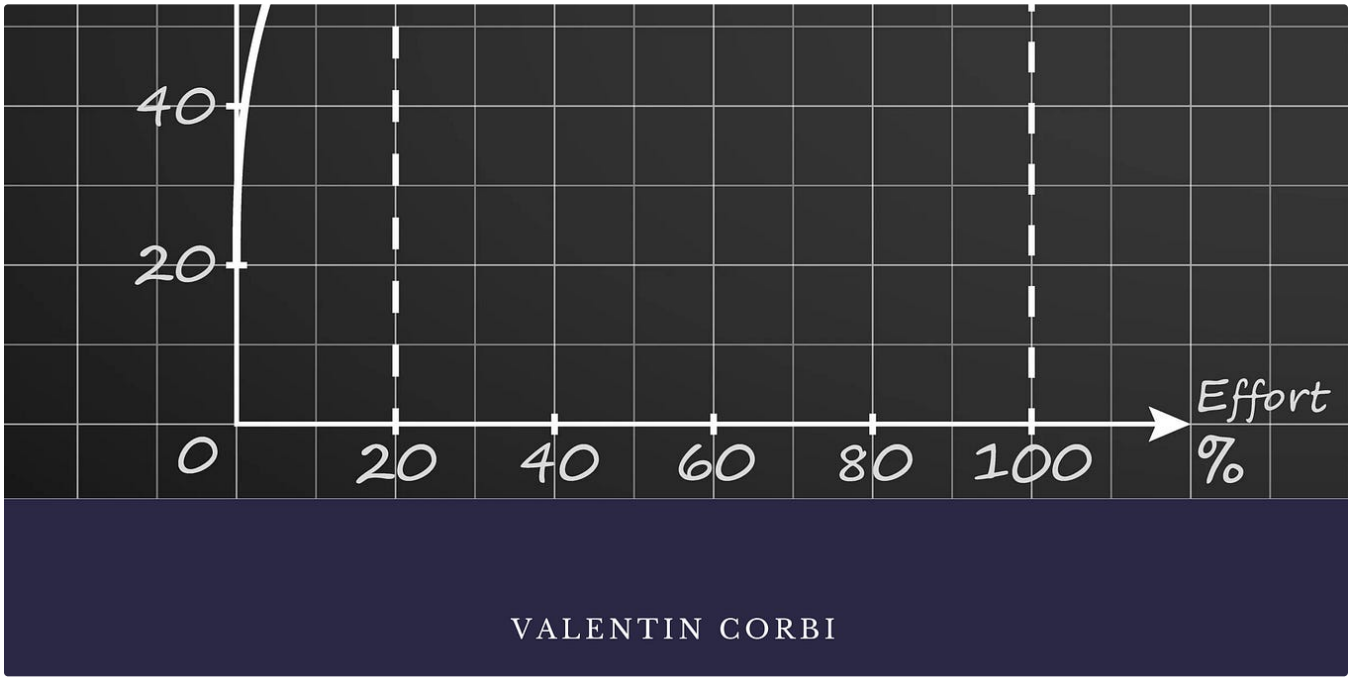


10 stories · 709 saves



ChatGPT

22 stories · 277 saves



The Pareto Investor

The Little Principle That Beats the Market

The First Book Revealing the Power of the Pareto Principle for Stocks and Guiding You to Invest Smarter, Not Harder!

· 36 min read · Oct 29

483 6

Avi Loeb

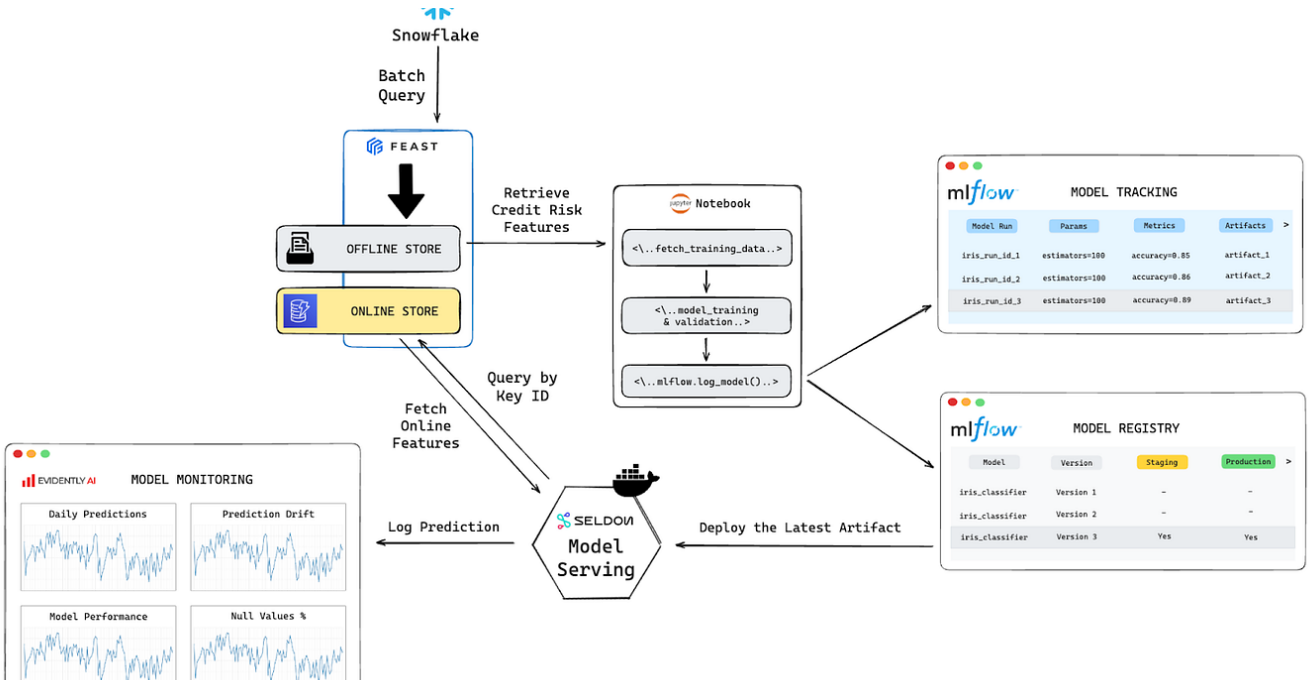
Are UAPs Relics from an Earlier Civilization on Earth?

The Paleozoic Era, between 541 and 252 million years ago, was a remarkable time for new life on Earth. It started with the Cambrian...

5 min read · 3 days ago


455


17



Qwak

Building a

MLOps Ope

11 min read · Oct 18

312 3

RoboFoundry

Building Cheapest ROS2 Robot using ESP32—Part 1—Hardware Build

I have always wanted to build a robot that was inexpensive and accessible to many people as an entry level robot to build that was running...

12 min read · Sep 22

16 2

See more recommendations