Our first Unikernels in OCaml — *for fun and profit*!

Lambda Nantes - Workshop 4

April 2026

Lambda Nantes

- **One meetup per month** (Thanks Arnaud)
- We are **always** looking for speakers (for presentations or workshops)
- Despite what the name (*Lambda*) might suggest, **we're open to all kinds of topics** (with a particular interest in Applicative languages, software engineering and formal method).

Feel free to **join us** as a speaker (and obviously as a participant).

The Workshop

Hello! In this **interactive presentation** (*a workshop*), we'll get started with creating **unikernels** using the wonderful OCaml language.

In my opinion, this is a topic that's very easy to summarize:

The Workshop

Hello! In this **interactive presentation** (*a workshop*), we'll get started with creating **unikernels** using the wonderful OCaml language.

In my opinion, this is a topic that's very easy to summarize:

> *A unikernel is **a single-application VM** image where the application and **minimal operating system** are compiled into one optimized binary.*

The Workshop

Hello! In this **interactive presentation** (*a workshop*), we'll get started with creating **unikernels** using the wonderful OCaml language.

In my opinion, this is a topic that's very easy to summarize:

> *A unikernel is **a single-application VM** image where the application and **minimal operating system** are compiled into one optimized binary.*

But **very hard** summarize in a workshop (I was being very ambitious!)

Plan: a very quick, dirty and informal presentation

- What are *Unikernels*
  - Why are they not as good?
  - What are they good for?
  - Unikernels in OCaml
    - Mirage and Mkernel
    - Solo5
    - Actors in the world of Unikernels in OCaml

Plan: a very quick, dirty and informal presentation

- What are *Unikernels*
  - Why are they not as good?
  - What are they good for?
  - Unikernels in OCaml
    - Mirage and Mkernel
    - Solo5
    - Actors in the world of Unikernels in OCaml
- Our first Unikernel (*Quick Demonstration*)
  - Hello World (Unikernel that just print “Hello World”)
  - Hello World as a server (Unikernel that always respond “Hello World”)
  - A first webserver (using a Webframework)
  - A Webserver with HTML (crunching files)
  - Free Project (The fun part — and it might take a while)

Plan: a very quick, dirty and informal presentation

- What are *Unikernels*
  - Why are they not as good?
  - What are they good for?
  - Unikernels in OCaml
    - Mirage and Mkernel
    - Solo5
    - Actors in the world of Unikernels in OCaml
- Our first Unikernel (*Quick Demonstration*)
  - Hello World (Unikernel that just print "Hello World")
  - Hello World as a server (Unikernel that always respond "Hello World")
  - A first webserver (using a Webframework)
  - A Webserver with HTML (crunching files)
  - Free Project (The fun part — and it might take a while)

The first four unikernels will mainly be presentations (which you can follow interactively), and the last one will be the actual open-source project—and ***OCaml experts*** *can lend a hand!*

Why

Because I love promoting OCaml and...

Why

Because I love promoting OCaml and...

I'd really love for us to finally create a website for LambdaNantes, and I like to imagine it as **a galaxy of Unikernels surrounding a static site**, allowing us to develop libraries that are useful to the community (like `ocaml-biscuit`).

Why

Because I love promoting OCaml and...

I'd really love for us to finally create a website for LambdaNantes, and I like to imagine it as **a galaxy of Unikernels surrounding a static site**, allowing us to develop libraries that are useful to the community (like `ocaml-biscuit`).

- Easy to host/deploy
- Funny to maintain
- A beautful **over-engineered** approach!
- **We'll probably need to get organized** — if that sounds good to you — in a smart way :)
- A nice community project

So, what are Unikernels ?

So, what are Unikernels ?

*A unikernel is a **specialized**, **single-purpose machine image** that links application code with **only** the operating system components it actually needs.*

So, what are Unikernels ?

*A unikernel is a **specialized**, **single-purpose machine image** that links application code with **only** the operating system components it actually needs.*

*The result is a **single bootable image** that runs directly on a hypervisor, microVM, or bare metal target. There is no shell, no unnecessary drivers, and no multi-user support — only the application and the minimal runtime required to execute it.*

So, what are Unikernels ?

*A unikernel is a **specialized**, **single-purpose machine image** that links application code with **only** the operating system components it actually needs.*

*The result is a **single bootable image** that runs directly on a hypervisor, microVM, or bare metal target. There is no shell, no unnecessary drivers, and no multi-user support — only the application and the minimal runtime required to execute it.*

So yes, we're going to build an operating system for a single application

The main difference with containers

With a Unikernel, our application doesn't need a full operating system to run; in terms of execution, it might superficially resemble containerization?

The main difference with containers

With a Unikernel, our application doesn't need a full operating system to run; in terms of execution, it might superficially resemble containerization?

Container	Unikernel
Shares the host Linux kernel	Has its **own** minimal kernel image
Lightweight process isolation	Stronger isolation
	Even smaller runtime surface
Great tooling ecosystem	
isolated process	isolated single-purpose mini-OS

Why are they not as good?

As an user

- Harder debugging
- Limited Ecosystem (the clean slate approach imply a full rewriting of the world)
- Usually one service per image
- Rebuild required for changes

In term of perfomance

An unikernel will not outperform a native application in I/O.

- OCaml (GC) vs C
- Indirection in calls
  - A regular process on Linux **can make system calls** that interact with the kernel's network stack **directly**
  - **A unikernel cannot**: it runs inside a sandboxed environment and must go through two layers of indirection for every I/O operation. (that can be partially mitigated with things like `virtio`)

What are they good for

- Microservices (yeah, the constellation :D)
- Serverless platforms
- Network appliances
- Edge computing
- security (reducing the attack-surface)
- super fast scale-to-zero/startup systems

Think of unikernels as a way to build smaller, safer, and more composable services.

In term of perfomance and cost

unikernels **boot in milliseconds** because there is no operating system to initialize, and their images are only a few megabytes compared to hundreds for a typical container. The per-instance cost of running a unikernel is therefore very low.

So for microservices, it's really cool, cost-effective, highly secure, and very efficient. Plus, since the boot time is so short, **a good orchestration strategy makes the system highly resilient**!

And that lets you **use OCaml in unexpected places**:

- The space (SpaceOS)
- Docker For Mac
- QubeOS
- Some popular streaming services

And that lets you **use OCaml in unexpected places**:

- The space (SpaceOS)
- Docker For Mac
- QubeOS
- Some popular streaming services

And this requires a level of portability that benefits other contexts (such as a web browser), ***providing excellent generic libraries****.*

Unikernels in OCaml

MirageOS

> *A library/framework for building a clean-slate unikernel, originally developed at Xen (**state of the art and battle-tested**) with network stack, multiple filesystems, verified cryptography etc.*

These days, Mirage can be seen in two ways:

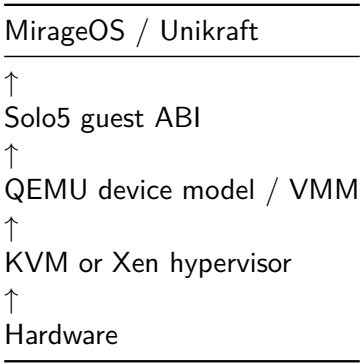
- A toolchain for building an application intended to be "run" by a unikernel
- A set of libraries properly packaged for portability (beyond the unikernel ecosystem)

An alternative of MirageOS is Unikraft (but nowaday, Unikraft **can be a backend for MirageOS**).

In practice

Xen or KVM (typically via a VMM, virtual machine monitor, such as QEMU) provide **the virtual machine boundary**, while **Solo5 provides the minimal guest ABI** (without needing to produces elf) that lets **MirageOS or Unikraft** run portably across hypervisors without depending on backend-specific interfaces.

Typical Unikernel App Tower



MirageOS and MKernel

MirageOS is also a **packaging discipline** that, for cross-compilation purposes, requires the organization and maintenance of a package cluster.

MirageOS and MKernel

MirageOS is also a **packaging discipline** that, for cross-compilation purposes, requires the organization and maintenance of a package cluster.

MKernel is a newer, **alternative toolchain** developed by the Robur cooperative (which also makes use of the MirageOS libraries) but which, in my opinion, is easier to get started with (and does not rely on excessive functorization) and enforce direct-style (vs Monad Based API).

It allows you to quickly run a unikernel via Solo5 without any complicated ceremony (even if the stack is pretty new and need some more polish).

To keep things simple in this workshop, we will use Mkernel. (*But I hope you'll have the chance to try out the Mirage toolchain if this workshop has convinced you.*)

Let's code

https://github.com/xvw/mkernel-empty-project

You can follow the instructions in the README while I show you our first Unikernels.

First Unikernel, Hello World

- dune/opam and vendor (**explain the need for vendor, for cross-compilation reasons instead of a strong packaging policy**)
- source code

Second Unikernel, Our first server

- RNG
- Mnet
- Miou

Third Unikernel, Let's see the web

- Vif and Vifu

Fourth Unikernel, let's crunch files

- Mcrunch, Filesystem Without Filesystem
- Helpers libraries (`uniko.util`)

Your unikernel

There are several references in the README — it's up to you!

A simple idea

A tiny URL shortener, which simply uses the shared parameter of `Vifu.run` to pass a mutable table that allows identifiers to be associated with URLs, with various possible enhancement levels:

- Handle error-induced collisions
- Improving ID generation (for example, if `xvw` already exists, it becomes `xvw-1`)
- A cool UI with crunch
- A directory of links (with TyXML)

Let's go !