Bridging the stepping stones:

using pieces of NixOS without full commitment

Michael Raskin · 7c6f434c

(TU Munich)

17.10.2020

Do you have to use... (and do you care)

- ...Nix package manager

  yes, and doubling software installation space use is minor

- ...Nixpkgs

  most likely, and it's at most a few GiBs of clones/`stdenv`

- ...NixOS

  maybe, and it's most a change to every habit about managing OS

Do you have to use... (and do you care)

- ...Nix package manager

  yes, and doubling software installation space use is minor

- ...Nixpkgs

  most likely, and it's at most a few GiBs of clones/`stdenv`

- ...NixOS

  maybe, and it's most a change to every habit about managing OS

Do you have to use... (and do you care)

- ...Nix package manager

  yes, and doubling software installation space use is minor

- ...Nixpkgs

  most likely, and it's at most a few GiBs of clones/`stdenv`

- ...NixOS

  maybe, and it's most a change to every habit about managing OS

NixOS: enticing features

NixOS has a ton of nice sides

`https://nixos.org/` list of NixOS features boils down to:

- your system is Nix package

  ...and you have Nix around

- unless early boot broken, boots to consistent state
- declarative config

NixOS: enticing features

NixOS has a ton of nice sides

`https://nixos.org/` list of NixOS features boils down to:

- your system is Nix package

  ...and you have Nix around

- unless early boot broken, boots to consistent state
- declarative config

NixOS: enticing features

NixOS has a ton of nice sides

`https://nixos.org/` list of NixOS features boils down to:

- your system is Nix package

  ...and you have Nix around

- unless early boot broken, boots to consistent state
- declarative config

NixOS: enticing features

NixOS has a ton of nice sides

`https://nixos.org/` list of NixOS features boils down to:

- your system is Nix package

  ...and you have Nix around

- unless early boot broken, boots to consistent state
- declarative config

NixOS: control your expectations

So I can safely experiment with...

- init systems?
- OS kernels?
- service overrides like with packages in Nixpkgs?

...and no more complicated interactions between choices?

NixOS: control your expectations

So I can safely experiment with...

- init systems?
- OS kernels?
- service overrides like with packages in Nixpkgs?

...and no more complicated interactions between choices?

NixOS: control your expectations

So I can safely experiment with...

- init systems?
- OS kernels?
- service overrides like with packages in Nixpkgs?

...and no more complicated interactions between choices?

...not exactly

What NixOS hardcodes

NixOS hardcodes:

- Linux
- systemd
- configuration via module system

  global namespace and lots of moving parts

  ...modules are less overridable than packages
- how your setup core is described

What NixOS hardcodes

NixOS hardcodes:

- Linux
- systemd
- configuration via module system

  global namespace and lots of moving parts

  ...modules are less overridable than packages
- how your setup core is described

What NixOS hardcodes

NixOS hardcodes:

- Linux
- systemd
- configuration via module system

  global namespace and lots of moving parts

  ...modules are less overridable than packages
- how your setup core is described

What NixOS hardcodes

NixOS hardcodes:

- Linux
- systemd
- configuration via module system

  global namespace and lots of moving parts

  ...modules are less overridable than packages
- how your setup core is described

What NixOS hardcoding implies

Trapped inside are less opinionated things:

- daemon config generator
- daemon start flag knowledge

...duplicated by

- `nix-darwin`
- `home-manager`
- `nix-processmgmt`

What NixOS hardcoding implies

Trapped inside are less opinionated things:

- daemon config generator
- daemon start flag knowledge

...duplicated by

- `nix-darwin`
- `home-manager`
- `nix-processmgmt`

Is code really trapped?

Is code really trapped?

`#invoke`

What do you want to invoke?

Law of headlines

You hear the answer:

Is code really trapped?

`#invoke`

What do you want to invoke?

Law of headlines

You hear the answer:

«... not exactly»

Is code really trapped?

Not completely

Strategy:

- evaluate NixOS with configuration talking only of the service
- grab the parts you care about

  contents of files for `etc`

  NixOS systemd unit generation can export `Exec*` to script — thanks for that!

That's what I use on my system for CUPS and Xorg

Is code really trapped?

Not completely

Strategy:

- evaluate NixOS with configuration talking only of the service
- grab the parts you care about

  contents of files for `etc`

  NixOS systemd unit generation can export `Exec*` to script — thanks for that!

That's what I use on my system for CUPS and Xorg

Is code really trapped?

Not completely

Strategy:

- evaluate NixOS with configuration talking only of the service
- grab the parts you care about

  contents of files for `etc`

  NixOS systemd unit generation can export `Exec*` to script — thanks for that!

That's what I use on my system for CUPS and Xorg

Is code really trapped?

Not completely

Strategy:

- evaluate NixOS with configuration talking only of the service
- grab the parts you care about

  contents of files for `etc`

  NixOS systemd unit generation can export `Exec*` to script — thanks for that!

That's what I use on my system for CUPS and Xorg

Current functionality in LangOS

- `serviceScript`: service name, config → script
- `etcSelectTarget`: filename, config → store path
- `etcSelectPrefix`: path prefix, config → attrSet of store paths

`https://github.com/7c6f434c/lang-os/blob/master/use-from-nixos.nix`

Current functionality in LangOS

- `serviceScript`: service name, config → script
- `etcSelectTarget`: filename, config → store path
- `etcSelectPrefix`: path prefix, config → attrSet of store paths

`https://github.com/7c6f434c/lang-os/blob/master/use-from-nixos.nix`

Implications — and limitations

NixOS real value is a large database of config generators

Many services already reusable (if you know how)

But:

- pay attention: some arguments live in different branches
- some services too complicated for working runner script

  might want to get the parts and assemble carefully
- some configs do not got to `/etc`,

  do not get option name,

  only referenced inside service

Implications — and limitations

NixOS real value is a large database of config generators

Many services already reusable (if you know how)

But:

- pay attention: some arguments live in different branches
- some services too complicated for working runner script

  might want to get the parts and assemble carefully
- some configs do not got to `/etc`,

  do not get option name,

  only referenced inside service

Implications — and limitations

NixOS real value is a large database of config generators

Many services already reusable (if you know how)

But:

- pay attention: some arguments live in different branches
- some services too complicated for working runner script

  might want to get the parts and assemble carefully
- some configs do not got to `/etc`,

  do not get option name,

  only referenced inside service

Implications — and limitations

NixOS real value is a large database of config generators

Many services already reusable (if you know how)

But:

- pay attention: some arguments live in different branches
- some services too complicated for working runner script

  might want to get the parts and assemble carefully
- some configs do not got to `/etc`,

  do not get option name,

  only referenced inside service

System structuring

Use of services without NixOS bootscripts feasible...

...but what if not the module system?

- Mimic Nixpkgs overlays
- `makeExtensible` small core system
- Core system adapts by reading from `self`
- Override in overlays whatever you configure
- Few `let` instances, everything inspectable and modifiable!

System structuring

Use of services without NixOS bootscripts feasible...

...but what if not the module system?

- Mimic Nixpkgs overlays
- `makeExtensible` small core system
- Core system adapts by reading from `self`
- Override in overlays whatever you configure
- Few `let` instances, everything inspectable and modifiable!

System structuring

Use of services without NixOS bootscripts feasible...

...but what if not the module system?

- Mimic Nixpkgs overlays
- `makeExtensible` small core system
- Core system adapts by reading from `self`
- Override in overlays whatever you configure
- Few `let` instances, everything inspectable and modifiable!

System structuring

Use of services without NixOS bootscripts feasible...

...but what if not the module system?

- Mimic Nixpkgs overlays
- `makeExtensible` small core system
- Core system adapts by reading from `self`
- Override in overlays whatever you configure
- Few `let` instances, everything inspectable

  and modifiable!

System structuring

But what if not the module system?

...can we make it not matter?

- Services as packages with parameters
- Rich `passthru`
- Can request and inspect other service
- It's user's choice how to make sure things match
- Module system still the default

  as well-defined layer

System structuring

But what if not the module system?

...can we make it not matter?

- Services as packages with parameters
- Rich `passthru`
- Can request and inspect other service
- It's user's choice how to make sure things match
- Module system still the default

  as well-defined layer

System structuring

But what if not the module system?

...can we make it not matter?

- Services as packages with parameters
- Rich `passthru`
- Can request and inspect other service
- It's user's choice how to make sure things match
- Module system still the default

  as well-defined layer

System structuring

But what if not the module system?

...can we make it not matter?

- Services as packages with parameters
- Rich `passthru`
- Can request and inspect other service
- It's user's choice how to make sure things match
- Module system still the default

  as well-defined layer

NixOS and bootloader

- NixOS configures bootloader from system generations
- Assumes NixOS format of entry
- Needs knowledge of Xen etc.

...that's why I am too lazy to dual-boot with NixOS

NixOS and bootloader

- NixOS configures bootloader from system generations
- Assumes NixOS format of entry
- Needs knowledge of Xen etc.

...that's why I am too lazy to dual-boot with NixOS

NixOS and bootloader

- NixOS configures bootloader from system generations
- Assumes NixOS format of entry
- Needs knowledge of Xen etc.

...that's why I am too lazy to dual-boot with NixOS

NixOS and bootloader

- NixOS configures bootloader from system generations
- Assumes NixOS format of entry
- Needs knowledge of Xen etc.

...that's why I am too lazy to dual-boot with NixOS

Dream

- NixOS services as Nixpkgs-like packages
- Argument overrides and overlays for configuration
- Module system as *one of the ways* to connect things
- Multiple independent options for core, sharing the service DB
- NixOS bootloader generator collecting bootloader config snippets

  Each system instance provides snippets for supported loaders

  Fail unless forced if `booted-system` does not support new loader?
- Also, support for atomic `/etc` switch in NixOS

Dream

- NixOS services as Nixpkgs-like packages
- Argument overrides and overlays for configuration
- Module system as *one of the ways* to connect things
- Multiple independent options for core, sharing the service DB
- NixOS bootloader generator collecting bootloader config snippets

  Each system instance provides snippets for supported loaders

  Fail unless forced if `booted-system` does not support new loader?
- Also, support for atomic `/etc` switch in NixOS

Dream

- NixOS services as Nixpkgs-like packages
- Argument overrides and overlays for configuration
- Module system as *one of the ways* to connect things
- Multiple independent options for core, sharing the service DB
- NixOS bootloader generator collecting bootloader config snippets

  Each system instance provides snippets for supported loaders

  Fail unless forced if `booted-system` does not support new loader?
- Also, support for atomic `/etc` switch in NixOS

Dream

- NixOS services as Nixpkgs-like packages
- Argument overrides and overlays for configuration
- Module system as *one of the ways* to connect things
- Multiple independent options for core, sharing the service DB
- NixOS bootloader generator collecting bootloader config snippets

  Each system instance provides snippets for supported loaders

  Fail unless forced if `booted-system` does not support new loader?
- Also, support for atomic `/etc` switch in NixOS

Why did I bother?

- VTs not owned by systemd

  custom tricks around Xorg launch

  physical presence check for eg. power-off command

- Integrated `nsjail` wrappers

  Nobody runs browsers outside containers nowadays, right?

  Only hand-picked things have sound access

- My on-boot mounts are easier (to me) to describe in shell than in Nix
- Full versions of everything in `initramfs`
- Nice `strace`-able scripts for services right there

Why did I bother?

- VTs not owned by systemd

  custom tricks around Xorg launch

  physical presence check for eg. power-off command

- Integrated `nsjail` wrappers

  Nobody runs browsers outside containers nowadays, right?

  Only hand-picked things have sound access

- My on-boot mounts are easier (to me) to describe in shell than in Nix
- Full versions of everything in `initramfs`
- Nice `strace`-able scripts for services right there

Why did I bother?

- VTs not owned by systemd

  custom tricks around Xorg launch

  physical presence check for eg. power-off command

- Integrated `nsjail` wrappers

  Nobody runs browsers outside containers nowadays, right?

  Only hand-picked things have sound access

- My on-boot mounts are easier (to me) to describe in shell than in Nix
- Full versions of everything in `initramfs`
- Nice `strace`-able scripts for services right there

Why did I bother?

- VTs not owned by systemd

  custom tricks around Xorg launch

  physical presence check for eg. power-off command

- Integrated `nsjail` wrappers

  Nobody runs browsers outside containers nowadays, right?

  Only hand-picked things have sound access

- My on-boot mounts are easier (to me) to describe in shell than in Nix
- Full versions of everything in `initramfs`
- Nice `strace`-able scripts for services right there

Why did I bother?

- VTs not owned by systemd

  custom tricks around Xorg launch

  physical presence check for eg. power-off command

- Integrated `nsjail` wrappers

  Nobody runs browsers outside containers nowadays, right?

  Only hand-picked things have sound access

- My on-boot mounts are easier (to me) to describe in shell than in Nix
- Full versions of everything in `initramfs`
- Nice `strace`-able scripts for services right there

What you can take?

- Wrappers for NixOS piece extraction

  `https://github.com/7c6f434c/lang-os/blob/master/use-from-nixos.nix`

- Wrappers to run things in isolated DBus sessions

  `https://github.com/7c6f434c/lang-os/blob/master/bus-wrappers.nix`

- Firefox empty profile builder

  `https://github.com/7c6f434c/lang-os/blob/master/firefox-profile.nix`

- Wrapper to use TTF fonts in Linux console

  `https://github.com/7c6f434c/lang-os/blob/master/ttf-to-psf.nix`

What you can take?

- Wrappers for NixOS piece extraction

  `https://github.com/7c6f434c/lang-os/blob/master/use-from-nixos.nix`

- Wrappers to run things in isolated DBus sessions

  `https://github.com/7c6f434c/lang-os/blob/master/bus-wrappers.nix`

- Firefox empty profile builder

  `https://github.com/7c6f434c/lang-os/blob/master/firefox-profile.nix`

- Wrapper to use TTF fonts in Linux console

  `https://github.com/7c6f434c/lang-os/blob/master/ttf-to-psf.nix`

What you can take?

- Wrappers for NixOS piece extraction

  `https://github.com/7c6f434c/lang-os/blob/master/use-from-nixos.nix`

- Wrappers to run things in isolated DBus sessions

  `https://github.com/7c6f434c/lang-os/blob/master/bus-wrappers.nix`

- Firefox empty profile builder

  `https://github.com/7c6f434c/lang-os/blob/master/firefox-profile.nix`

- Wrapper to use TTF fonts in Linux console

  `https://github.com/7c6f434c/lang-os/blob/master/ttf-to-psf.nix`

What you can take?

- Wrappers for NixOS piece extraction

  `https://github.com/7c6f434c/lang-os/blob/master/use-from-nixos.nix`

- Wrappers to run things in isolated DBus sessions

  `https://github.com/7c6f434c/lang-os/blob/master/bus-wrappers.nix`

- Firefox empty profile builder

  `https://github.com/7c6f434c/lang-os/blob/master/firefox-profile.nix`

- Wrapper to use TTF fonts in Linux console

  `https://github.com/7c6f434c/lang-os/blob/master/ttf-to-psf.nix`

Thanks for you attention!

Questions?