CHANNELS 2.0

ARTEM MALYSHEV

@PROOFIT404

CHANNELS 1.0

3 years old design

no standard interface like WSGI

push ***everything*** over network

tricky deploy

django session abuse

3 YEARS OF EXPERIENCE

800+ commits

450+ issues closed

100+ contributors

40+ releases

NO STANDARD INTERFACE

PEP 333

This wide variety of choices can be a problem for new Python users, because generally speaking, their choice of web framework will limit their choice of usable web servers, and vice versa.

PUSH EVERYTHING OVER NETWORK

TBH, the main reason I like microservices is that I feel like my method calls are too fast and I'd prefer to throw in some latency.

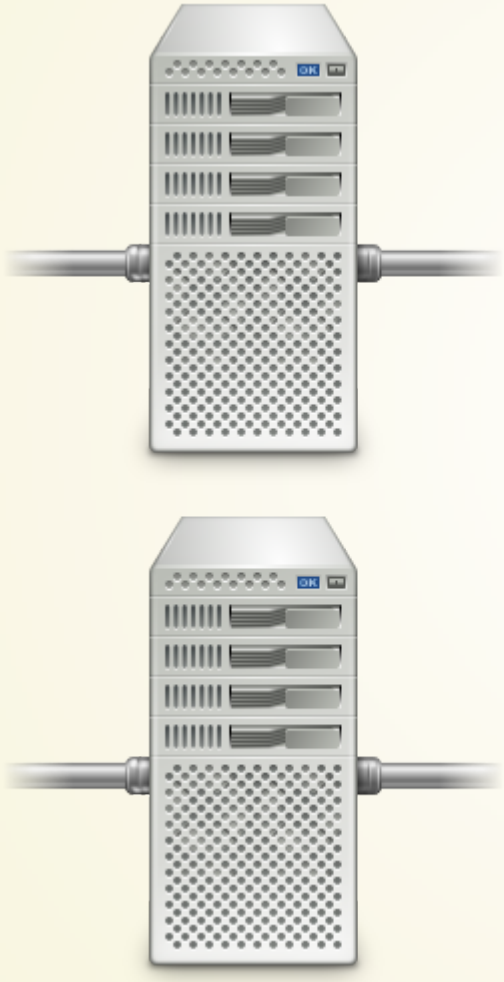
Aaron Patterson (@tenderlove)

March 6, 2015

TRICKY DEPLOY

FOR HELLO WORLD

FOR CHAT



SESSION USAGE

```
@channel_session
```

```
@http_session
```

```
@channel_and_http_session
```

```
@channel_and_http_session_user_from_http
```

```
@enforce_ordering
```

GROUPS ISSUES

Mostly used wrong

Exposed to user

Too complex to be fully implemented

ASGI REDIS ISSUES

Incomplete groups support

Daphne constantly pools redis

Does not support transparent scale

ASGI RABBITMQ ISSUE

Have ***really*** complex implementation

Needs ***really*** careful production setup

But fully compatible with ASGI spec

CHANNELS 1.0

Too complex to show incoming message counter


uvicorn

ACTUAL REQUIREMENTS

Send to channel from everywhere

Simultaneous usage of sync and async code

React on websocket events on another machine

Cross-socket and cross-process communication

MOVING THE LINE

Run workers and Daphne in the same process

Store socket state locally

Remove "send-to-layer" conventions

HELLO WORLD DEPLOY

CHAT DEPLOY

IMPLEMENTATION

Sync <=> async bridge

Twisted over asyncio

Remove Python 2 support

Consumers as first class citizens

Routing is a consumer too

SYNC TO ASYNC

```
class SyncToAsync:

    threadpool = ThreadPoolExecutor()

    def __init__(self, func):
        self.func = func

    async def __call__(self, *args, **kwargs):
        loop = asyncio.get_event_loop()
        future = loop.run_in_executor(
            self.threadpool,
            partial(self.func, *args, **kwargs),
        )
        return await asyncio.wait_for(future)
```

ASYNC TO SYNC

```
class AsyncToSync:

    def __init__(self, awaitable):
        self.awaitable = awaitable

    def __call__(self, *args, **kwargs):
        call_result = Future()
        event_loop.call_soon_threadsafe(
            asyncio.ensure_future,
            wrap(self.awaitable args, kwargs),
        )
        call_result.result()
```

MODERN TWISTED

```
@router.route("/gethostbyname/<name>")
async def hostname(self, request: IRequest) -> IResponse:
    try:
        address = await getHostName()
    except DNSNameError:
        request.setResponseCode(http.NOT_FOUND)
        return "no such host"
    except DNSLookupError:
        request.setResponseCode(http.NOT_FOUND)
        return "lookup error"
    return address
```

CONSUMERS

```
class AsyncChatConsumer(AsyncConsumer):

    async def websocket_connect(self, message):

        await self.send({
            "type": "websocket.accept",
        })

        self.username = "Anonymous"

        await self.send({
            "type": "websocket.send",
            "text": "[Welcome %s!]" % self.username,
        })
```

ROUTING

```
application = ProtocolTypeRouter({
    "http": URLRouter([
        url("^", DjangoViewSystem),
    ]),
    "websocket": URLRouter([
        url("^chat/$", AsyncChatConsumer),
    ]),
    "mqtt": MqttTemperatureConsumer,
    "email": EmailToRouter([
        regex("@support.org", SupportTicketHandler),
    ]),
    "sms": SMSFromRouter([
        phone("+1", USTextHandler),
    ]),
})
```

RESULTS

Removes a lot of handshaking traffic

Groups are be hidden in the consumer

First steps to async Django

LINKS

Channels 2.0 Docs
Towards Channels 2.0
Channels 2: October
Uvicorn: The lightning-fast asyncio server.

TO BE CONTINUED