ept when it isn't

om image to upload on DigitalOcean), I've been bothered by one thing: the default installation size is large. To give you an idea, this simple system (using flakes):

```
.nix")
s.nix")
```

eadless!

blog post to be very different than what it became, but you can't win everything in life. There's a bit of pain ahead.

n their documentation otherwise). After spending some time managing NixOS servers, I really can't see myself going back to other systems unless required by some external f

the microVMs themselves. Currently, the system on the microVMs is taking ~210MB (including kernel) of disk space, but it's based on Alpine. The worker machines are alread

You can push an entirely new system configuration without the server changing its behaviour, and then almost atomically switch the server to the new configuration. You can ea
uration under a VM so you can locally test things if you wish to.

inimum software required for things to work, which would be an amazing help to lock the system down, prevent any escalations in case some piece of software was broken int

not extend this to the OS running on the microVMs and keep things really lean? This would be super helpful to cut boot times as much as possible, short of using a unikernel.

ate lean images by default, so a couple days ago I started looking into this to see if I could fix things, or at least significantly improve them.

purposes stated here. I don't really care about having some runtime-flexible server OS which lets me install packages and configure things ad-hoc, I want a thin, locked-down
ainers without containers. I comment on this at the end.)

ies and their sizes

powerful tools, but they're severely underdocumented, sometimes functionality is hidden by their naming, and/or some tools have some really specific assumptions, which mak

nown than some other more "obscure" tools). More specifically, `nix-store --query --tree` will give you a tree of packages[2], starting from a package you specify, and show yo

```
xos-24.05.20240323.44d0940


.38-44
c-13.2.0-libgcc
idn2-2.3.7
-libunistring-1.1
wdnf-libunistring-1.1 [...]
-libidn2-2.3.7 [...]
```

the size that a specific package takes on disk. It's slightly more complicated than this, but for our purposes it will be enough to understand how much disk is used.

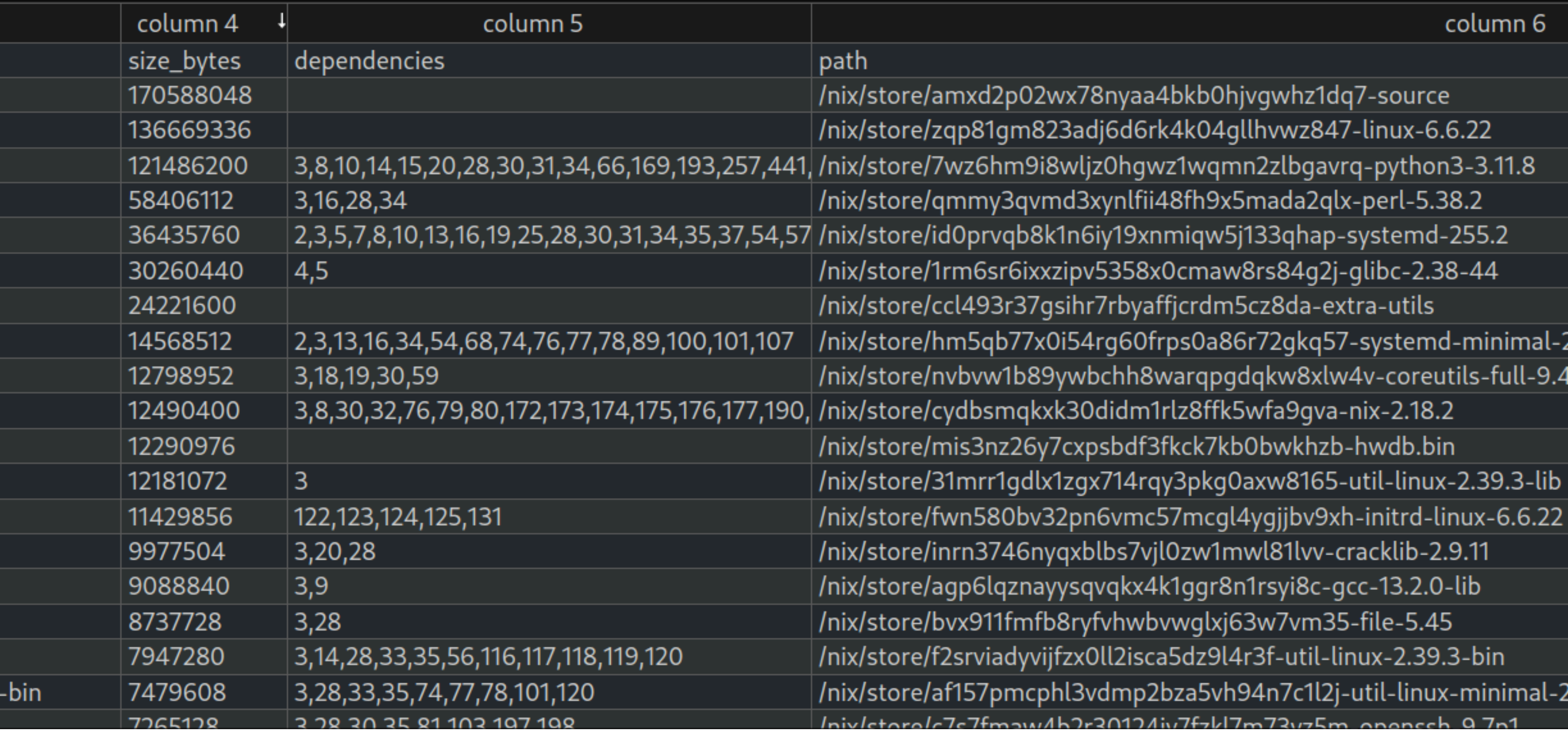
ol ways. Two of my favorites are nix-tree and nix-visualize. However, ideally I wanted an interactive graph so I could see each node in the graph by their size on disk, and insp
and the node sizes weren't based on disk usage, so I decided to write my own.

nviz file, with node sizes based on disk usage. Coupled with vscode-interactive-graphviz, I felt like I had a good approach to interactively working with the graph, but the visua
to be the one to position elements. In the end, I gave up on that idea and decided to just generate a CSV, which worked way better than I expected. No wonder we still use sp

em from this post is here.

headless NixOS system

encies, let's look at the minimal, headless system I mentioned in the beginning of the post. The one that takes ~900MB.



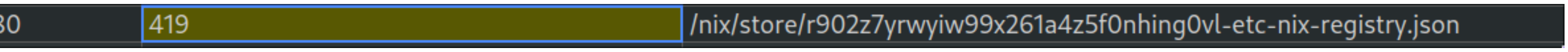
	column 4	column 5	column 6
	size_bytes	dependencies	path
	170588048		/nix/store/amxd2p02wx78nyaa4bkb0hjvgwhz1dq7-source
	136669336		/nix/store/zqp81gm823adj6d6rk4k04gllhvwz847-linux-6.6.22
	121486200	3,8,10,14,15,20,28,30,31,34,66,169,193,257,441,	/nix/store/7wz6hm9i8wljz0hgwz1wqmn2zlbgavrq-python3-3.11.8
	58406112	3,16,28,34	/nix/store/qmmy3qvmd3xynlfii48fh9x5mada2qlx-perl-5.38.2
	36435760	2,3,5,7,8,10,13,16,19,25,28,30,31,34,35,37,54,57	/nix/store/id0prvqb8k1n6iy19xnmiqw5j133qhap-systemd-255.2
	30260440	4,5	/nix/store/1rm6sr6ixxzipv5358x0cmaw8rs84g2j-glibc-2.38-44
	24221600		/nix/store/ccl493r37gsihr7rbyaffjcrdm5cz8da-extra-utils
	14568512	2,3,13,16,34,54,68,74,76,77,78,89,100,101,107	/nix/store/hm5qb77x0i54rg60frps0a86r72gkq57-systemd-minimal-2
	12798952	3,18,19,30,59	/nix/store/nvbvw1b89ywbchh8warqpgdqkw8xlw4v-coreutils-full-9.4
	12490400	3,8,30,32,76,79,80,172,173,174,175,176,177,190,	/nix/store/cydbsmqkxk30didm1rlz8ffk5wfa9gva-nix-2.18.2
	12290976		/nix/store/mis3nz26y7cxpsbdf3fkck7kb0bwkhzb-hwdb.bin
	12181072	3	/nix/store/31mrr1gdlx1zgx714rqy3pkg0axw8165-util-linux-2.39.3-lib
	11429856	122,123,124,125,131	/nix/store/fwn580bv32pn6vmc57mcgl4ygjjbv9xh-initrd-linux-6.6.22
	9977504	3,20,28	/nix/store/inrn3746nyqxblbs7vjl0zw1mwl81lvv-cracklib-2.9.11
	9088840	3,9	/nix/store/agp6lqznayysqvqkx4k1ggr8n1rsyi8c-gcc-13.2.0-lib
	8737728	3,28	/nix/store/bvx911fmfb8ryfvhwbvwglxj63w7vm35-file-5.45
	7947280	3,14,28,33,35,56,116,117,118,119,120	/nix/store/f2srviadyvijfzx0ll2isca5dz9l4r3f-util-linux-2.39.3-bin
-bin	7479608	3,28,33,35,74,77,78,101,120	/nix/store/af157pmcphl3vdmp2bza5vh94n7c1l2j-util-linux-minimal-2
	7265128	3,28,30,35,81,103,197,198	/nix/store/c7s7fmaw4b2r30124iv7fzkl7m73vz5m-openssh-9.7p1

e Edit csv extension for VSCode

me items in this CSV[3]. It starts “easy” and gets progressively more complicated. Feel free to skim and skip any part if you don’t feel like it.

uick look into what the heck could be taking 170MB of disk space shows it’s actually a complete copy of Nixpkgs!

e that generates the CSV and the graphviz files) shows that it’s only used by this other package:



30	419	/nix/store/r902z7yrwyiw99x261a4z5f0nhing0vl-etc-nix-registry.json

n a link to the “source” package. A search through Nixpkgs shows the file coming from here, the actual content of `registry` coming from here, and the `source` attribute being se

function from Nixpkgs’s `flake.nix`, which means by default I get this extra 170MB in the system. I *think* it would’ve been easy to just undo what Nixpkgs’s `flake.nix` is doing, I

pace as well (for example, `aws-sdk-cpp-1.11.207` eats another 5.7MB by itself, and is **only** used by Nix).

systems. I definitely don’t need it in a microVM, but I also don’t need it in my servers, because I’m building their configurations in an external machine and deploying the built b

ction)

s Perl.

shame, why waste so much disk space like this!), and Perl comes in through a bunch of perl-envs (search for `perl-5.38.2-env` in the CSV and you’ll see them). Those perl-en

```
3jvjgn2mq3106-nixos-system-nixos-24.05.20240323.44d0940
xos-24.05.20240323.44d0940/dry-activate:23:/nix/store/d3qxgm4ffhi2ixx3n9clwqlr6z21dd8i-perl-5.38.2-env/bin/perl \
xos-24.05.20240323.44d0940/activate:43:/nix/store/d3qxgm4ffhi2ixx3n9clwqlr6z21dd8i-perl-5.38.2-env/bin/perl \
xos-24.05.20240323.44d0940/activate:63:/nix/store/zkmm5iha0rsm4ypwfc67byq52gz0jb8b-perl-5.38.2-env/bin/perl /nix/store/rg5rf512szdxmnj9qal3wfdnpfsx38qi-setup
xos-24.05.20240323.44d0940/bin/switch-to-configuration:1:#! /nix/store/8mlvyl3sab5hxpxz2naz5g2sfd42a40q-perl-5.38.2-env/bin/perl
```

`-activate`, `activate`, and `bin/switch-to-configuration` scripts. `dry-activate` only needs Perl to run the `update-users-groups.pl` script, while the `activate` script runs the san

ocumentation bits, but the perl man pages are the only thing that still get included in the system because of the perl-envs!

d to at least take a look. After all, judging only by the naming, `update-users-groups.pl` doesn’t seem like the kind of thing I need - I don’t expect my servers to create any extra

es, this was just my thinking from reading its name)

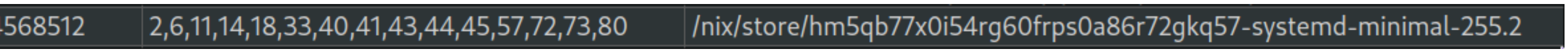
ow it was being added to the system. It was through this search that I stumbled upon a Nixpkgs tracking issue called Perlless Activation - Tracking Issue.

xOS system for slightly different reasons, and they did a lot of work to get rid of it! Luckily for me, I could piggyback off their work and include the following module in my syste

```
x")
```

s now gone as well!

that I think could be removed, but since it's an integral part of the system, let's overlook it for now. Going through the list of packages, what's this in 5th place?



```
-568512    2,6,11,14,18,33,40,41,43,44,45,57,72,73,80    /nix/store/hm5qb77x0i54rg60frps0a86r72gkq57-systemd-minimal-255.2
```

d-minimal! A look through which packages use systemd-minimal show that only dbus uses it. It comes from here.

ndencies or to keep the size of dependencies smaller, it introduces variants of packages/functions that have reduced functionality. If you contribute to Nixpkgs, chances are th size (a common example is using `stdenvNoCC` instead of `stdenv`). systemd-minimal probably exists to avoid certain circular dependencies, but I'm not sure. It's defined here.

have the full systemd in our system anyway. There is no easy way to override the package used by the NixOS module that brings in dbus, so we'll have to add a Nixpkgs ove

appers (~30MB reduction)

the list of heaviest packages, I saw an "hwdb.bin" package which seems linked to udev. I don't know about udev too much, but it feels like it's only needed for scenarios that w

he system, I have a feeling that a workaround could be hard-coded and wouldn't require udev anyway. I'd gladly go into that rabbit hole, but (spoiler alert) you'll see that I gave

s also enabled by default. Similar reasoning to udev, I don't think I'd need lvm for these servers, so I disabled it.

ome more packages in the list. At that point, it became clear that I'd have to butcher a LOT of NixOS config to remove many packages in there. I made a decision to continue of some of these packages. To get to a barebones system, I'd need to remove a lot of them.

ard-coded by default (and changing those gets complicated quickly). I saw they're used by some security wrappers, which also set other security wrappers for mount, umount nat dynamically sets some capabilities and permissions, and then executes any other binary with the elevated bits.

ary which receives an argument with the binary to execute with elevated permissions, I'd rather have X wrapper binaries with hardcoded paths and no parametrisation of any k

he proper permissions through systemd unit configs.

gle it off, so one way to get rid of it completely is to add it to the `disabledModules` attribute. This requires me to provide dummy options that were provided by the security wrap nything because they're not enabled). Some of these modules set additional wrappers, so the dummy options are needed to make the module system happy.

nt or fuse (because those are hardcoded in the security wrappers module), but I also think that most scripts that use those are being run directly as root, so I'm not sure.

s useless without its security wrapper.

util-linux-minimal, respectively. Well, this seems similar to that systemd-minimal thing from a while ago!

t-pstore” shell script.

mess in the config so far, it wouldn’t even look that bad anymore). But we can at least try to make them use util-linux instead of util-linux-minimal, right?

packages.nix. We’ll need an overlay, but trying to change `fusePackages` to use the normal util-linux will hit an infinite recursion error, so I’ll start by overlaying `fuse3`:

```
ckage (nixpkgs.outPath + "/pkgs/os-specific/linux/fuse") { })).fuse_3;
```

nite recursion error is back. *Sigh*. Whatever.

libs sneaking in there. It’s being used by a bunch of other packages, and it’s equally difficult to add more overlays to get rid of it. Infinite recursions all around.

es I made of things to look into that I haven’t yet (the list is right there in the next section), think about how much worse it’ll get by trying to fix all of this, and give up.

136MB. I know I can get it down to ~50MB easily (the kernel used by default on NixOS has a lot of modules and extra things that a server doesn’t need), so I left that for later

o a different configuration at runtime, because the script that does this is written in Perl. This isn’t an issue for most of the servers in my scenario. MicroVMs don’t need that, a

out how much work it would be to build a replacement. This is something that needs to happen anyway at some point, and would benefit the NixOS community at large.

onalisation content. Those end up taking some good space, so I made a note to look at how to simplify this and get rid of most of the locales and files I wouldn’t need.

s only used for netcat, but I don’t really need it in the servers. In fact, NixOS includes a lot of utilities by default (and marks them as required, making it super annoying to remo

as bashrc) could be removed. This would also remove some packages that they use, such as bash-completion, which won’t ever be needed in the servers.

ly that the scripts and things that use them only really need a few binaries from each one. Perhaps an overlay that filters the binaries only to the list that are used would help fr

dentical files and hard-linking them. This could be helpful in some cases, but might not be possible in others, depending on how a server is imaged, or how new configuration share exactly the same file).

a lot of the defaults and modules present in NixOS reflect that. NixOS can still be used as a server OS, but it requires a very different set of configurations, and it still ends up r

my existing servers and make them leaner, which will already be useful, because it cuts ~300MB of stuff I don’t need. I got some experience and figured out some tools to help

o mold NixOS into the shape I envisioned just isn’t the way to go, but I also don’t like the other option if I want to stick with it, which is creating a “fork” of NixOS that is **very** opi

e similar to building containers with the bare minimum required for the software in the container to run. I think this is a worthy endeavour. I think we have all the tools in regular land, thus removing quite a bunch of complexity from the systems we build.

defined by the Nix manual, because for most people this is an easier way to reason about store objects. ↩

d it, but you won’t be able to inspect the store paths unless you happen to build the same configuration with the same Nixpkgs version. ↩