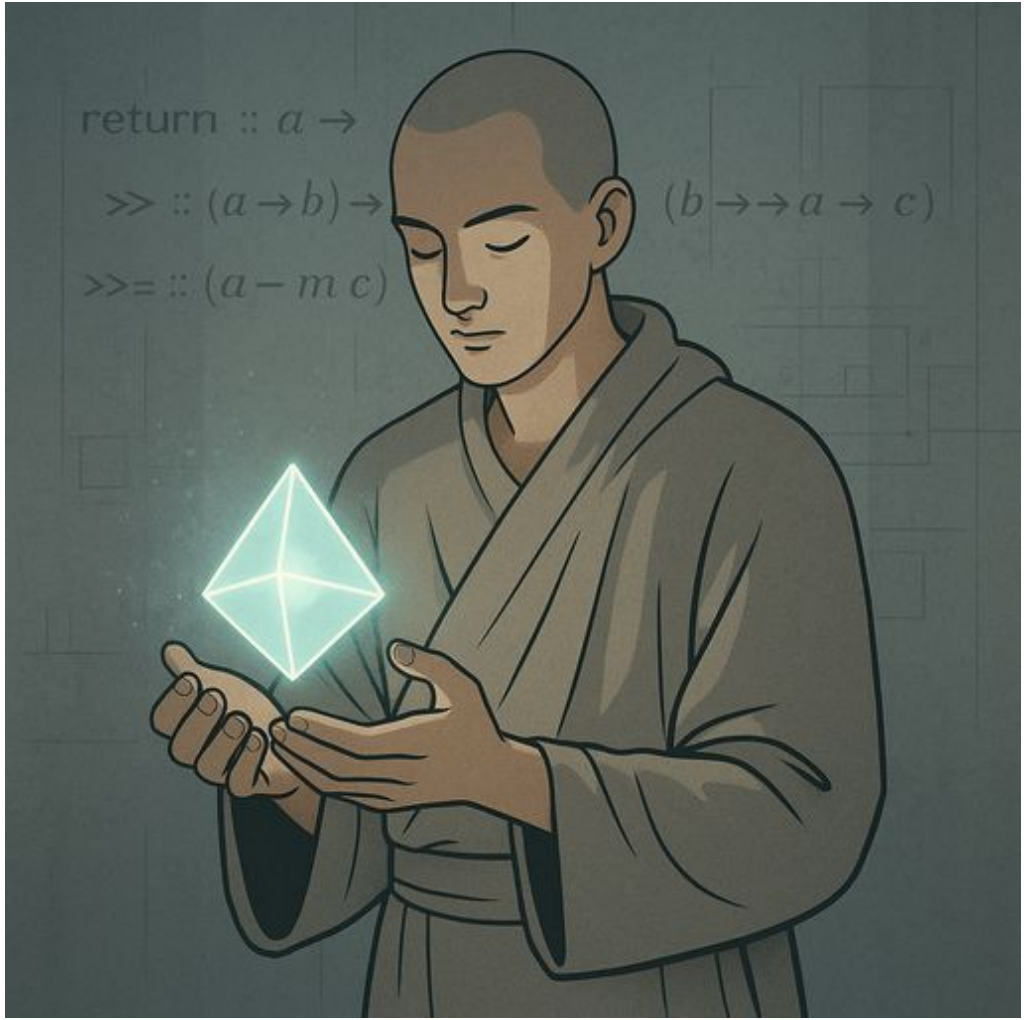

Always has been
Wait, it's all monads ?

Companion code and talk outline:
https://github.com/bassosimone/kleisli-compose

“A monad is a way of sequencing computations where each step may fail or produce effects, while keeping the plumbing out of your core logic.”

–ChatGPT 4o


return :: a →
>> :: (a → b) →
(b →→ a → c)
>>= :: (a − m c)

💎[x / y] := 💎[x] / 💎[y]

What's the value of 💎[x / y]?

How can be 💎[x / 0] OK?

Let's find out!

(Read `[x]` as "contains x")

```
# Kleisli monad in ~Haskell
# 💎 is a monad iff:
# 1. there is a constructor operation

💎 ::  a -> 💎 a     - read "💎 a" as 💎 contains a

# 2. there is a composition operation

f ::  a -> 💎 b
g ::  b -> 💎 c
f 🐡 g    ::  a -> 💎 c

# (Read `::` as "has type" and `->` as "returns".)
```

Problem statement: measuring internet censorship
Initial naive measurement pipeline

```
# kliesly_compose/mocks.py

def dns(domain: str) -> IpAddr: ...

>>> dns("www.example.com")
IPAddr("130.192.91.231")

>>> dns("www.instagram.com")
Traceback:
  ...
    Raise DNSError("no such domain")
DNSError: no such domain
```

```
# kliesly_compose/mocks.py

def fetch(addr: IpAddr) -> WebPage: ...

>>> fetch(IpAddr("130.192.91.231"))
WebPage("<html> ...")

>>> fetch(IpAddr("130.192.91.211"))
Traceback:
  ...
    raise FetchError("connection reset by peer")
FetchError: connection reset by peer
```

```
# kliesly_compose/base.py

def measure(domain: str) -> WebPage:
  return fetch(dns(domain))

>>> measure("www.example.com")
WebPage("<html> ... </html>")

>>> measure("www.instagram.com")
Traceback:
  ...
    raise DNSError("no such domain")
DNSError: no such domain
```

```
# kliesly_compose/base.py & examples/000.py

def measure_list(domains: list[str]) -> list[WebPage]:
  result = []
  for domain in domains:
    try:
      result.append(measure(domain))
    except:
      pass
  return result

>>> measure_list(["www.example.com", "www.instagram.com"])
[WebPage("<html> ... </html>")]

# 😭 "Varo, ubi sunt errores?"
```

Introducing a Monad constructor 💎

Instead of using the type `T`, we use `T | Exception`:

- `WebPage` becomes `WebPage | Exception`
- `IpAddr` becomes `IpAddr | Exception`

We could write this as a `@dataclass` but it's not as idiomatic as just writing the union type inline.

In other words:

`💎 T` corresponds to `T | Exception`

```
# kliesly_compose/catcher.py & examples/002.py

def measure_list(
  domains: list[str]) -> list[WebPage | Exception]: # 💎
  result = []
  for domain in domains:
    try:
      result.append(measure(domain))
    except Exception as exc:
      result.append(exc)
  return result

>>> measure_list(["www.example.com", "www.instagram.com"])
[WebPage("<html> ..."), DNSError("no such host")]
```


IL FAUT…
CHARBONNER!

```
# Objective: take the exceptions shock away

def measure_list(
  domains: list[str]) -> list[WebPage | Exception]:
  result = []
  for domain in domains:
    result.append(measure(domain))
  return result

# How?
```

```
# kliesly_compose/bettercatcher.py & examples/002.py

def measure(domain: str) -> WebPage | Exception:  # 💎
  try:
    addr = dns(domain)
  except Exception as exc:
    return exc

  try:
    return fetch(addr)
  except Exception as exc:
    return exc
```

```
# Objective: take the exceptions shock away

def measure(domain: str) -> WebPage | Exception:
  return fetchx(dnsx(domain))

# How?
```

```
def dnsx(domain: str) -> IpAddr | Exception: # 💎
  try:
    return dns(domain)
  except Exception as exc:
    return exc

def fetchx(
  addr: IpAddr | Exception) -> WebPage | Exception: # 💎
  if isinstance(addr, Exception):
    return addr
  try:
    return fetch(addr)
  except Exception as exc:
    return exc
```

```python
# kliesly_compose/landing.py & examples/003.py
# Objective: extract a monadic building block from fetchx

def fetchx(
  addr: IpAddr | Exception) -> WebPage | Exception:
  if isinstance(addr, Exception):
    return addr
  return fetchxy(addr)

def fetchxy(addr: IpAddr) -> WebPage | Exception: # 💎
  try:
    return fetch(addr)
  except Exception as exc:
    return exc
```

```
# Now, this starts to feel very monadic
# f: A -> B | Exception

def dnsx(domain: str) -> IpAddr | Exception: ...

def fetchxy(addr: IpAddr) -> WebPage | Exception: ...

# The time is ripe to write the 🐡 operator!
```

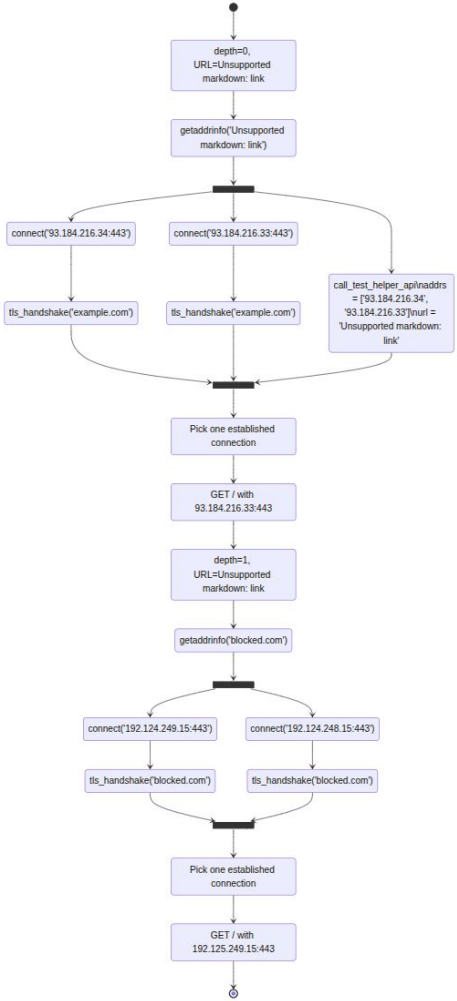
```python
class Compose:   # 🐡
  def __init__(
    self,
    fx: Callable[[str], IpAddr | Exception],
    gx: Callable[[IpAddr], WebPage | Exception],
  ):
    self.fx, self.gx = fx, gx

  def __call__(self, value: str) -> WebPage | Exception:
    rv = self.fx(value)
    if isinstance(rv, Exception):
      return rv
    return self.gx(rv)
```

```
# kliesly_compose/full.py & examples/004.py

def measure(domain: str) -> WebPage | Exception:
  pipeline = Compose(dnsx, fetchxy)  # f 🐡 g
  return pipeline(domain)
```

```python
class Compose[A, B, C]:  # Make 🐡 generic!
  def __init__(
    self,
    fx: Callable[[A], B | Exception],
    gx: Callable[[B], C | Exception],
  ):
    self.fx, self.gx = fx, gx

  def __call__(self, value: A) -> C | Exception:
    rv = self.fx(value)
    if isinstance(rv, Exception):
      return rv
    return self.gx(rv)
```

```
# So, the measurement algorithm now reduces to

def dnsx(domain: str) -> IpAddr | Exception:
  try:
    return dns(domain)
  except Exception as exc:
    return exc

def fetchxy(addr: IpAddr) -> WebPage | Exception:
  try:
    return fetch(addr)
  except Exception as exc:
    return exc

def measure_list(domains: list[str]) -> list[WebPage | Exception]:
  return list(map(Compose(dnsx, fetchxy), domains))
```

```
# kliesly_compose/fancy.py & examples/005.py

...

# f 🐡 g

measure = dnsx | fetchxy

# more similar to what we were trying to do

import ooni as oo
measure = oo.dns | oo.tcp | oo.tls | oo.http
```


depth=0, URL=Unsupported markdown: link
getaddrinfo('Unsupported markdown: link')
connect('93.184.216.34:443')
connect('93.184.216.33:443')
call_test_helper_api\naddrs = ['93.184.216.34', '93.184.216.33']\nurl = 'Unsupported markdown: link'
tls_handshake('example.com')
tls_handshake('example.com')
Pick one established connection
GET / with 93.184.216.33:443
depth=1, URL=Unsupported markdown: link
getaddrinfo('blocked.com')
connect('192.124.249.15:443')
connect('192.124.248.15:443')
tls_handshake('blocked.com')
tls_handshake('blocked.com')
Pick one established connection
GET / with 192.125.249.15:443


SADZ
NEW AMERICA
NEW

Sono intorno a noi

sono in mezzo a noi

you cannot unsee this

```
% set -euo pipefail

% find . -type f -name \*.py |
    awk -F/ '{print $NF}' |
    sort |
    uniq -c |
    cat |
    sort -rnk1 |
    head -n1
```

Workflows

Workflows allow for organizing the single operations in a advanced management pipelines, to perform a series operation of data processing, ML model training and serving, etc. Workflows represent long-running procedures defined as Directed Acyclic Graphs (DAGs) where each node is a single unit of work performed by the platform (e.g., as a Kubernetes Job).

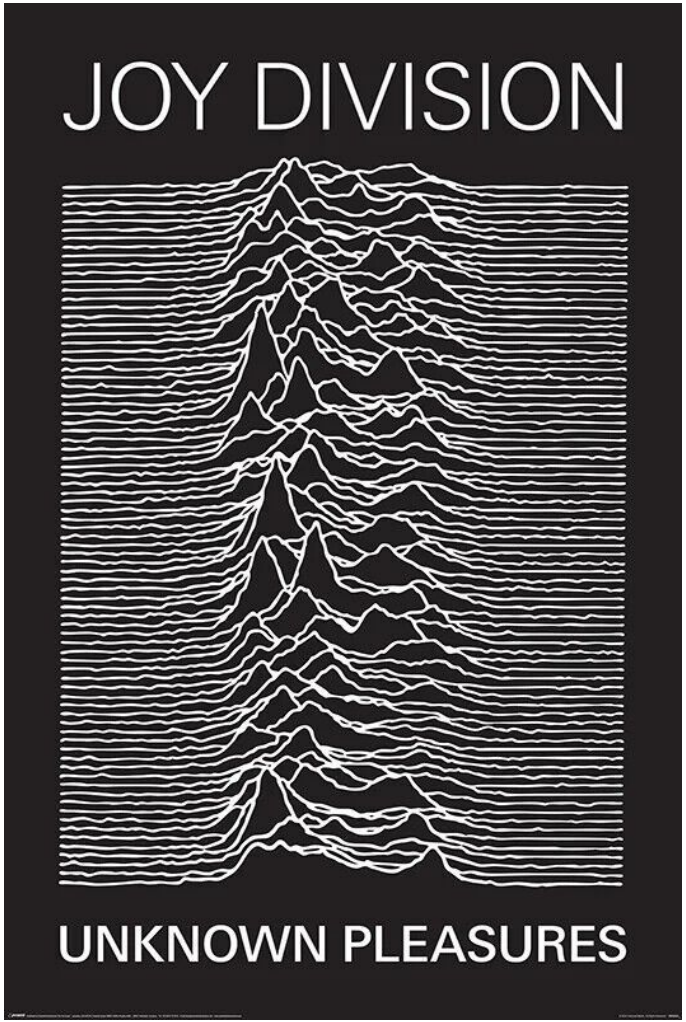
As in case of functions, it is possible for the platform to have different workflow runtimes. Currently, the only workflow runtime implemented is the one based on Kubeflow Pipelines infrastructure. See KFP Runtime for further details about how the workflow is defined and executed with the Kubeflow Pipelines component of the platform.

Similarly, to functions the workflows may be managed via console UI or via Python SDK.

Management via UI

Workflows can be created and managed as *entities* from the console. You can access them from the dashboard or the left menu. You can:

- `create` a new workflow
- `expand` a workflow to see its 5 latest versions
- `show` the details of a workflow



We (aim to) build robust systems by accepting failure as part of the computation, not as an afterthought.

Monads—like `Compose`—help us structure this with grace.

But, is this real fun?

—

I've got the spirit
And I’ve lost the feeling
Take the shock away 💎