Building a Raspberry Pi Cluster

Part II — Some Simple Jobs

Garrett Mills · Follow

8 min read · Apr 1, 2019

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

Now that we have our cluster up and running, we can start running jobs on it. While the specific applications of your cluster are up to you, in the rest of this series I will be looking at how to set up a few different pieces of software, as well as how to use the actual cluster scheduler, Slurm.

In this part, we will dive into some Slurm basics, set up some software on our cluster the easy way, and create some example jobs that run many, many individual tasks making use of the scheduler.

In the next part, we'll look closer at how to install software the harder (better) way, how to set up Open MPI, and create some sample jobs that run just a few tasks across several nodes on the cluster.

1. Slurm Basics

As discussed before, Slurm is a piece of software called a scheduler. This allows you to submit jobs that request a specific amount of resources like CPU cores, memory,

or whole compute nodes. The scheduler will run each job as the resources become available. T l it'll figure it ou

1.a. Basic Slurm Commands

Slurm provides several useful command line tools that we'll use to interface with the cluster. Log into your master/login node that we set up last time:

```
ssh pi@node01
```

The first command we'll look at is `sinfo`. This is pretty straight forward, it just provides information about the cluster:

```
$ sinfo
PARTITION   AVAIL   TIMELIMIT   NODES   STATE   NODELIST
glmdev*     up      infinite    3       mix     node[1-3]
```

Here, we have the name of the partition, whether it can be used, the default time limit, the number of nodes and their states. The state "mix" occurs when a node has a job running on it, but it still has some available resources. (Such as when only 1 core is used.)

The `srun` command is awesome. It is used to directly run a command on however many nodes/cores you want. Let's test it out:

```
$ srun --nodes=3 hostname
node1
node2
node3
```

Here, we ran the `hostname` command on 3 nodes. This is different than running it on 3 cores, which may all be on the same node:

```
$ srun
node1
node1
node1
```

Here, `ntasks` refers to the number of processes. This is effectively the number of cores on which the command should be run. These are not necessarily on different machines. Slurm just grabs the next available cores.

We can also combine the two:

```
$ srun --nodes=2 --ntasks-per-node=3 hostname
node1
node2
node2
node1
node2
node1
```

This runs the command on 2 nodes and launches 3 tasks per node, effectively 6 tasks.

squeue — view scheduled jobs

When you start running longer and longer jobs, it is useful to check their status. To do this, run the `squeue` command. By default, it displays all jobs submitted by all users, and their states:

```
$ squeue
JOBID PARTITION  NAME     USER ST    TIME  NODES NODELIST(REASON)
  609 glmdev     24.sub.s pi   R     10:16     1 node2
```

Most of this info is pretty self-explanatory. The only thing I'll note is the `ST` column, which is the state of the job. `R` means that the job is running. Here's the full list of state codes.

scancel — c

Once a job mand:

```
$ scancel 609
```

(where `609` is the `JOBID` that you want to cancel) Note that you can only cancel jobs started by your user.

sbatch — schedule a batch script

`sbatch` is really the meat & potatoes of the Slurm scheduler. It's what we use most often when we want to schedule a job to run on the cluster. This command takes a number of flags and configuration, as well as a shell file. That shell file is then executed once and whatever requested resources (nodes/cores/etc) are made available to it. Let's create a basic job as an example.

The Batch File

Our job begins with the definition of a batch file. This batch file is usually a bash script that runs our job, however it looks a bit different. We'll create the file `/clusterfs/helloworld.sh`:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --partition=<partition name>

cd $SLURM_SUBMIT_DIR

echo "Hello, World!" > helloworld.txt
```

The file begins with a shebang. This is required, as it tells Slurm how to execute your job. This is followed by a number of flags that take the following form:

```
#SBATCH <flag>
```

These flags nand. These near nain difference: jobs aren't automatically re-launched on each specified node/core.

Rather, each job is run on the first core of the first node allocated it, but the job is given access to the other nodes it has requested. More on that later.

The `cd $SLURM_SUBMIT_DIR` guarantees that our job is running in whatever directory it was submitted from. In our case, this is `/clusterfs`.

Now, we can tell Slurm to schedule and run our job:

```
$ sbatch ./helloworld.sh
Submitted batch job 639
```

Since our job is very simple, it should be done basically immediately. If everything has gone according to plan, we should see the `/clusterfs/helloworld.txt` file that we created.

Output

You'll notice that the job doesn't output anything to the shell, which makes sense. If you had a job running for 4 hours, it's not very useful to have to have a terminal open the whole time to get output. Instead, Slurm outputs standard error and standard out to a file in the format `slurm-XXX.out` where `XXX` is the Job's ID number.

2. Our First Project

For our first project on the cluster, we're going to do some statistics! Data processing is a big part of what HPC clusters are used for. So, we're going to build a simple R program that generates some random values following a normal distribution, then creates a histogram and graph of those values and outputs them to an image file. Then, we're going to create a script to generate 50 of them using the scheduler.

Goal: Output 50 randomly-generated normal distribution graphs & histograms to a folder, as images using R.

2.a. Set-Up

Before we (ıfamiliar, R is a programming language for statistical programming. This makes it very well-suited for our task.

Now, there are several ways to install software on a cluster. Chiefly, the good way and the lazy way. We'll cover a better way in the next part, so for now, we'll use the lazy way. That is, we'll install R from the repos on each node.

However, we are *not* going to do them one-by-one. We have a shiny new scheduler, remember? So, we're going to cheat and use `srun`:

```
$ sudo su -
# srun --nodes=3 apt install r-base -y
```

This will run the `apt install r-base -y` command on all of the nodes in the cluster (change the `3` to match your setup). This will probably take a while, but when it completes, you should be able to use R on any of the nodes:

```
pi@node2 ~$ R --version
R version 3.3.3 (2017-03-06) -- "Another Canoe"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: arm-unknown-linux-gnueabihf (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License versions 2 or 3.
For more information about these matters see
http://www.gnu.org/licenses/.
```

2.b. The Theory

We need to run a large number of relatively small jobs. So, what we will do is create a script that, when executed, runs the `sbatch` command to schedule the same job over and over again.

We will use script to run, and te once for each number in the array, and the script can access its index during each job. This is how we will generate 50 random normal curves.

2.c. The R Program

Before we can schedule our program, we need to write a quick R script to generate the normal data-sets. So, we'll create the file `/clusterfs/normal/generate.R`:

```
arg = commandArgs(TRUE)

samples = rep(NA, 100000)
for ( i in 1:100000 ){ samples[i] = mean(rexp(40, 0.2)) }

jpeg(paste('plots/', arg, '.jpg', sep=""))
hist(samples, main="", prob=T, color="darkred")
lines(density(samples), col="darkblue", lwd=3)
dev.off()
```

Okay, a lot to unpack here:

- `arg = commandArgs(TRUE)` grabs the command line arguments passed to R when this script is run. This will be the job ID number.
- `samples = rep(NA,100000)` replicates `NA` 100000 times. This effectively creates an empty array with 100000 slots.
- `for( i in 1:100000 ){...}` iterate over the numbers 1–100000. This will be used to generate our random values.
- `samples[i] = mean(rexp(40, 0.2))` randomly generates 40 values following an exponential distribution with a range of 0.2. Then, find the mean of those values and store that mean in the samples array. This is our random dataset.
- `jpeg(paste('plots/', arg, '.jpg', sep=""))` open a new JPEG image to hold our graph. The name will be `plots/XX.jpg` where `XX` is the Job's ID number.

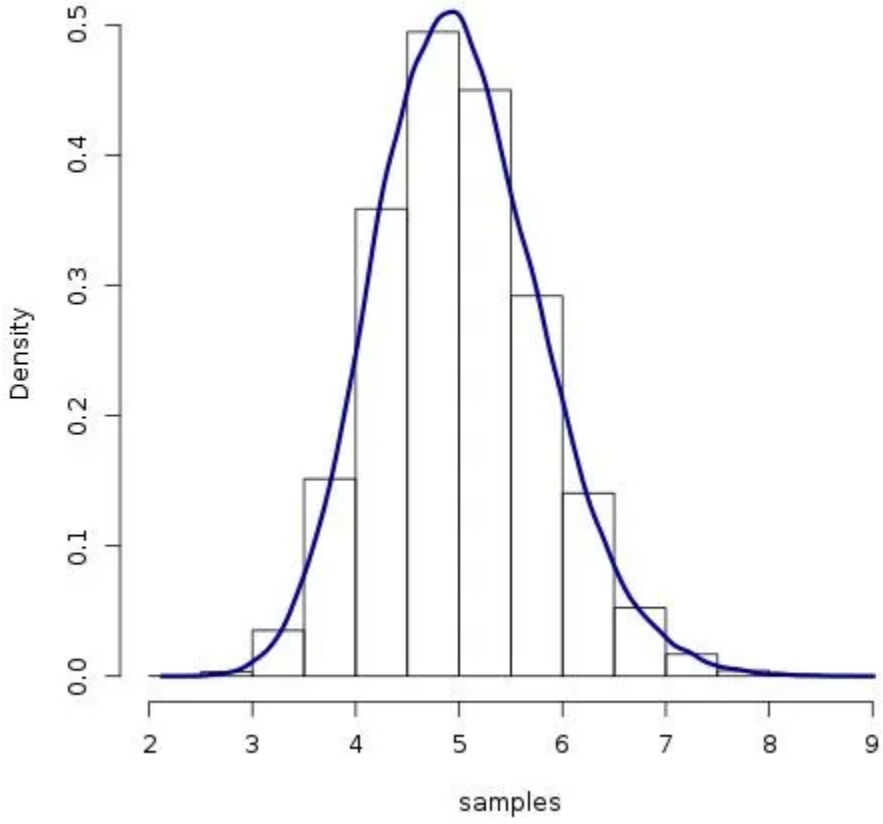
- `hist(samples, main="", prob=T, color="darkred")` display a histogram of our randon

- `lines(density(samples), col="darkblue", lwd=3)` plot the line of the value densities over the histogram.
- `dev.off()` close the JPEG file

As a test, you can run this program once as a test:

```
$ mkdir plots
$ R --vanilla -f generate.R --args "plot1"
...R output...
```

Now, in the plots folder, we should have the file `plot1.jpg` that looks something like so:



2.d. The Sub

Now that w ı our jobs.

Create the file `/clusterfs/normal/submit.sh`.

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --partition=<partition name>

cd $SLURM_SUBMIT_DIR
mkdir plots

R --vanilla -f generate.R --args "plot$SLURM_ARRAY_TASK_ID"
```

Here, we tell Slurm to run the job on 1 node, with 1 core on whatever partition you specified. Then, we change directories to the `/clusterfs/normal` folder where we will submit the job from.

Then we run the R program. This command looks very similar to the command we used to test the program but with one change. The name of the plot file is set to `plot$SLURM_ARRAY_TASK_ID`. This will name the image file after whatever index of the array we tell Slurm to run our job against. For example, when this job is run for the 23rd time, it will output the file: `plots/plot23.jpg`.

2.e. Run the Job!

We now have everything we need to run our job. From the login node, you can run the job like so:

```
$ cd /clusterfs/normal
$ sbatch --array=[1-50] submit.sh
Submitted batch job 910
```

Now, if we run `squeue`, we should see something like so:

When the jobs complete, the `/clusterfs/normal/plots` folder should have 50 of our randomly generated histograms.

Hpc Slurm Raspberry Pi Glmdev R

Going Forward

In the next part in this series, we will look at installing software the better way, as well as setting up a message-passing-interface for multi-node programs.

Part III is available here.

— Garrett

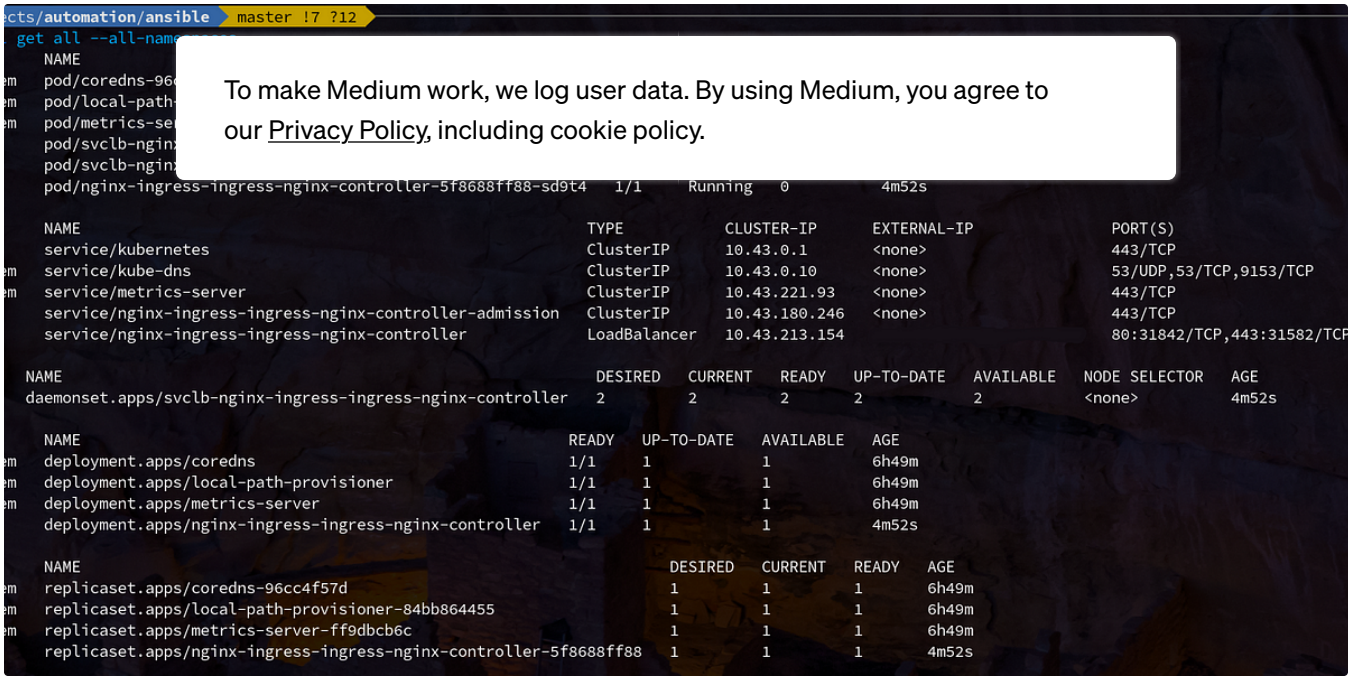
Follow

Written by Garrett Mills

350 Followers

Hi, there. I'm a software developer and speaker who likes to make things: https://garrettmills.dev/

More from Garrett Mills



Garrett Mills in Better Programming

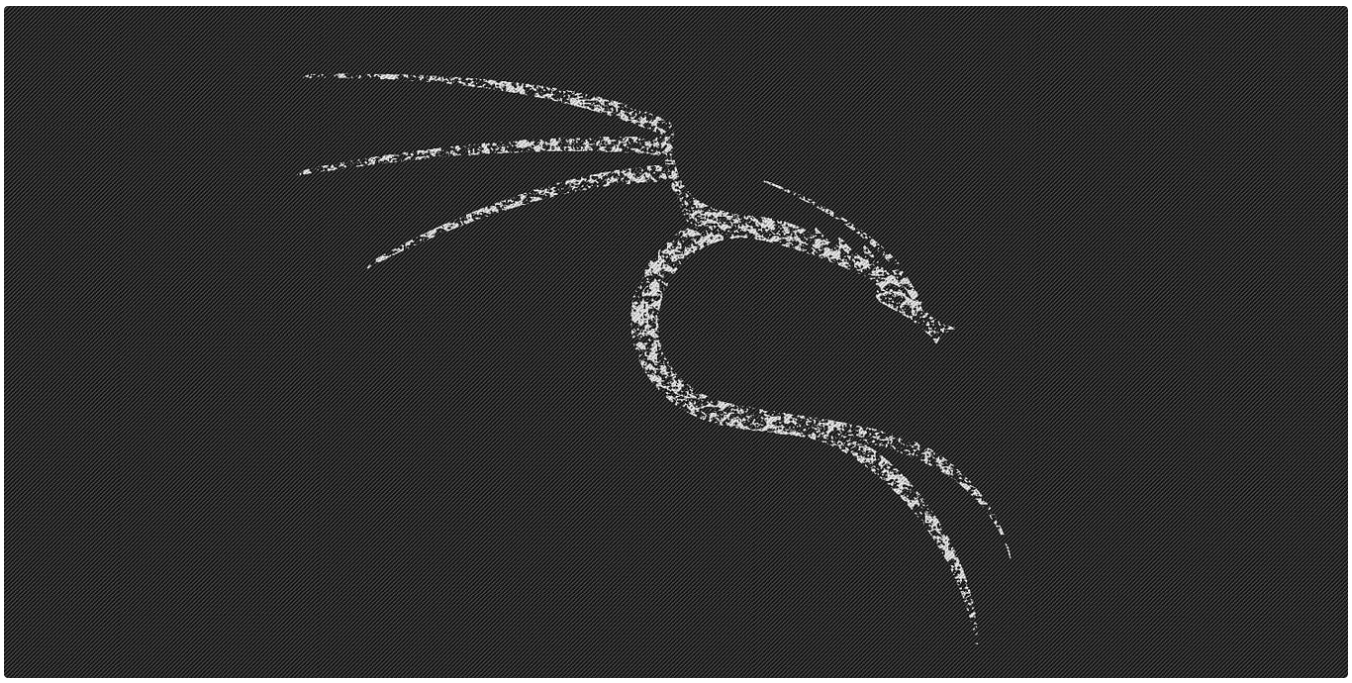
Rancher K3s: Kubernetes on Proxmox Containers

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

9 min read · Apr 19, 2022


93


5



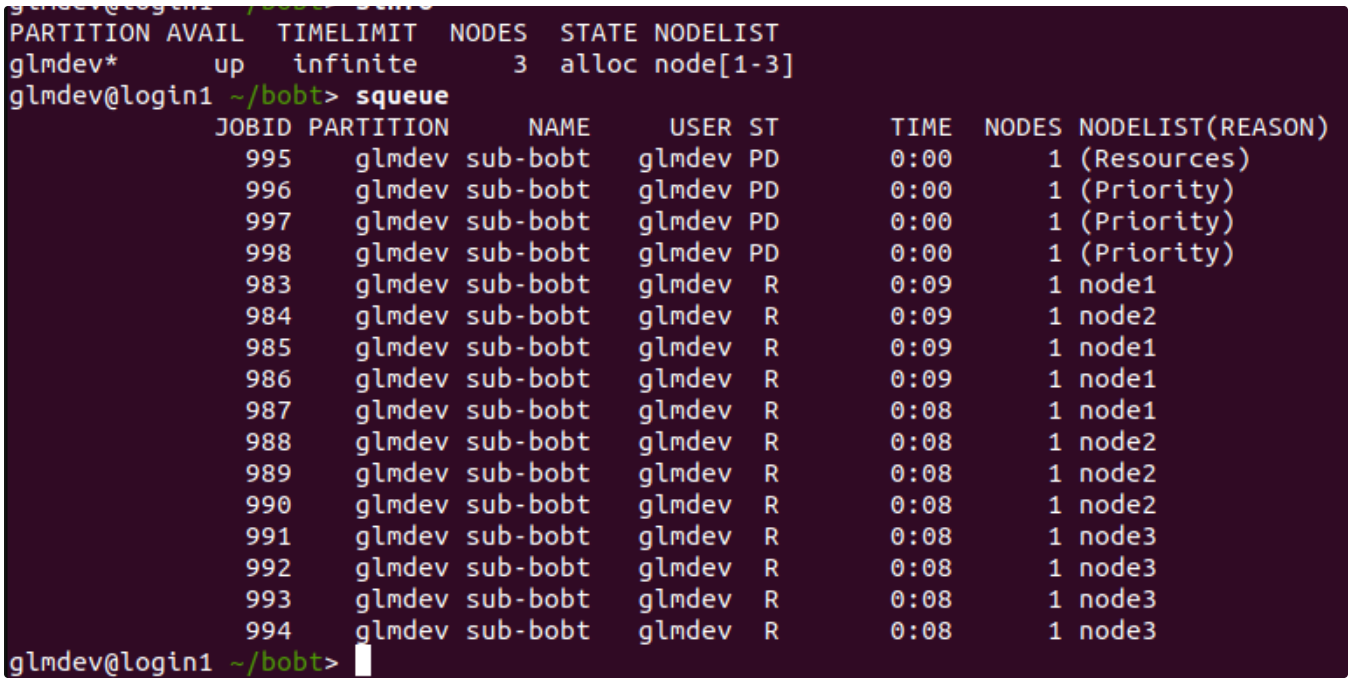
Garrett Mills

Installing

Linux. When ...s of code; however, today there are many modern Linux...

7 min read · Oct 4, 2015

98 4



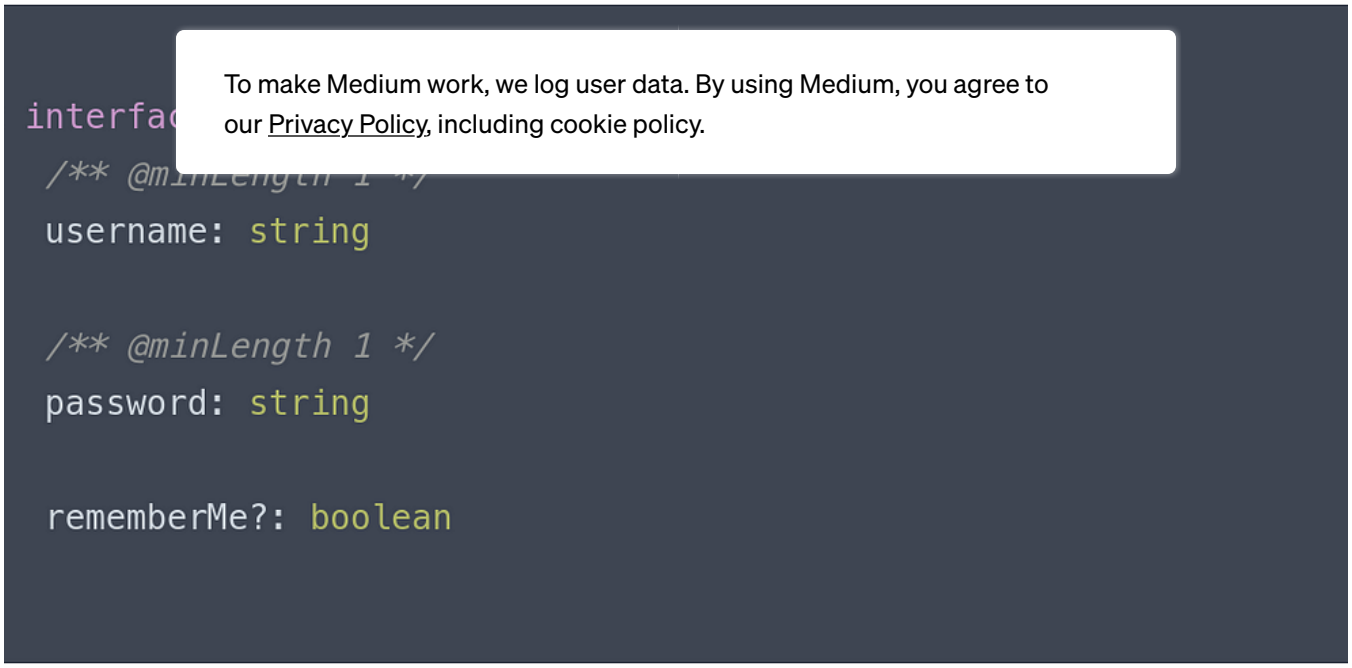
Garrett Mills

Building a Raspberry Pi Cluster

Part I—The Basics

12 min read · Nov 15, 2018

646 30



Garrett Mills in Better Programming

Runtime Data Validation from TypeScript Interfaces

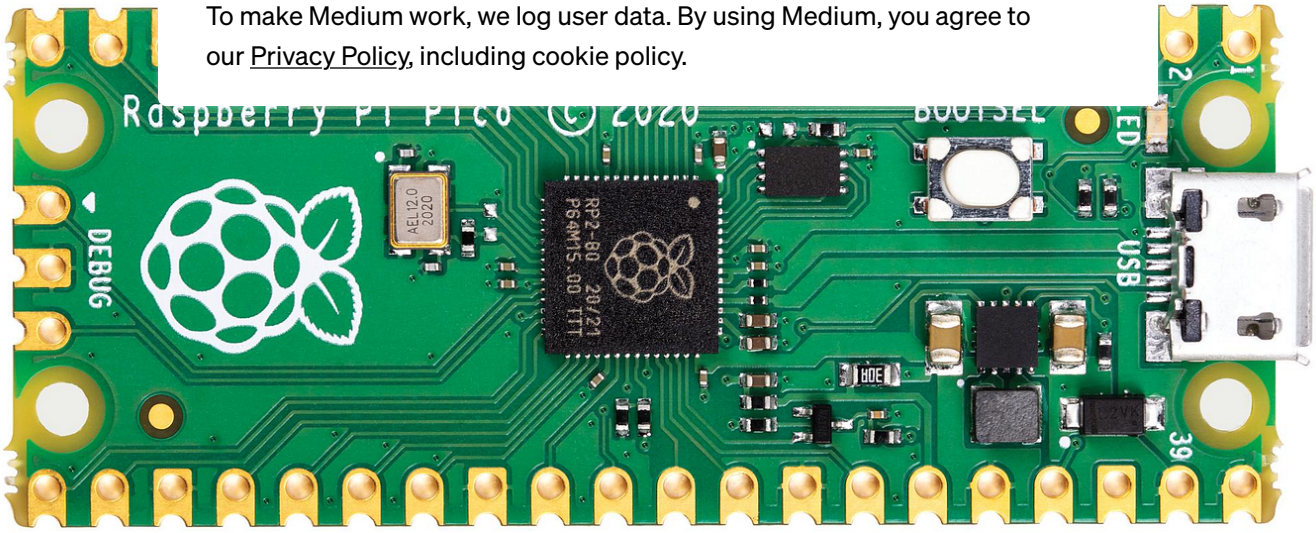
How I (ab)used the TypeScript compiler to enable transparent runtime validation using Zod and TypeScript interfaces.

10 min read · Jan 14, 2022


68

Recommended from Medium

See all from Garrett Mills



Christian Baghai in Dev Genius

Raspberry Pi Pico: A Comprehensive Guide

Introduction

4 min read · May 25

Isaac Saul

A personal

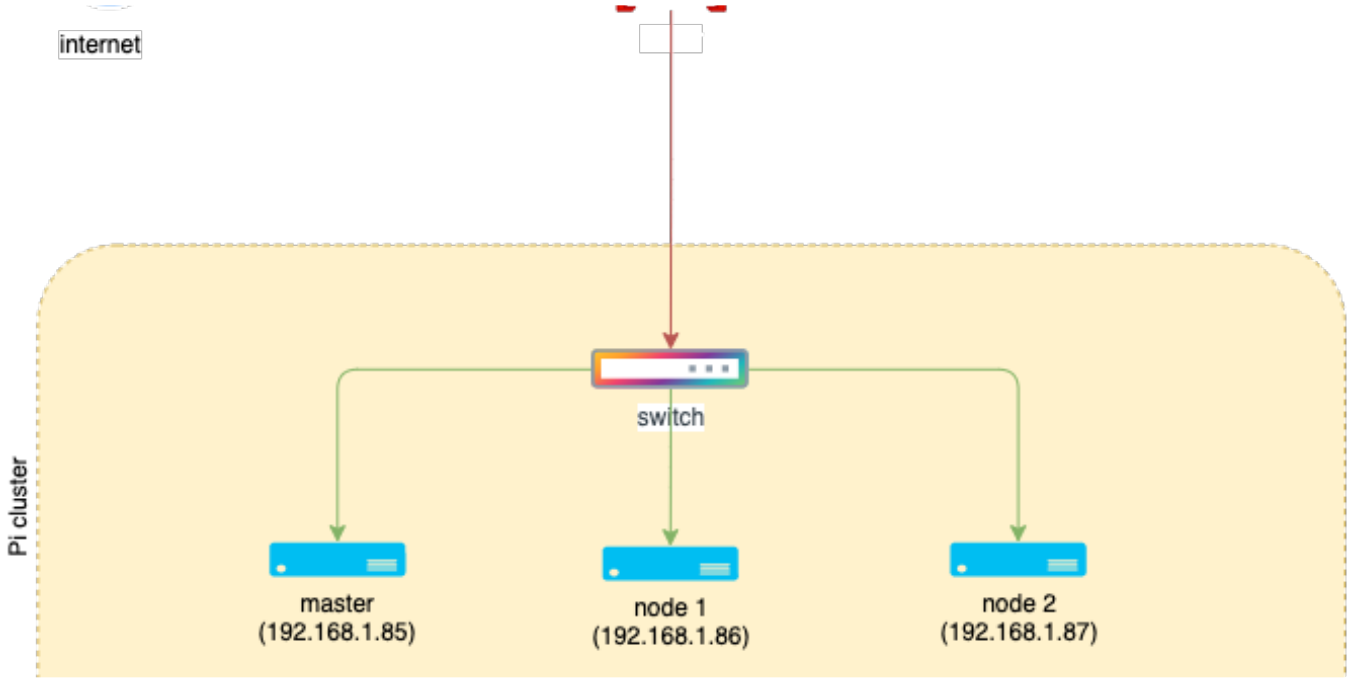
To understand this trial, we must understand the thousand year history that led us here

11 min read · Oct 12

Lists

Staff Picks

510 stories · 461 saves



drunkcodinq.net

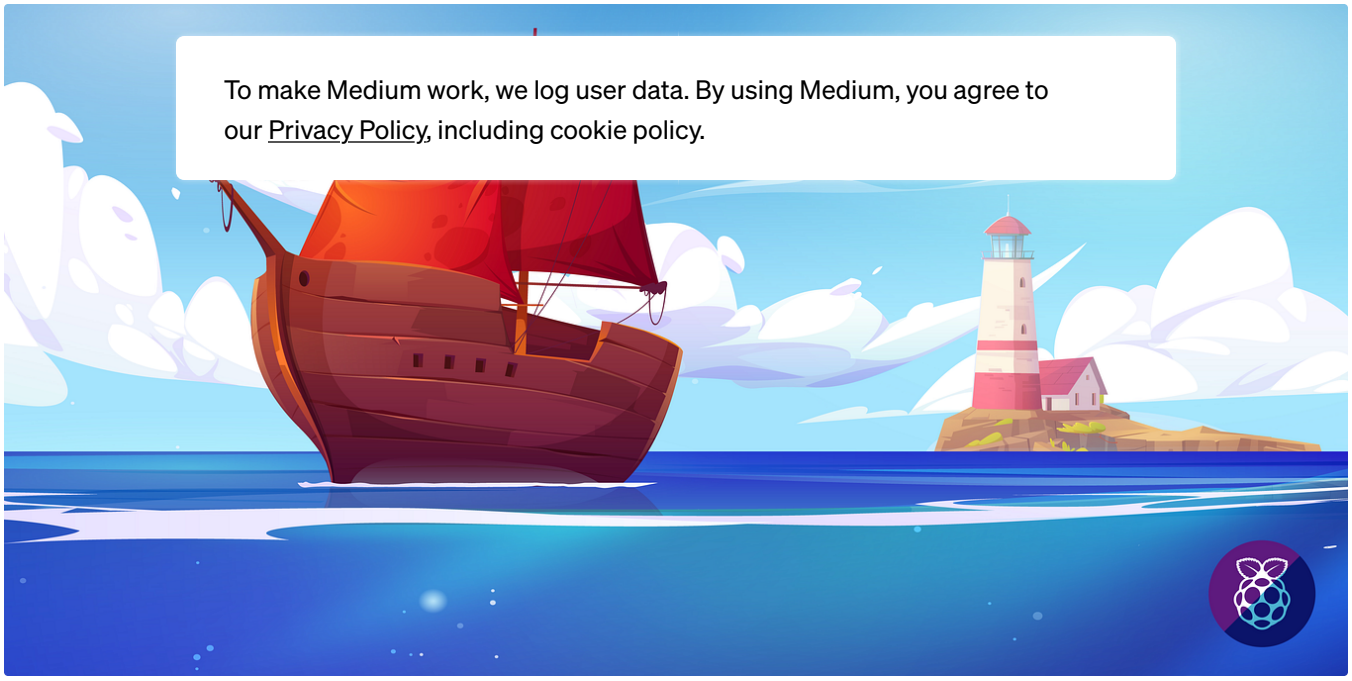
Life-is-short--so--enjoy-it

Raspberry Pi 5: Picked one from Local Computer Store

Finally, I got a Raspberry Pi 5 and a Active Cooler.

4 min read · 6 days ago

24 2



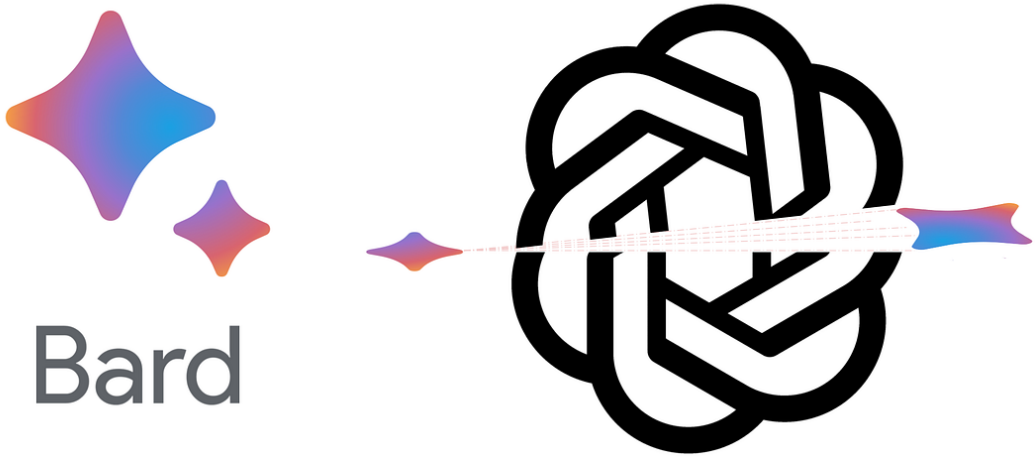
Alexander Sniffin

A Guide to Building a Kubernetes Cluster with Raspberry Pi's

A few years ago, I set up a Kubernetes Cluster on Raspberry Pi's. At the time, the ARM architecture of Raspberry Pi's posed some...

11 min read · Jul 5

71 3



AL Anany

The Chat atGPT.

It never happ

· 6 min read · Sep 1

19.3K 592