
# Making Sanic Even *Faster*

## What's in store for v21

Adam Hopkins

```
start = datetime(2021, 3, 24, 12, 0, 0, tzinfo=ZoneInfo(key="America/New_York"))
end = start + timedelta(minutes=45)
```


```python
class Adam:

    def __init__(self):
        self.work = PacketFabric("Sr. Software Engineer")
        self.oss = Sanic("Core Maintainer")
        self.home = Israel("Negev")

    async def run(self, inputs: Union[Pretzels, Coffee]) -> None:
        while True:
            await self.work.do(inputs)
            await self.oss.do(inputs)

    def sleep(self):
        raise NotImplemented
```

- PacketFabric - Network-as-a-Service platform; private access to the cloud; secure connectivity between data centers
- Sanic Framework - Python 3.7+ `asyncio` enabled framework and server. Build fast. Run fast.
- GitHub - /ahopkins
- Twitter - @admhpkns

# What is Sanic?

# What is Sanic?

## Framework

```
from sanic import Sanic, text

app = Sanic("My Hello, world app")

@app.get("/")
async def hello_world(request):
    return text("Hello, world.")
```

# What is Sanic?

## Framework

```
from sanic import Sanic, text

app = Sanic("My Hello, world app")

@app.get("/")
async def hello_world(request):
    return text("Hello, world.")
```

## Web server (production ready)

```
$ sanic server.app
$ python -m sanic server.app
$ python server.py
```

| | Framework | Server | ASGI ready |
|---|---|---|---|
| Django | | | |
| Flask | | | |
| Starlette | | | |
| Gunicorn | | | |
| Hypercorn | | | |
| Nginx | | | - |
| Sanic | | | |

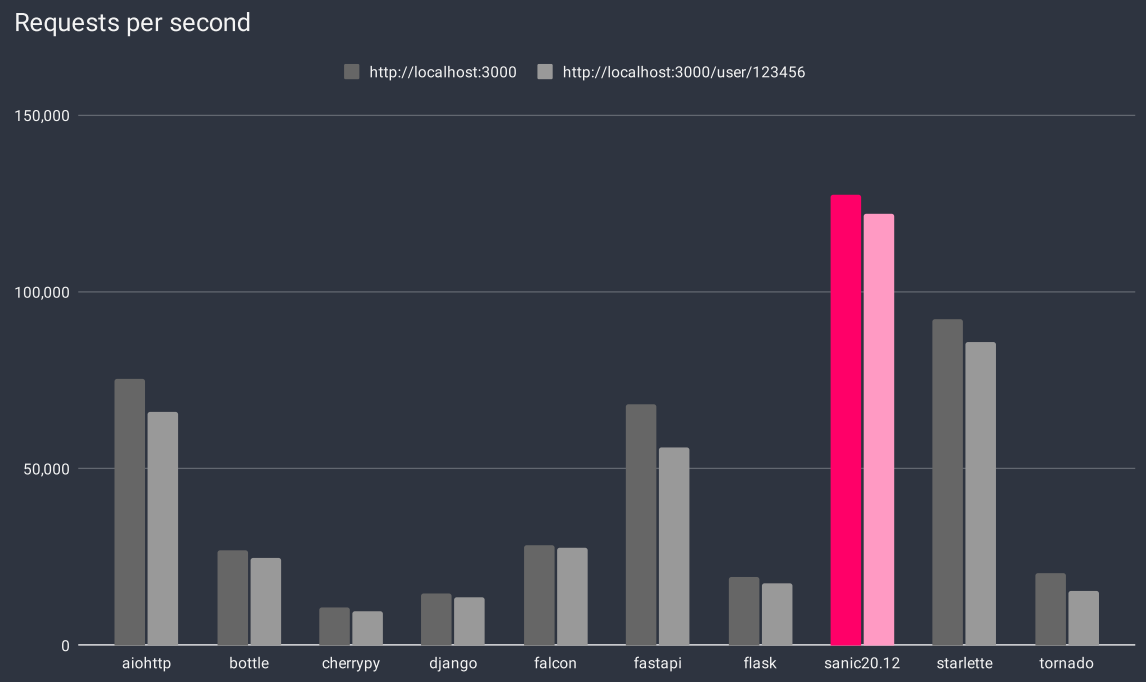
# Hot off the press!

# Quick Overview

1. What's new in 21.3
    - A lot!
    - Optimized routing
    - Stream everything
    - Signals
2. Why these changes were made
3. How these changes were made

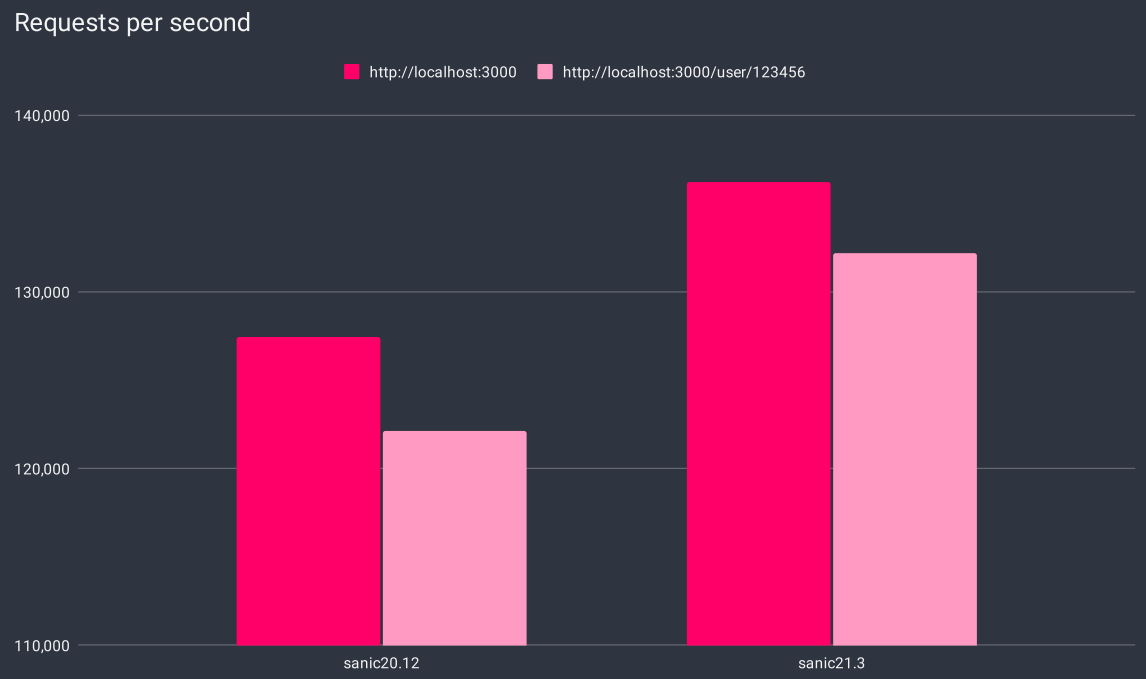

Benchmarks are lousy
CHANGE MY MIND
imgflip.com

# How does Sanic 20.12 stack up?




Higher is better

# ... and v21.3?



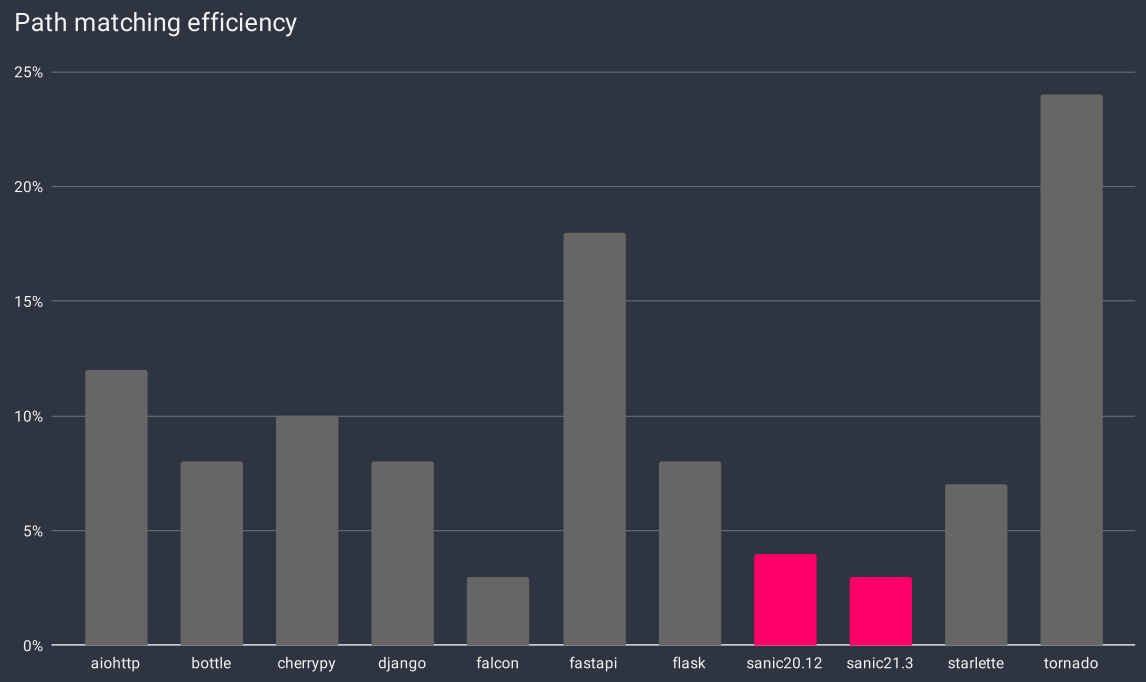

Higher is better

# Why is it faster?

# Why is it faster?

… *Efficiency*

# Router efficiency




Lower is better

# What is a router?

## From this...

- `/login`
- `/profile/<username>`
- `/orders`
- `/orders/<order_id:int>`

## ... to this ...

```
def login(request): ...
def view_profile(request): ...
def view_all_orders(request): ...
def view_single_order(request): ...
```

## ... using this:

```
GET /v4 HTTP/1.1
Host: localhost:8181
User-Agent: curl/7.75.0
Accept: */*
```

# Old Sanic Router

From this:

```
/foo/<bar:int>/<fuzz:alpha>/<buzz:number>
```

To this:

```
r"/foo/(-?\d+)/([A-Za-z]+)/(-?(?:\d+(?:\.\d*)?|\.\d+))"
```

# Old Sanic Router

From this:

```
/foo/<bar:int>/<fuzz:alpha>/<buzz:number>
```

To this:

```
r"/foo/(-?\d+)/([A-Za-z]+)/(-?(?:\d+(?:\.\d*)?|\.\d+))"
```

Using this:

```
for route in routes:
    match = route.pattern.match(url)
    if match and method in route.methods:
        break
```

There must be a better way

# Runtime optimized router

- AST style compiled
- Route matching does not exist *until* startup
- Analyze declared routes into a
- Build matching function using the

```
/a/much/longer/item/<id>
/a/much/<longer>/item/<id>
/a/thing/<id>
/a/thing/<id>/and/doit
/a/<banana>
/<foo>
/<foo>/bar
/<foo>/buzz
/<foo>/fuzz
/<foo:int>
```

```
/a/much/longer/item/<id>
/a/much/<longer>/item/<id>
/a/thing/<id>
/a/thing/<id>/and/doit
/a/<banana>
/<foo>
/<foo>/bar
/<foo>/buzz
/<foo>/fuzz
/<foo:int>
```

```
Node(level=0)
     Node(part=a, level=1)
         Node(part=much, level=2)
             Node(part=longer, level=3)
                 Node(part=item, level=4)
                     Node(part=<id>, level=5, route=<Route: /a/much/longer/item/<id>>, dynamic=True)
             Node(part=<longer>, level=3, dynamic=True)
                 Node(part=item, level=4)
                     Node(part=<id>, level=5, route=<Route: /a/much/<longer>/item/<id>>, dynamic=True)
         Node(part=thing, level=2)
             Node(part=<id>, level=3, route=<Route: /a/thing/<id>>, dynamic=True)
                 Node(part=and, level=4)
                     Node(part=doit, level=5, route=<Route: /a/thing/<id>/and/doit>)
         Node(part=<banana>, level=2, route=<Route: /a/<banana>>, dynamic=True)
     Node(part=<foo>, level=1, route=<Route: /<foo>>, dynamic=True)
         Node(part=bar, level=2, route=<Route: /<foo>/bar>)
         Node(part=buzz, level=2, route=<Route: /<foo>/buzz>)
         Node(part=fuzz, level=2, route=<Route: /<foo>/fuzz>)
     Node(part=<foo:int>, level=1, route=<Route: /<foo:int>>, dynamic=True)
```

# Is this better?

## v20.12

```
-------------------------------------------------------------------------------
Name (time in ns)   Min               Max               Mean              StdDev
-------------------------------------------------------------------------------
static_route        183.5220 (1.0)    260.4150 (1.0)    193.0517 (1.0)     6.9385 (1.0)
dynamic_route       184.8240 (1.01)   590.3360 (2.27)   201.4882 (1.04)   30.8262 (4.44)
-------------------------------------------------------------------------------
```

## v21.3

```
-------------------------------------------------------------------------------
Name (time in ns)   Min               Max               Mean              StdDev
-------------------------------------------------------------------------------
static_route        104.7200 (1.0)    192.7400 (1.0)    109.5443 (1.0)    7.1217 (2.22)
dynamic_route       117.8700 (1.13)   193.0490 (1.00)   126.3849 (1.15)   3.2035 (1.0)
-------------------------------------------------------------------------------
```

# SIDETRACK

```
def if_else(path):
    parts = path.split("/")
    if parts[0] == "":
        if parts[1] == "foo":
            if len(parts) == 2:
                return ROUTES[(parts[0], parts[1], None)]
    return False

def and_parts(path):
    parts = path.split("/")
    if parts[0] == "" and parts[1] == "foo" and len(parts) == 2:
        return ROUTES[(parts[0], parts[1], None)]
    return False

def path_grab(path):
    parts = path.split("/")
    return ROUTES[(parts[0], parts[1], None)]
```

# SIDETRACK

```
@timy.timer(loops=LOOPS)
def do_if_else():
    if_else(PATH)

@timy.timer(loops=LOOPS)
def do_and_parts():
    and_parts(PATH)

@timy.timer(loops=LOOPS)
def do_path_grab():
    path_grab(PATH)

do_if_else()
do_and_parts()
do_path_grab()
```

# SIDETRACK

```
Timy executed (do_if_else) for 500000 times in 0.254344 seconds
Timy best time was 0.000000 seconds
Timy executed (do_and_parts) for 500000 times in 0.196365 seconds
Timy best time was 0.000000 seconds
Timy executed (do_path_grab) for 500000 times in 0.221170 seconds
Timy best time was 0.000000 seconds
```

```
compiled_src = compile(find_route_src, "", "exec")
ctx = {}
exec(compiled_src, None, ctx)
find_route = ctx["find_route"]
```

```
compiled_src = compile(find_route_src, "", "exec")
ctx = {}
exec(compiled_src, None, ctx)
find_route = ctx["find_route"]
```

```
compile(src, filename, mode)
```

- `src` : the source that will be compiled, `str` , `bytes` , or AST object
- `filename` : from where source came from
- `mode` :
  - `"eval"` : single expression
  - `"exec"` : block of statements
  - `"single"` : single interactive statement

```
src = [
    Line("def find_route(path, router, basket, extra):", 0),
    Line("parts = tuple(path[1:].split(router.delimiter))", 1),
    Line("try:", 1),
    Line("route = router.static_routes[parts]", 2),
    Line("basket['__raw_path__'] = path", 2),
    Line("return route, basket", 2),
    Line("except KeyError:", 1),
    Line("pass", 2),
]
```

```
def find_route(path, router, basket, extra):
    parts = tuple(path[1:].split(router.delimiter))
    try:
        route = router.static_routes[parts]
        basket['__raw_path__'] = path
        return route, basket
    except KeyError:
        pass
```

```
def find_route(path, router, basket, extra):
    parts = tuple(path[1:].split(router.delimiter))
    try:
        route = router.static_routes[parts]
        basket['__raw_path__'] = path
        return route, basket
    except KeyError:
        pass
```

```
find_route_src = """<see above>"""
```

```
compiled_src = compile(find_route_src, "", "exec")
ctx = {}
exec(compiled_src, None, ctx)
find_route = ctx["find_route"]
```

```
exec(object[, globals[, locals]])
```

- `object` : the code object to be executed
- `globals` : the global context to be applied, ie `globals()`
- `locals` : the local context to be applied, ie `locals()`

```
compiled_src = compile(find_route_src, "", "exec")
ctx = {}
exec(compiled_src, None, ctx)
find_route = ctx["find_route"]
```

```
def find_route(path, router, basket, extra):
    ...

find_route(...)
```

# Potential applications in Sanic

- Optimizing request/response cycle
- Minimizing middleware lookups
- Streamlining HTTP and WS response logic
- Smarter error response handling
- Predictive caching

```
@app.post("/login")
def login(request):
    ...

@app.get("/profile/<username>")
def view_profile(request):
    ...

@app.get("/orders")
def view_all_orders(request):
    ...

@app.get("/orders/<order_id:int>")
def view_single_order(request):
    ...

@app.put("/orders/<order_id:int>")
def update_single_order(request):
    ...
```

Static routes

- `/login`
  - `POST` handler
- `/orders`
  - `GET` handler

Dynamic routes

- `/profile/<username>`
  - `GET` handler
- `/orders/<order_id:int>`
  - `GET` handler
  - `PUT` handler

```
def find_route(path, router, basket, extra):
    parts = tuple(path[1:].split(router.delimiter))
    try:
        route = router.static_routes[parts]
        basket['__raw_path__'] = path
        return route, basket
    except KeyError:
        pass
    num = len(parts)
    if num > 0:
        if parts[0] == "orders":
            if num == 2:
                basket[1] = parts[1]
                try:
                    basket['__params__']['order_id'] = int(basket[1])
                except ValueError:
                    ...
                else:
                    basket['__raw_path__'] = 'orders/<order_id:int>'
                    return router.dynamic_routes[('orders', '<order_id:int>')], basket
            raise NotFound
        elif parts[0] == "profile":
            if num == 2:
                basket[1] = parts[1]
                try:
                    basket['__params__']['username'] = str(basket[1])
                except ValueError:
                    ...
                else:
                    basket['__raw_path__'] = 'profile/<username>'
                    return router.dynamic_routes[('profile', '<username>')], basket
            raise NotFound
        raise NotFound
    raise NotFound
```

# Stream *EVERYTHING*

- Unify **all** response types:
  - `text()`, `json()`, `html()`, `file()`, `stream()`, `file_stream()`
- Lighten the path to HTTP/2 server push
- Remove callbacks
- Side benefit: *SPEED*

# Old Sanic Request/Response

```
handler = router.get(request)
response = await handler(request)
if stream_callback:
    await stream_callback(response)
else:
    write_callback(response)
```

```python
@app.route("/")
async def index(request):
    async def stream_from_db(response):
        conn = await asyncpg.connect(database='test')
        async with conn.transaction():
            async for record in conn.cursor('SELECT generate_series(0
                await response.write(record[0])

    return stream(stream_from_db)
```

```python
@app.route("/")
async def index(request):
    async def stream_from_db(response):
        conn = await asyncpg.connect(database='test')
        async with conn.transaction():
            async for record in conn.cursor('SELECT generate_series(0
                await response.write(record[0])

    return stream(stream_from_db)
```

```python
@app.route("/")
async def test(request):
    response = await request.respond(content_type="text/csv")
    await response.send("foo,")
    await response.send("bar")
    await response.send("", True)
    return response
```

## Version 20.12

```
Running 30s test @ http://127.0.0.1:3333
  12 threads and 400 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     4.48ms    5.95ms 131.35ms   91.06%
    Req/Sec    10.21k     1.94k   56.63k    76.23%
  3659132 requests in 30.09s, 411.78MB read
Requests/sec: 121614.26
Transfer/sec:     13.69MB
```

## With new streaming internals

```
Running 30s test @ http://127.0.0.1:3333
  12 threads and 400 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     3.81ms    4.55ms 107.27ms   90.82%
    Req/Sec    11.22k     2.29k   36.57k    74.27%
  4014399 requests in 30.09s, 394.33MB read
Requests/sec: 133394.06
Transfer/sec:     13.10MB
```

## With new AST router

```
Running 30s test @ http://127.0.0.1:3333
  12 threads and 400 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     3.74ms    4.60ms 102.41ms   90.73%
    Req/Sec    11.60k     2.61k   46.51k    75.23%
  4148337 requests in 30.08s, 407.48MB read
Requests/sec: 137923.77
Transfer/sec:     13.55MB
```

# Introducing signals

```
@app.signal("foo.bar.baz")
async def handler():
    ...
```

- Convenient method to push work to the event loop
- Can be dispatched from anywhere in your application
- Familiar API resembling route handlers
- *BETA* in 21.3
  - Intended to replace listeners and middleware in future
- Built using the new router

# Dispatch with context

```python
@app.signal("user.registration.created")
async def send_registration_email(context):
    await send_email(context["email"], template="registration")

@app.post("/register")
async def handle_registration(request):
    await do_registration(request)

    await request.app.dispatch(
        "user.registration.created",
        context={"email": request.json.email}
    )
    return redirect(request.app.url_for("profile"))
```

# Dynamic paths

```
@app.signal("foo.bar.<thing>")
async def signal_handler(thing):
    print(f"[signal_handler] {thing=}")

@app.post("/trigger")
async def trigger(request):
    await app.dispatch("foo.bar.baz")
    return text("Done.")
```

# Internal messaging

```
@app.signal("do.something.expensive")
async def signal_handler(thing):
    await do_something()
    await app.dispatch("do.something.complete")

@app.post("/trigger")
async def trigger(request):
    await app.dispatch("do.something.expensive")

    await app.event("do.something.complete")

    return text("Done.")
```

# What else?

- NEW docs
- Drop Python 3.6
- Expanded and consistent route naming
- New convenience decorators
- Alterable route `match_info`
- New version types allowd in routes (int, float, str)
- App, Blueprint, and Blueprint group parity
- Removed testing client to standalone package
- Application and connection level context objects

# Questions?

GitHub - /ahopkins

Twitter - @admhpkns

PacketFabric - packetfabric.com

Sanic homepage - sanicframework.org

Sanic repo - /sanic-org/sanic