



PyCon JP 2018
ひろがる Python

Djangoだって カンバンつくれるもん

Django Channelsで作るうカンバンアプリケーション

scouty

PyConJP 2018

@denzowill or denzow

**タイトルを書いてたときは酔ってた。
今は後悔している**

- でんぞう (@denzowill, denzow)
- scouty,inc シニアエンジニア
 - Scrapy, Djangoあたり
- DBスペシャリスト(PostgreSQLが好き、●racleは得意だけど苦手)
- (元?)StartPythonClubスタッフ



The image shows a presentation slide with a dark blue background. At the top left is a logo with the letters 'S' and 'J'. At the top center is the text 'いまさら振り返る Django Migration'. In the top right corner is a logo for '2018 Tokyo Django Congress' with the text 'djangocongress.jp'. The main title 'いまさら振り返る Django Migration' is centered in large white font. Below it is the subtitle 'Migrationの内部動作からやっちゃった事例まで' in smaller white font. At the bottom center is the 'scouty' logo followed by the text '20180519 DjangoCongress @denzowill or denzow'. The bottom of the slide features a footer with a 'denzow' logo and 'May 19, 2018' on the left, and 'Technology', '3 stars', '950 views', and a download icon on the right.

<https://speakerdeck.com/denzow/imasarazhen-rifan-ru-django-migration>

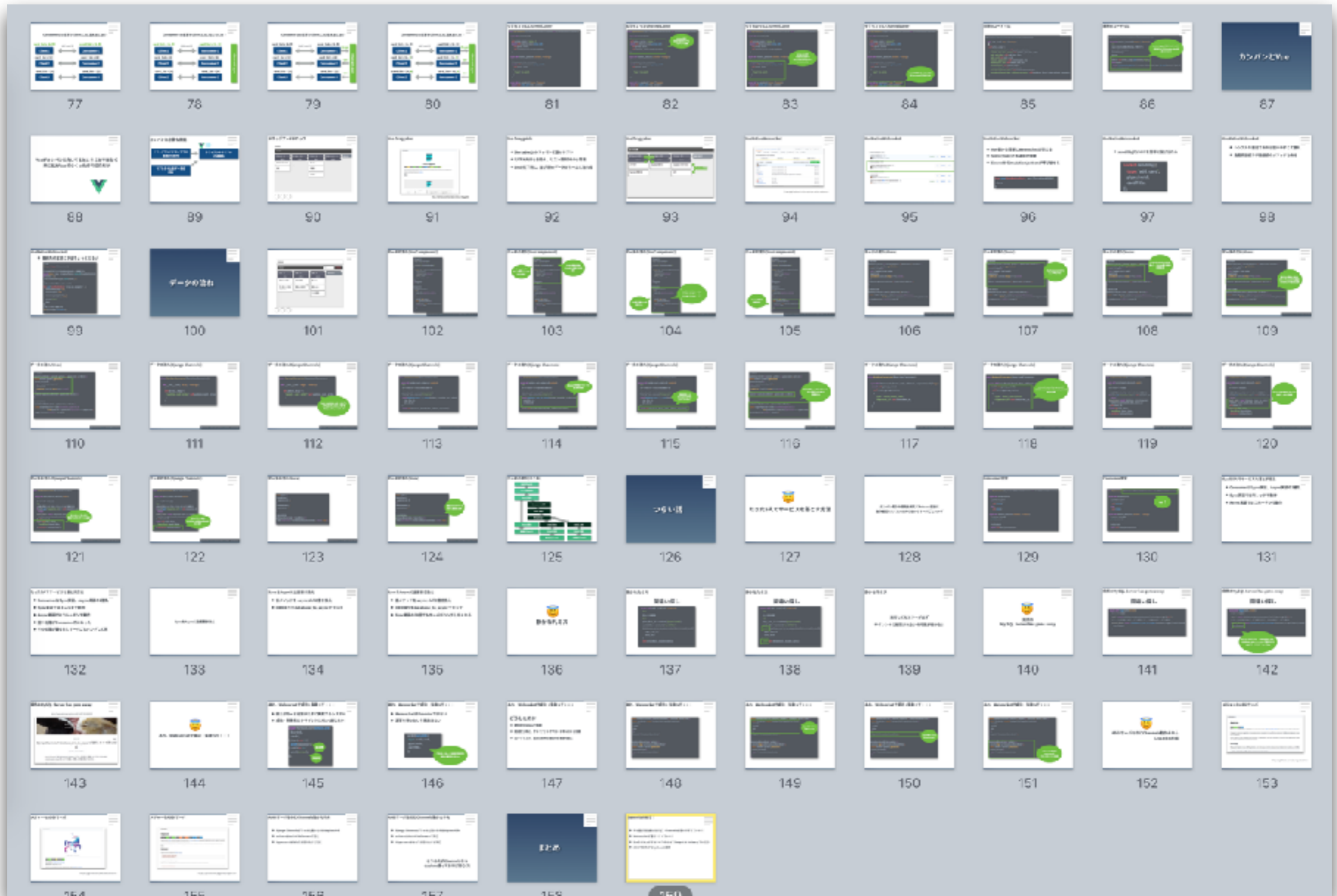
<https://qiita.com/denzow/items/77df4b45cfbbf2f0df92>

全90Pは50分には
長すぎた



<https://speakerdeck.com/denzow/imasarazhen-rifan-ru-django-migration>

<https://qiita.com/denzow/items/77df4b45cfbbf2f0df92>





170Pを45分で
お送りします

人工知能が、天職を探し出す。

日本初のAIヘッドハンティングサービス



scoutyとは

高度なAIで、能力に最適な企業を自動分析。
エンジニアのための、登録不要の新しい転職サービスです。

scouty のミッション

「世の中のミスマッチを無くす」

自分のまわりには、自分でも気づいていないたくさんの可能性や偶然性が存在するはずなのに、人はいつもそれに巡り会えるとは限りません。

そしてその結果、仕事や人材におけるミスマッチに悩む人も少なくはないでしょう。

scoutyは、インターネット上にあふれるデータと最先端の人工知能技術を使って情報と機会を適切にお届けすることで、偶然を必然に変え、世の中のミスマッチをなくしていくことを目指します。そして、それは結果として、個人の市場価値や生活の質を高め、企業の競争力を高めることにつながると考えています。

月1で麹町あたりで
100人くらいの勉強会やってきました
11月からは新橋

Start Python Club
Pythonでスタートする人たちの集い

開催前イベント [もっと見る](#)
2018/11/14(水) みんなのPython勉強会...
2018/12/11(火) みんなのPython勉強会...

イベント メンバー 資料 [BI 2](#) [G+](#) [いいね! 108](#) [ツイート](#) [メンバーになる](#)

次回イベント

	開催前 2018/11/14 (水) 19:00~ <u>みんなのPython勉強会#39</u> Takeshi Akutsu 他 〒105-0004 東京都港区新橋4-1-1 (新橋通りCORE)	0/155人
	開催前 2018/12/11 (火) 19:00~ <u>みんなのPython勉強会#40</u> Takeshi Akutsu 他 東京都港区新橋4-1-1	0/155人

終了したイベント

[全てのイベントを見る \(47件\)](#)

	2018/09/14 (金) 19:00~ <u>【Stapy x MUFG共催】Python Global Meetup (同時通訳あり)</u> Takeshi Akutsu 他 東京都千代田区有楽町1-12-1 (新有楽町ビル7F)	135/165人
--	--	----------

メンバー (4031人)

管理者
他のメンバー

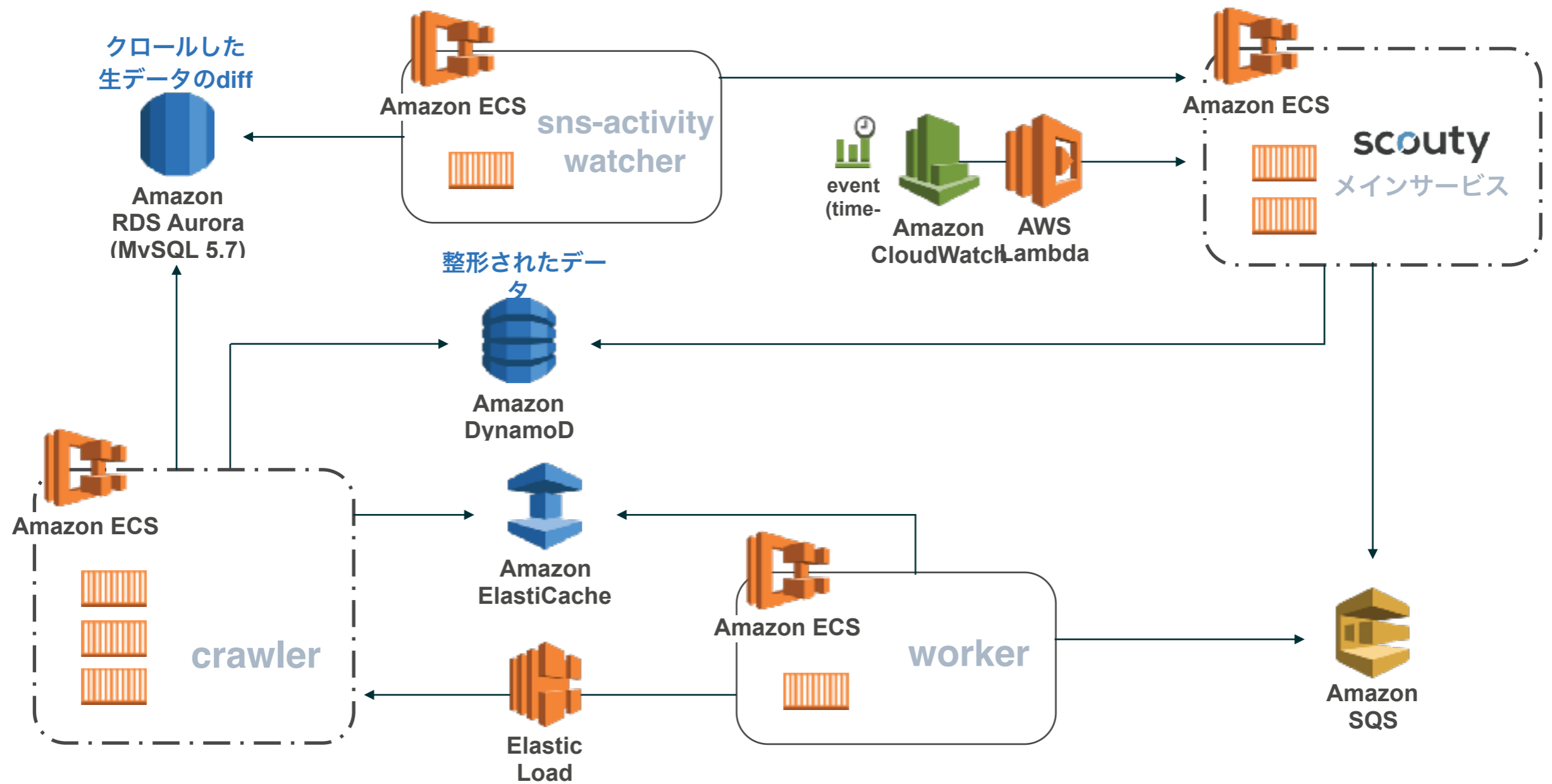
資料 (196件)

- Excel帳簿と仲良くなるツールを作...
倉前 & Shizuka.py #7 開催されたよ
- みんなのPython勉強会#38
- Entrance of Docker for Pythonista
~ Dockerではじめる 機械学習モデルのデプロイ ~
- みんなのPython勉強会#38
- 技術習得5で Django の使い (簿く...

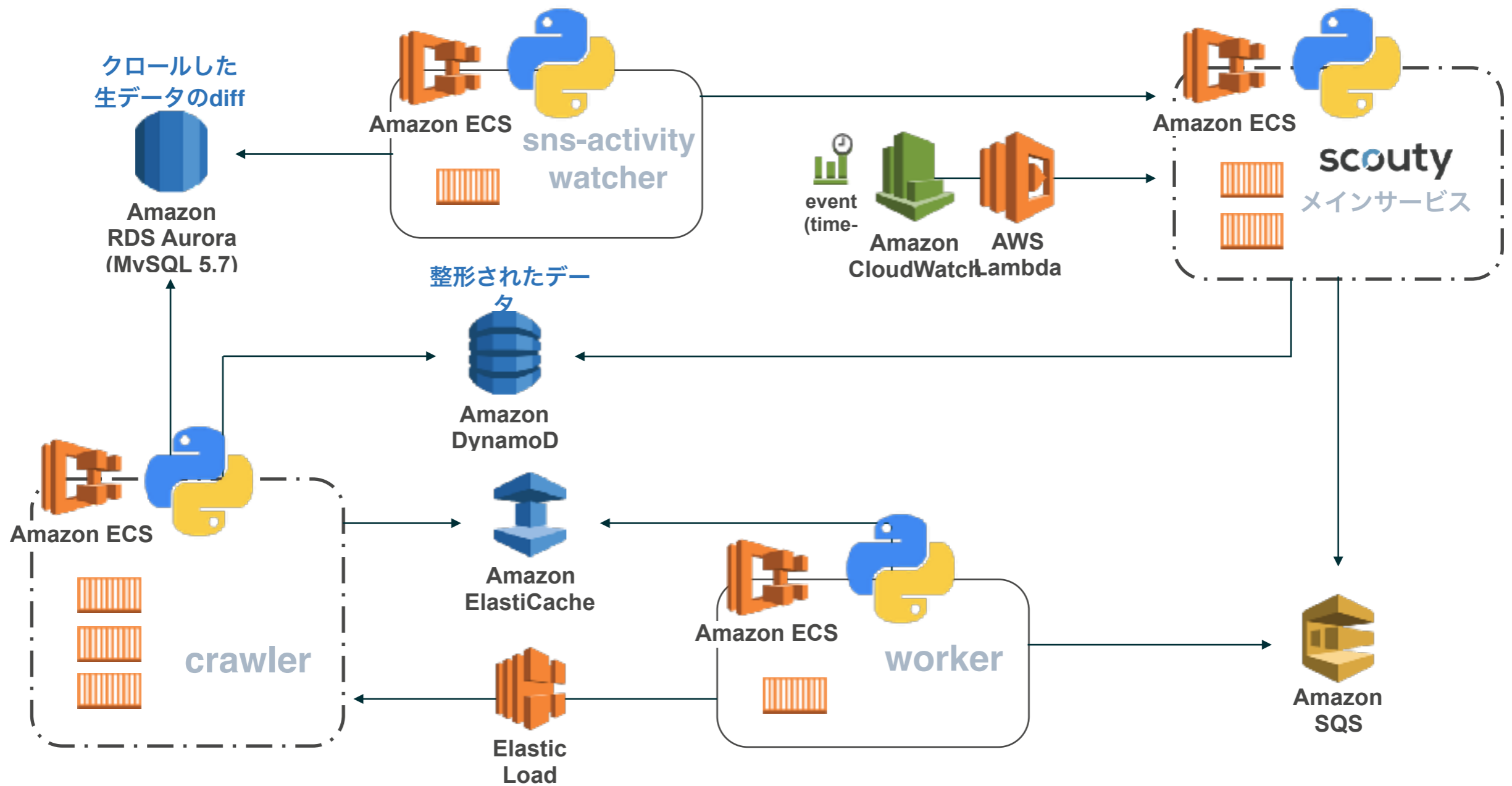
<https://startpython.connpass.com/>

Pythonとscouty

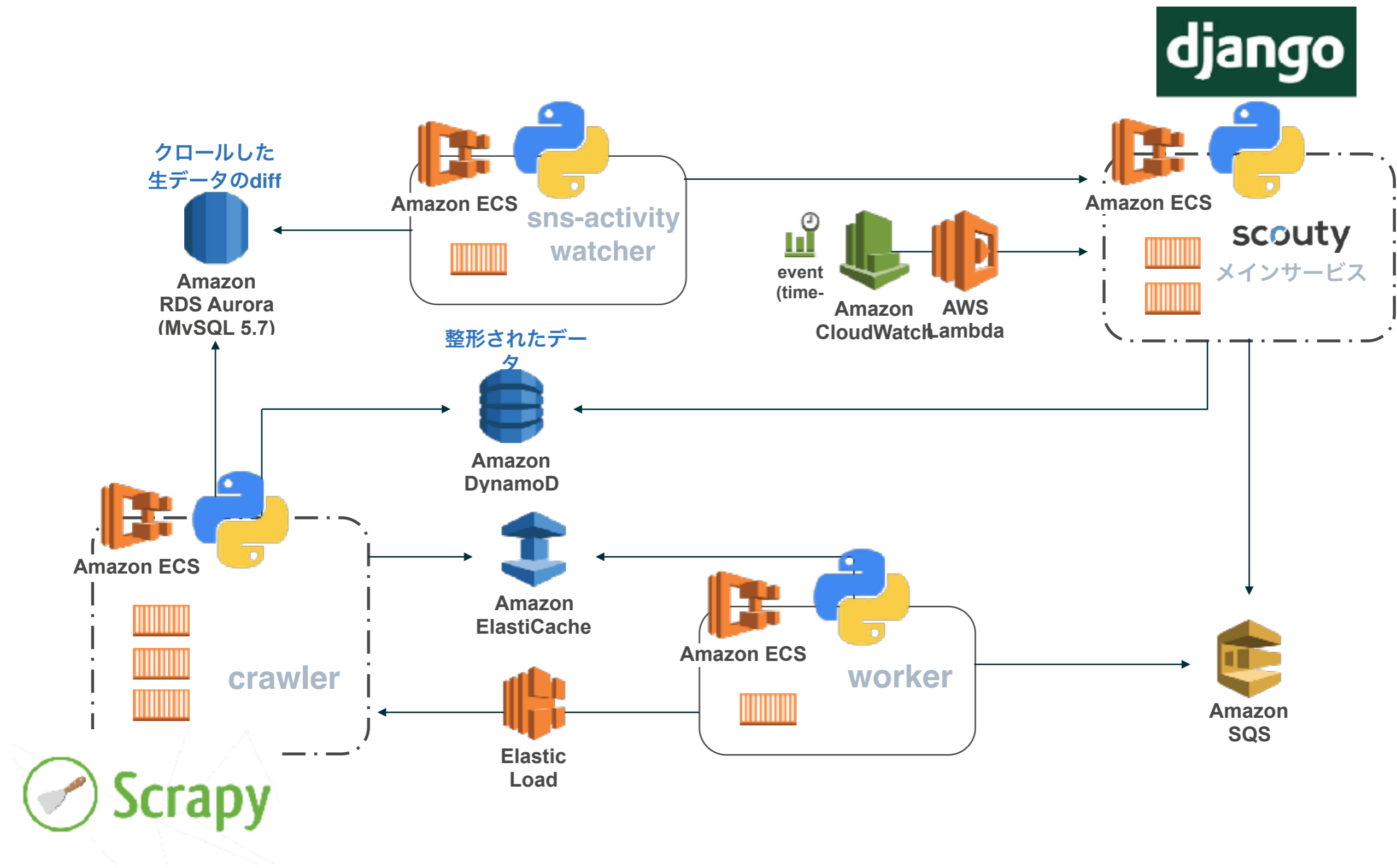
scouty インフラ構成図



scouty インフラ構成図



scouty インフラ構成図



お詫び

2018-9-18 13:30 Talk(45min) 特別会議室

発表言語: 日本語 資料言語: 日本語 Level: Beginner

Djangoだってカンバンつくれるもん(Django Channels + Vue)

DjangoのライブラリであるDjango Channelsを使用して、Websocketを使ったアプリケーションの構築についてお話しします。Trelloを作成した際の話が中心になります。

denzow

scoutyという会社でDBやバックエンドを中心に活動している、Djangoが好きなエンジニアです。

発表詳細

Django Channels 2.1系とVue.js + Vuexを使用して複数ブラウザでリアルタイムに同期するいわゆるカンバンアプリケーションを作るとの組み合わせの実践的な内容をコードを含めて紹介します。

* Django Channelsの概要

* Django Channelsの基本概念

- asgi
- routing
- consumer

* 本番でのアプリケーションサーバ選定(not uWSGI, use daphne)

Beginnerの定義読み違えた



Beginner向けにチュートリアル書いてくので許して

<https://qiita.com/denzow/items/046f3c8b9bd8d3378eb4>



@denzow 2018年09月18日に更新 5 views

編集する ⚙️

DjangoとVueでカンバンアプリケーションを作る(1)

Python

Django

カンバン

vue.js

PyConJP 2018の発表用に作成したDjangoDeKanbanを作成するチュートリアル記事です。

最終的に以下のようなアプリケーションが構築できることを目指します。

本題



A screenshot of a Trello board titled "PyConJP". The board is set to "プライベート" (Private) and "非公開" (Not Public). It features four columns: "TODO", "DOING", "DONE", and "FAILED". The background of the board is a scenic image of a mountain range with a waterfall. The "TODO" column contains one card: "打ち上げで美味しく乾杯する". The "DOING" column contains one card: "本番の発表". The "DONE" column contains three cards: "資料作成", "下調べ", and "発表の練習". The "FAILED" column contains two cards: "拍手喝采を浴びる" and "ここまでの発表で大受け". Each column has a "+ さらにカードを追加" (Add more cards) button at the bottom.

ZenHub

The screenshot displays the ZenHub interface for the repository `denzow / DjangoDeKanban`. At the top, there are navigation options: `<> Code`, `🕒 Issues 0`, `🔗 Pull requests 0`, `Z ZenHub` (the active tab), `📁 Projects 0`, `📖 Wiki`, `📊 Insights`, and `⚙️ Settings`. On the right side of the top bar, there are buttons for `👁️ Unwatch 1`, `★ Star 0`, and `🍴 Fork 0`.

Below the navigation bar, there is a row of filters: `Repos (1/1) ▾`, `🏷️ Labels ▾`, `📅 Milestones ▾`, `👤 Assignees ▾`, `📖 Epics ▾`, `📄 Releases ▾`, `🕒 Estimates ▾`, and `👤 Authors ▾`. A search bar on the right contains the text `🔍 Find Issues (f+i)`.

The main area is a Kanban board with four columns: `Backlog`, `In Progress`, `Review/QA`, and `Done`. Each column header shows `0 Issues - 0 Story Points`. The `In Progress` column contains one issue card: `DjangoDeKanban #1` with the title `Daphneを構成する`. The `Backlog`, `Review/QA`, and `Done` columns are currently empty.

カンバンっていいよね

Djangoでカンバン つくりたい

**Djangoでカンバン
つくるう！**

カンバンに必要な機能

ドラッグアンドドロップでの
直感的な操作

リアルタイムなデータの
反映

複数人での
コラボレーション

カンバンに必要な機能

ドラッグアンドドロップでの
直感的な操作



クライアントサイドの
JS頑張る

リアルタイムなデータの
反映



WebSocketで
リアルタイム処理をする



複数人での
コラボレーション



カンバンに必要な機能

ドラッグアンドドロップでの
直感的な操作



クライアントサイドの
JS頑張る

リアルタイムなデータの
反映



django



WebSocketで
リアルタイム処理をする

複数人での
コラボレーション



カンバンに必要な機能

ドラッグアンドドロップでの
直感的な操作



クライアントサイドの
JS頑張る

リアルタイムなデータの
反映



WebSocketで
リアルタイム処理をする

複数人での
コラボレーション



DjangoでWebSocketの 何が辛いのか

Documentation

WSGI とともにデプロイするには

Django の主要なデプロイプラットフォームは、Web サーバと Web アプリケーションに関して Python の標準である **WSGI** です。

Django の **startproject** 管理用コマンドはシンプルなデフォルトの WSGI 設定をセットアップします。必要に応じて、あなたのプロジェクトと WSGI 準拠の Web サーバに合わせて微調整することができます。

Django には以下の WSGI サーバのために、手引きとなるドキュメントが用意されています:

- [Django を Apache と `mod_wsgi` とともに使うには?](#)
- [Django のユーザーデータベースに対する Apache からの認証](#)
- [How to use Django with Gunicorn](#)
- [How to use Django with uWSGI](#)

WSGI とともにデプロイするには

Django の主要なデプロイプラットフォームは、Web サーバと Web アプリケーションに関して Python の標準である **WSGI** です。

Django の [startproject](#) 管理用コマンドはシンプルなデフォルトの WSGI 設定をセットアップします。必要に応じて、あなたのプロジェクトと WSGI 準拠の Web サーバに合わせて微調整することができます。

Django には以下の WSGI サーバのために、手引きとなるドキュメントが用意されています:

- [Django を Apache と `mod_wsgi` とともに使うには?](#)
- [Django のユーザーデータベースに対する Apache からの認証](#)
- [How to use Django with Gunicorn](#)
- [How to use Django with uWSGI](#)

Djangoは普通WSGIで動かす

WSGI?

- アプリケーションとフレームワーク間の規格
- PEP3333(Python2)/ PEP3333(Python3)

Web Server A

Framework A

Web Server B

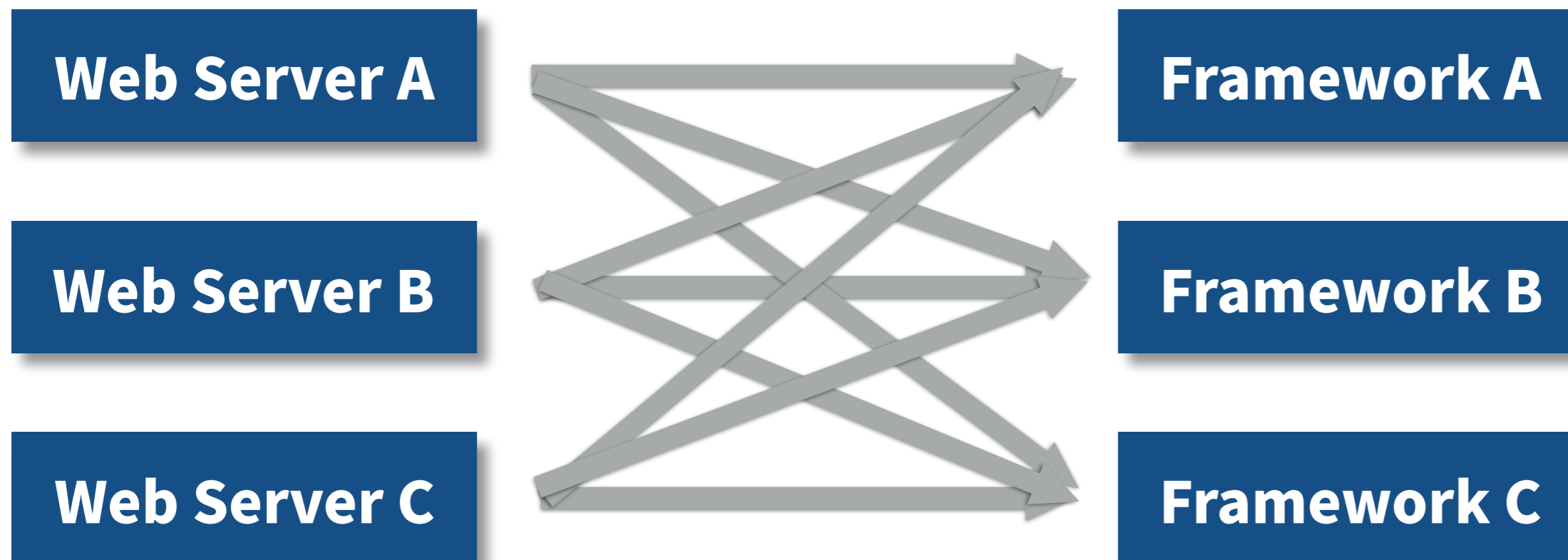
Framework B

Web Server C

Framework C

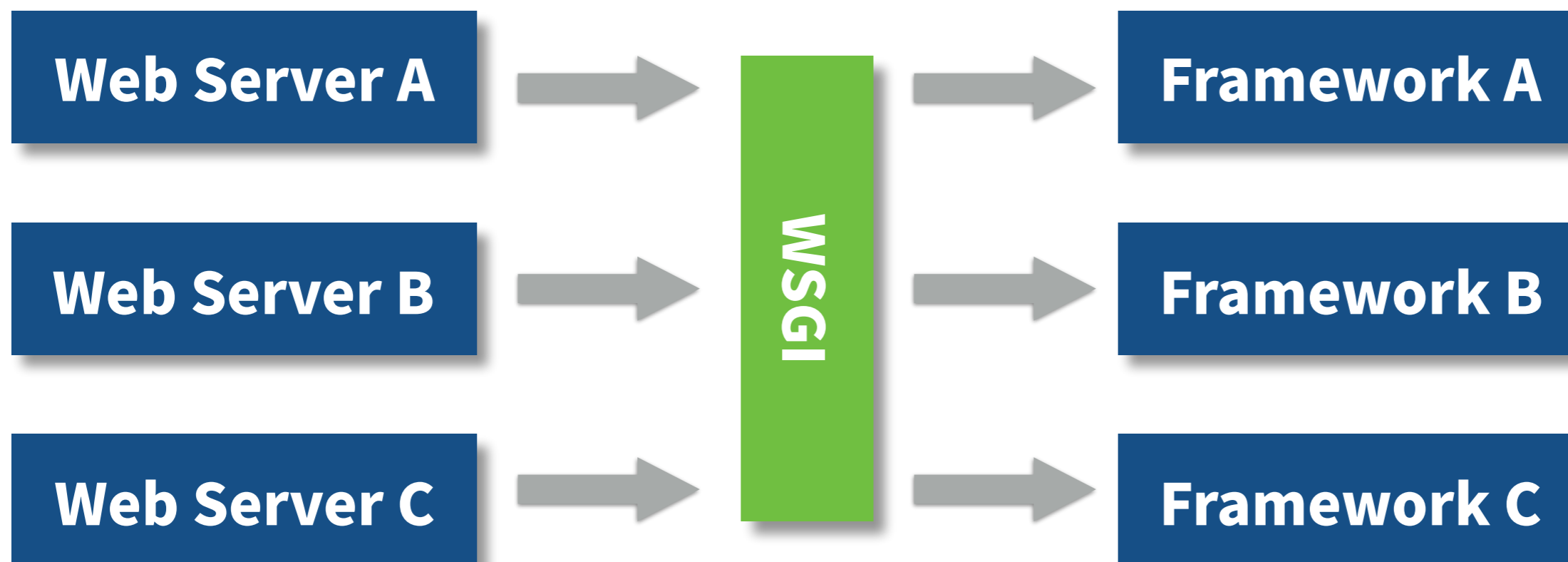
WSGI?

- アプリケーションとフレームワーク間の規格
- PEP3333(Python2)/ PEP3333(Python3)



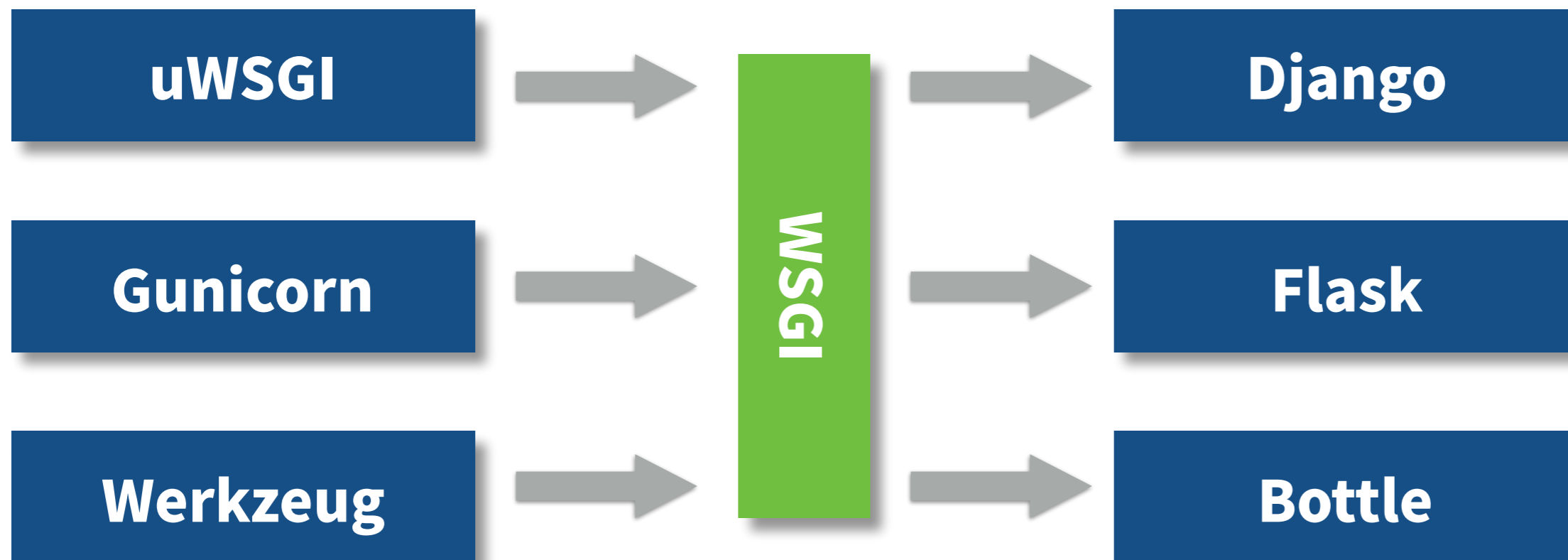
WSGI?

- アプリケーションとフレームワーク間の規格
- PEP3333(Python2)/ PEP3333(Python3)



WSGI?

- アプリケーションとフレームワーク間の規格
- PEP3333(Python2)/ PEP3333(Python3)



PEP 333の提案日は2003-12-07

PEP 333 -- Python Web Server Gateway Interface v1.0

PEP:	333
Title:	Python Web Server Gateway Interface v1.0
Author:	Phillip J. Eby <pje at telecommunity.com>
Discussions-To:	Python Web-SIG < web-sig at python.org >
Status:	Final
Type:	Informational
Created:	07-Dec-2003 
Post-History:	07-Dec-2003, 08-Aug-2004, 20-Aug-2004, 27-Aug-2004, 27-Sep-2010
Superseded-By:	3333

<https://www.python.org/dev/peps/pep-0333/>

websocketが生まれたのは2011頃

歴史的経緯 [編集]

プロトコルは2011年5月の完成を目標に進められていたが^[4]、その期日を過ぎても仕様の改訂は続けられ、2011年7月11日に最終草案のdraft-ietf-hybi-thewebsocketprotocol-10が勧告^[5]されたが、さらにその後も改訂は続き、2011年9月30日に draft-ietf-hybi-thewebsocketprotocol-17がリリースされ、それが2011年12月11日にRFC 6455のproposed standard（標準化への提唱）となった。

2010年11月26日にdraft-ietf-hybi-thewebsocketprotocol-03やそれ以前のWebSocketのプロトコルにセキュリティホールが発見され^[6]、2010年12月に、一時的に、Firefox 4とOpera 11のWebSocketが無効になり、Chromeはプロトコル改訂よりも先に攻撃コードが出た場合は無効にするとしていた。Opera 11は `opera:config#Enable%20WebSockets` を開き、設定を有効にすると利用可能。その後、2011年1月11日にdraft-ietf-hybi-thewebsocketprotocol-04が発表され、サーバにアップロード通信する際はプロキシを混乱させないために、通信内容をXORでマスキングさせる方法となった。2011年8月16日に再度WebSocketに対応させた、Firefox 6 がリリースされたが、まだ、仕様の改訂が続くという理由から、Firefox 10までは、MozWebSocketと頭にMozがつく形となった^[7]。Firefox for Mobileは7から対応^[8]。

ウェブブラウザで動作するJavaScript用のAPIの策定は、当初W3Cで行われていた。その後、HTML5とともにWHATWGに移ることとなった。

<https://ja.wikipedia.org/wiki/WebSocket>

NO WebSocket



WSGIはWebSocketより前に生まれた

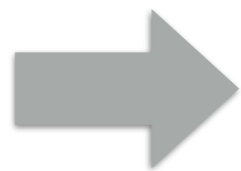
- サーバがwsgiのエンドポイントのcallableを呼び出す
- Callableはenvironとstart_responseを受け取る
- start_responseでステータスやヘッダを送信
- レスポンス本体をiterableな戻り値として返送

```
def application(environ, start_response):  
    start_response('200 OK', [('Content-Type', 'text/plain')])  
    return [b'Hello World\n']
```

WSGIはWebSocketより前に生まれた

- サーバがwsgiのエンドポイントのcallableを呼び出す
- Callableはenvironとstart_responseを受け取る
- start_responseでステータスやヘッダを送信
- レスポンス本体をiterableな戻り値として返送

```
def application(environ, start_response):  
    start_response('200 OK', [('Content-Type', 'text/plain')])  
    return [b'Hello World\n']
```



**1リクエスト毎にCallableが呼ばれ
終了する**

WebSocketのライフサイクル

- 初回接続時にセッションを確立
- セッションは切断処理がされるまで確立されたまま
- 確立したセッションでメッセージの送受信をする
- サーバからもプッシュ形式でメッセージが届く

WebSocketのライフサイクル

- 初回接続時にセッションを確立
- セッションは切断処理がされるまで確立されたまま
- 確立したセッションでメッセージの送受信をする
- サーバからもプッシュ形式でメッセージが届く

**WebSocket の
ライフサイクルは長い**

WSGIでWebsocket つらい 😇

ASGI誕生

Asynchronous Server Gateway Interface

<https://asgi.readthedocs.io/en/latest/introduction.html>

What's wrong with WSGI?

You may ask “why not just upgrade WSGI”? This has been asked many times over the years, and the problem usually ends up being that WSGI's single-callable interface just isn't suitable for more involved Web protocols like WebSocket.

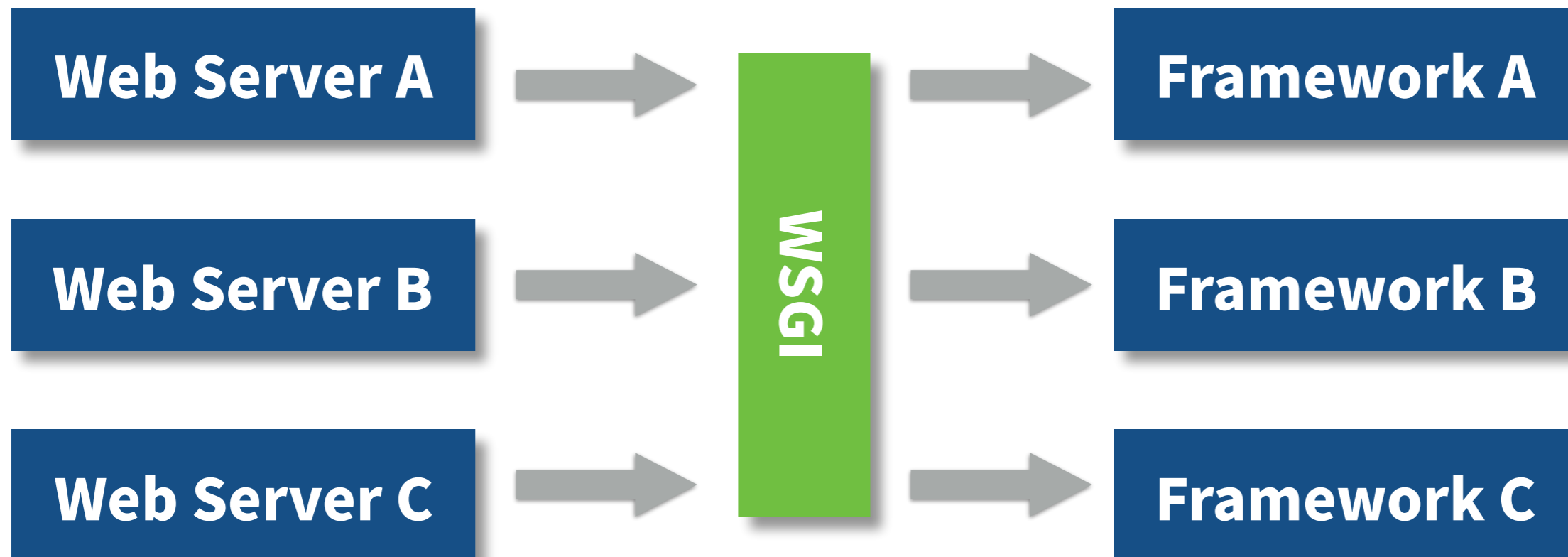
WSGI applications are a single, synchronous callable that takes a request and returns a response; this doesn't allow for long-lived connections, like you get with long-poll HTTP or WebSocket connections.

Even if we made this callable asynchronous, it still only has a single path to provide a request, so protocols that have multiple incoming events (like receiving WebSocket frames) can't trigger this.

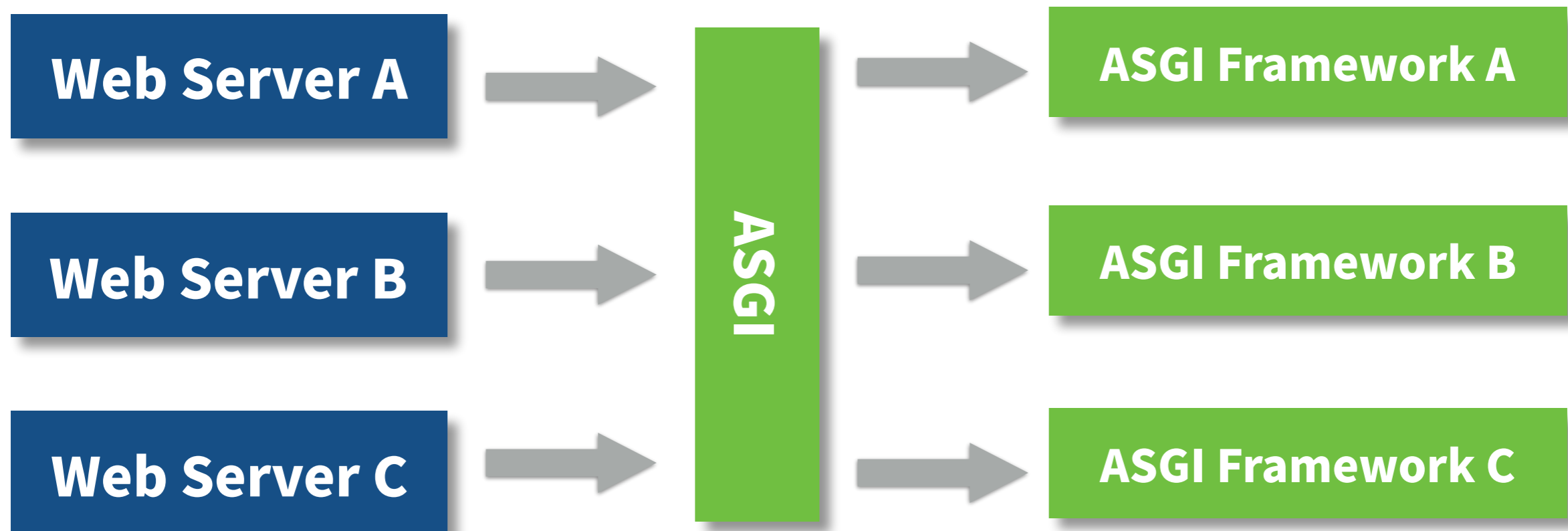
- **WSGIで対応が難しいWebSocket等のための規格**
- **WSGIのスーパーセットになるようにする**

ASGI?

- WSGIで対応が難しいWebSocket等のための規格
- WSGIのスーパーセットになるようにする



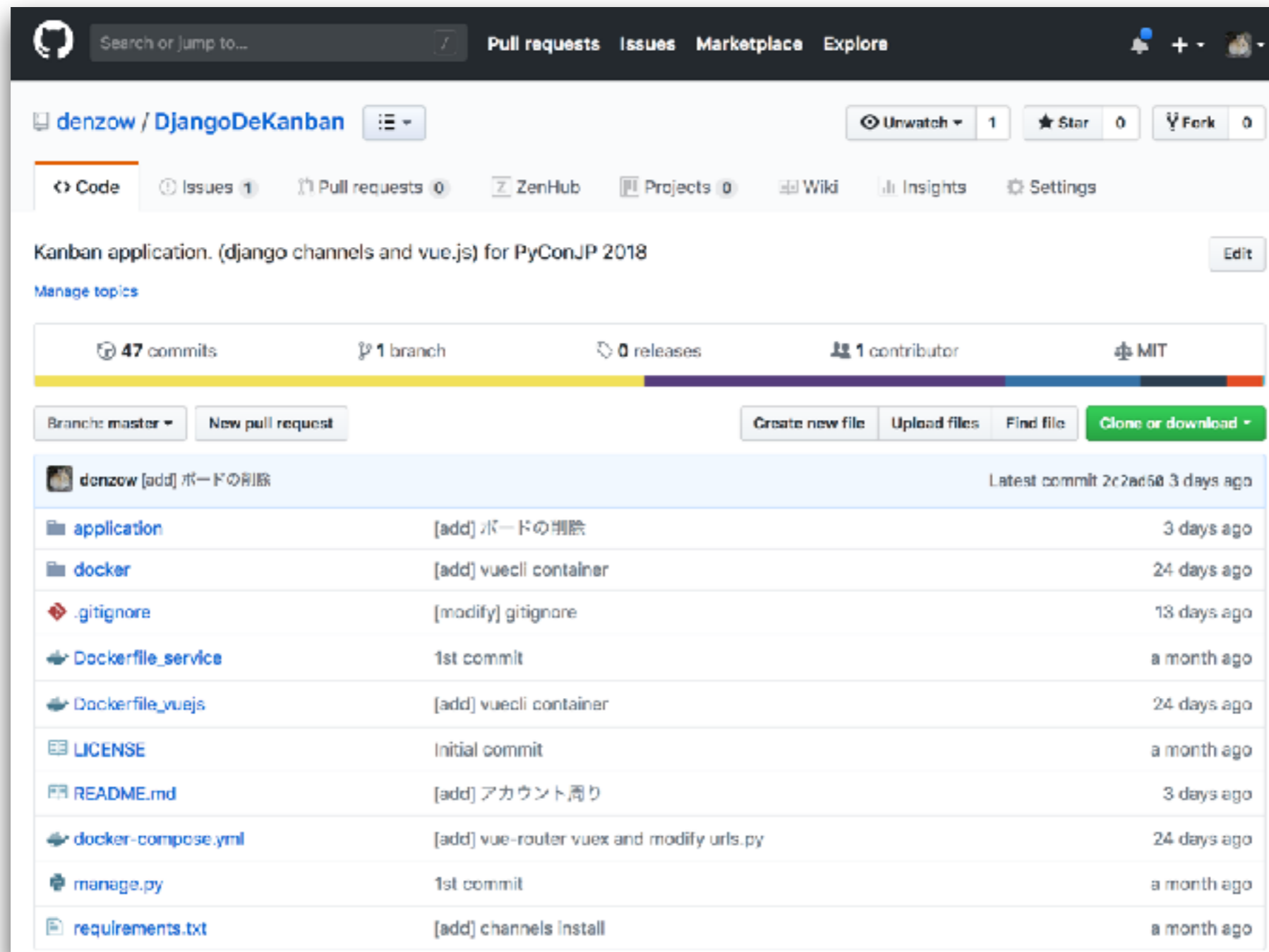
- WSGIで対応が難しいWebSocket等のための規格
- WSGIのスーパーセットになるようにする



ひろがる

Python

実際にカンバン
つくって見た



denzow / DjangoDeKanban

Unwatch 1 Star 0 Fork 0

Code Issues 1 Pull requests 0 ZenHub Projects 0 Wiki Insights Settings

Kanban application. (django channels and vue.js) for PyConJP 2018

Manage topics

47 commits 1 branch 0 releases 1 contributor MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

Commit	Message	Time
denzow [add]	ボードの削除	Latest commit 2c2ad50 3 days ago
application	[add] ボードの削除	3 days ago
docker	[add] vuecli container	24 days ago
.gitignore	[modify] gitignore	13 days ago
Dockerfile_service	1st commit	a month ago
Dockerfile_vuejs	[add] vuecli container	24 days ago
LICENSE	Initial commit	a month ago
README.md	[add] アカウント周り	3 days ago
docker-compose.yml	[add] vue-router vuex and modify urls.py	24 days ago
manage.py	1st commit	a month ago
requirements.txt	[add] channels install	a month ago

<https://github.com/denzow/DjangoDeKanban>

The screenshot shows a web application interface for a Kanban board. At the top left, the word "KANBAN" is displayed. At the top right, there is a user greeting "Welcome to denzow" and a "Logout" button. Below this is a dark header bar for the current board, "PyConJP", which includes a search input field and a red "delete" button. The main area is divided into four columns: "TODO (-)", "DOING (-)", "DONE (-)", and "Add List(+)". Each of the first three columns has an "add card" button. The "TODO" column contains two cards: "早く終わって気楽になりたい" and "参加ブログ書く". The "DOING" column contains two cards: "自分の発表" and "緊張した面持ち". The "DONE" column contains three cards: "朝ごはん", "遅刻しない", and "十分な睡眠".

KANBAN

Welcome to denzow Logout

PyConJP Search delete

TODO (-) DOING (-) DONE (-) Add List(+)

add card add card add card

早く終わって気楽になりたい
参加ブログ書く

自分の発表
緊張した面持ち

朝ごはん
遅刻しない
十分な睡眠

- **ログイン管理**
- **ボード作成**
- **パイプライン(リスト)追加**
- **パイプライン(リスト)の並び替え**
- **カード追加**
- **カード並び替え**
- **キーワードでのカード絞り込み**

- **Django 2.1**
- **Django Channels 2.1**

- **Vue/Vuex**
- **VueNativeWebSocket**
- **VueDraggable**



- **Django 2.1**
- **Django Channels 2.1**

- **Vue/Vuex**
- **VueNativeWebSocket**
- **VueDraggable**

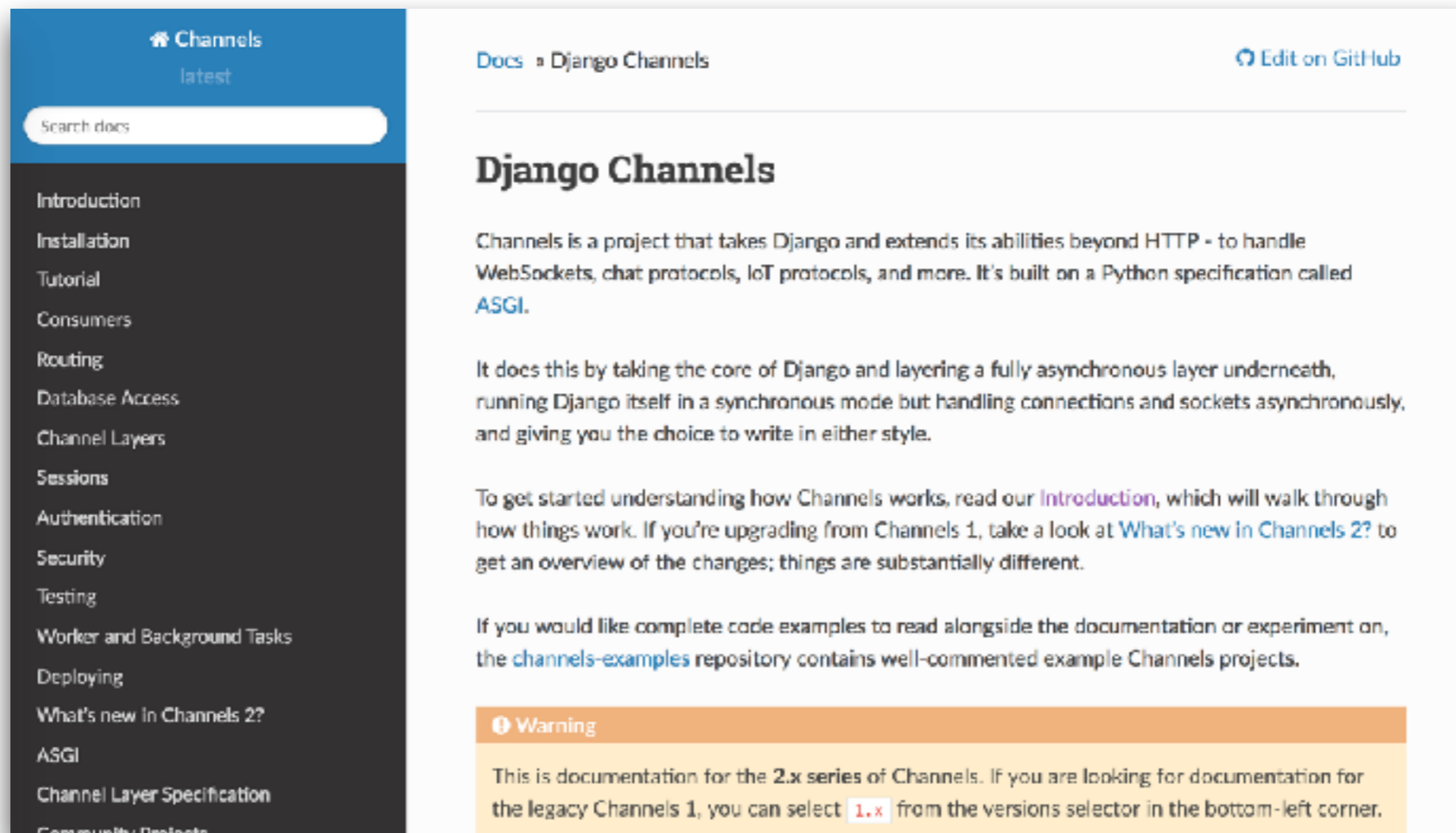


Django Channels 2

Django Channels 1.xの話はしません

2.xと1.xはPython 2とPython 3くらい違います

Django Channelsは DjangoをASGI対応にするライブラリ



The screenshot shows the Django Channels documentation page. On the left is a dark sidebar with a blue header containing the Channels logo and the word 'latest'. Below the header is a search bar and a list of navigation links including Introduction, Installation, Tutorial, Consumers, Routing, Database Access, Channel Layers, Sessions, Authentication, Security, Testing, Worker and Background Tasks, Deploying, What's new in Channels 2?, ASGI, Channel Layer Specification, and Community Projects. The main content area has a white background with a blue header containing 'Docs' and 'Django Channels', and a link to 'Edit on GitHub'. The main heading is 'Django Channels'. The text describes Channels as a project that extends Django's capabilities to handle WebSockets, chat protocols, IoT protocols, and more, built on the ASGI specification. It explains that Channels layers a fully asynchronous layer over Django's synchronous core. It also provides links to the Introduction and What's new in Channels 2? pages. A warning box at the bottom states that the documentation is for the 2.x series and provides instructions for finding legacy Channels 1 documentation.

Channels

latest

Search docs

Introduction

Installation

Tutorial

Consumers

Routing

Database Access

Channel Layers

Sessions

Authentication

Security

Testing

Worker and Background Tasks

Deploying

What's new in Channels 2?

ASGI

Channel Layer Specification

Community Projects

Docs » Django Channels

Edit on GitHub

Django Channels

Channels is a project that takes Django and extends its abilities beyond HTTP - to handle WebSockets, chat protocols, IoT protocols, and more. It's built on a Python specification called [ASGI](#).

It does this by taking the core of Django and layering a fully asynchronous layer underneath, running Django itself in a synchronous mode but handling connections and sockets asynchronously, and giving you the choice to write in either style.

To get started understanding how Channels works, read our [Introduction](#), which will walk through how things work. If you're upgrading from Channels 1, take a look at [What's new in Channels 2?](#) to get an overview of the changes; things are substantially different.

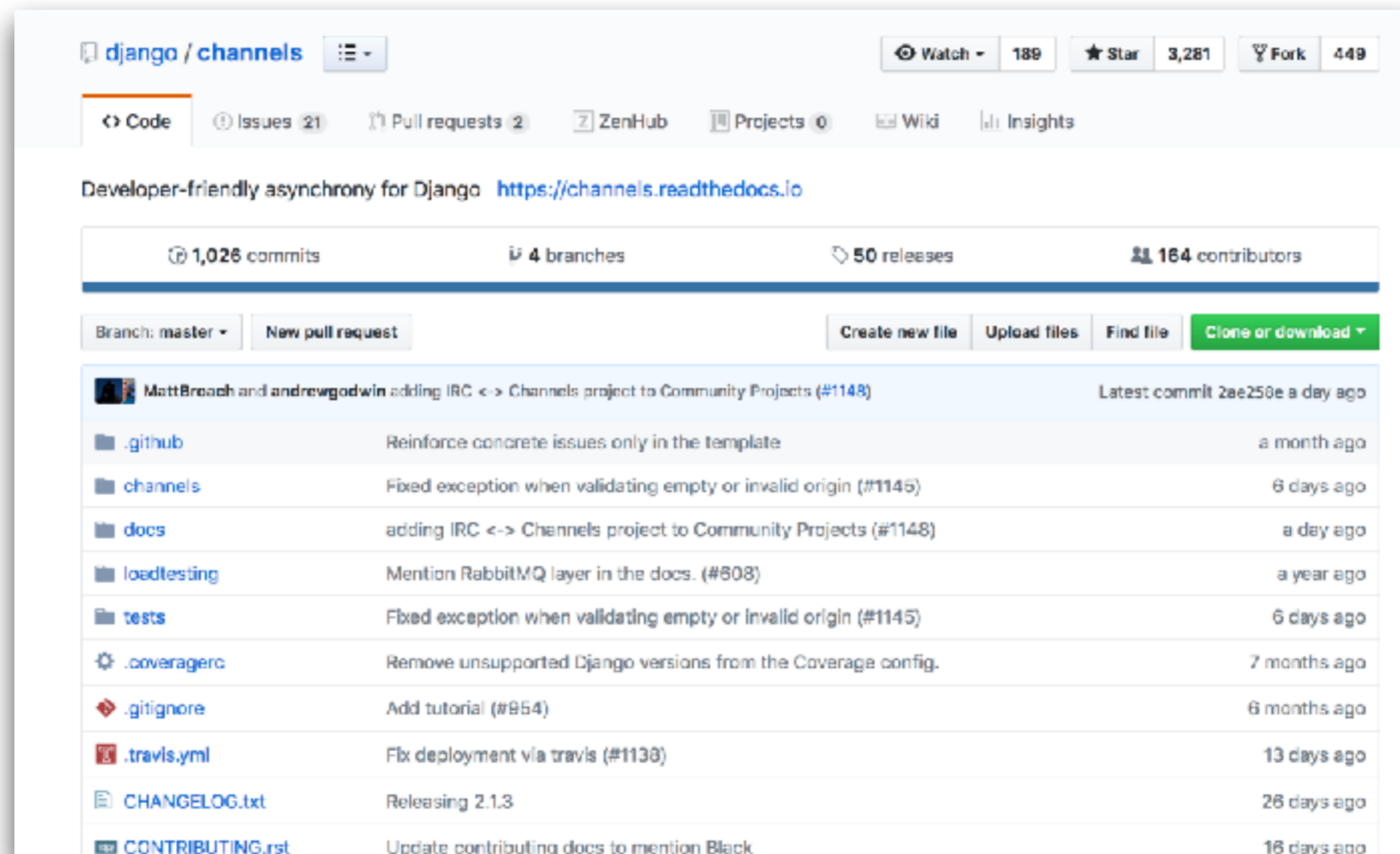
If you would like complete code examples to read alongside the documentation or experiment on, the [channels-examples](#) repository contains well-commented example Channels projects.

Warning

This is documentation for the 2.x series of Channels. If you are looking for documentation for the legacy Channels 1, you can select [1.x](#) from the versions selector in the bottom-left corner.

<https://channels.readthedocs.io/en/latest/>

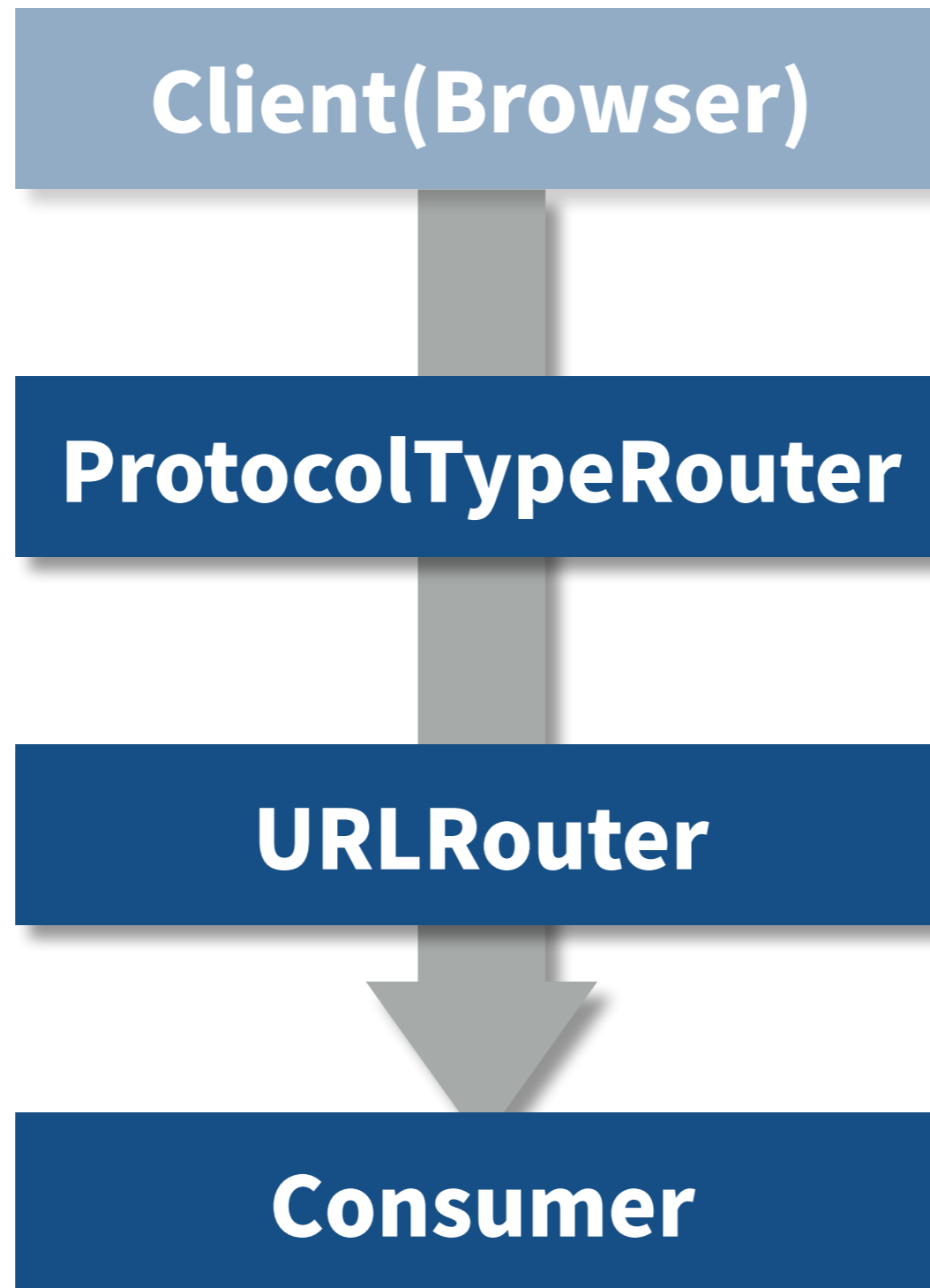
Djangoグループが開発している



The screenshot shows the GitHub repository page for Django Channels. At the top, it displays the repository name 'django / channels' with navigation icons for Watch (189), Star (3,281), and Fork (449). Below this, there are tabs for Code, Issues (21), Pull requests (2), ZenHub, Projects (0), Wiki, and Insights. The repository description is 'Developer-friendly asynchrony for Django' with a link to the documentation. A summary bar shows 1,026 commits, 4 branches, 50 releases, and 164 contributors. Below the summary, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The main content area shows a list of recent commits, including one by Matt Broach and Andrew Godwin adding IRC support to the community projects.

Commit	Description	Time
MattBroach and andrewgodwin	adding IRC <-> Channels project to Community Projects (#1148)	Latest commit 2ae250e a day ago
	.github Reinforce concrete issues only in the template	a month ago
	channels Fixed exception when validating empty or invalid origin (#1145)	6 days ago
	docs adding IRC <-> Channels project to Community Projects (#1148)	a day ago
	loadtesting Mention RabbitMQ layer in the docs. (#808)	a year ago
	tests Fixed exception when validating empty or invalid origin (#1145)	6 days ago
	.coveragerc Remove unsupported Django versions from the Coverage config.	7 months ago
	.gitignore Add tutorial (#854)	6 months ago
	.travis.yml Fix deployment via travis (#1138)	13 days ago
	CHANGELOG.txt Releasing 2.1.3	28 days ago
	CONTRIBUTING.rst Update contributing docs to mention Black	16 days ago

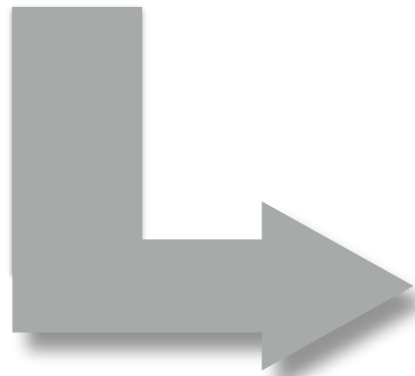
<https://github.com/django/channels>



```
application/settings/base.py
```

```
# ASGIの起点を指定
```

```
ASGI_APPLICATION = 'views.routing.application'
```



```
application/views/routing.py
```

```
from channels.auth import AuthMiddlewareStack
```

```
from channels.routing import ProtocolTypeRouter, URLRouter
```

```
# Childのルーティングルールに分割
```

```
from .ws.routing import urlpatterns
```

```
application = ProtocolTypeRouter({
```

```
# (http->django views is added by default)
```

```
'websocket': AuthMiddlewareStack(
```

```
    URLRouter(  
        urlpatterns
```

```
)
```

```
),
```

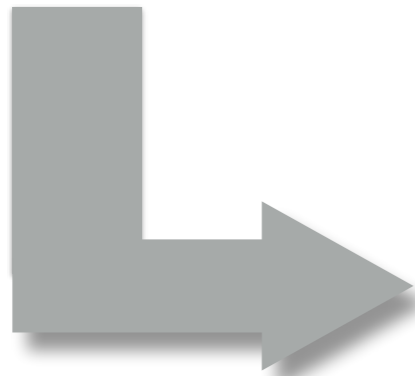
```
})
```

プロトコル毎の
処理振り分け

```
application/settings/base.py
```

```
# ASGIの起点を指定
```

```
ASGI_APPLICATION = 'views.routing.application'
```



```
application/views/routing.py
```

```
from channels.auth import AuthMiddlewareStack
```

```
from channels.routing import ProtocolTypeRouter, URLRouter
```

```
# Childのルーティングルールに分割
```

```
from .ws.routing import urlpatterns
```

```
application = ProtocolTypeRouter({
```

```
# (http->django views is added by default)
```

```
'websocket': AuthMiddlewareStack(
```

```
    URLRouter(
```

```
        urlpatterns
```

```
    )
```

```
),
```

```
})
```

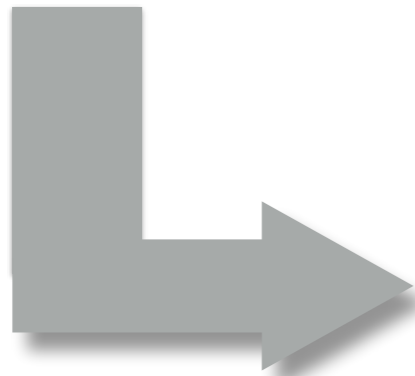
httpは書かなくても
勝手にurls.pyをもとに
したものが追加される

Websocketの場合

```
application/settings/base.py
```

```
# ASGIの起点を指定
```

```
ASGI_APPLICATION = 'views.routing.application'
```



```
application/views/routing.py
```

```
from channels.auth import AuthMiddlewareStack
```

```
from channels.routing import ProtocolTypeRouter, URLRouter
```

```
# Childのルーティングルールに分割
```

```
from .ws.routing import urlpatterns
```

```
application = ProtocolTypeRouter({  
    # (http->django views is added by default)
```

```
    'websocket': AuthMiddlewareStack(  
        URLRouter(  
            urlpatterns  
        )  
    ),  
})
```

Djangoの認証と
同じものを使えるように
する

```
application/settings/base.py
```

```
# ASGIの起点を指定
```

```
ASGI_APPLICATION = 'views.routing.application'
```

```
application/views/routing.py
```

```
from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter
```

```
# Childのルーティングルールに分割
from .ws.routing import urlpatterns
```

```
application = ProtocolTypeRouter({
    # (http->django views is added by
    'websocket': AuthMiddlewareSta
```

```
    URLRouter(
        urlpatterns
    )
),
})
```

どのようなURLにアクセス
してきたかでの振り分け
(別に直接書いてもいい)

基本的にはDjangoのurls.pyと同じ

```
application/views/ws/routing.py

from django.urls import path
from .consumers import kanban_consumer

urlpatterns = [
    path('ws/boards/<int:board_id>', kanban_consumer.KanbanConsumer)
]
```


基本的にはDjangoのurls.pyと同じ

```
application/views/ws/routing.py

from django.urls import path
from .consumers import kanban_consumer

urlpatterns = [
    path('ws/boards/<int:board_id>', kanban_consumer.KanbanConsumer)
]
```

ws/boards/1 のような
URLにマッチさせる

基本的にはDjangoのurls.pyと同じ

```
application/views/ws/routing.py

from django.urls import path
from .consumers import kanban_consumer

urlpatterns = [
    path('ws/boards/<int:board_id>', kanban_consumer.KanbanConsumer)
]
```

該当したときの処理

Viewsみたいなもん

```
application/views/ws/consumers/kanban_consumer.py
:
:
class KanbanConsumer(BaseJsonConsumer):
:
    async def connect(self):
        if not self.scope['user'].is_authenticated:
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
:
```

Viewsみたいなもん

```
application/views/ws/consumers/kanban_consumer.py
```

```
:
```

```
:
```

```
class KanbanConsumer(BaseJsonConsumer):
```

```
:
```

```
async def connect(self):
```

```
    if not self.scope['user'].is_authenticated:
```

```
        await self.close()
```

```
        return
```

```
    self.user = self.scope['user']
```

```
    self.board_id = self.scope['url_route']['kwargs']['board_id']
```

```
    self.room_group_name = self.user.username
```

```
    await self.channel_layer.group_add(
```

```
        self.room_group_name,
```

```
        self.channel_name
```

```
    )
```

```
    await self.accept()
```

```
:
```

AuthMiddlewareStackを
使くと、scope['user']に
いれてくれる

Viewsみたいなもん

```
application/views/ws/consumers/kanban_consumer.py
:
:
class KanbanConsumer(BaseJsonConsumer):
:
    async def connect(self):
        if not self.scope['user'].is_authenticated:
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
:
```

URLで <int:board_id> と
定義してた部分にマッチした情報が
取り出せる

```
application/views/ws/routing.py
```

```
from django.urls import path
```

```
from .consumers import kanban_consumer
```

```
urlpatterns = [
```

```
    path('ws/boards/<int:board_id>', kanban_consumer.KanbanConsumer)
```

```
]
```



これ

Viewsみたいなもん

```
application/views/ws/consumers/kanban_consumer.py
:
:
class KanbanConsumer(BaseJsonConsumer):
:
    async def connect(self):
        if not self.scope['user'].is_authenticated:
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
:
```

URLで <int:board_id> と
定義してた部分にマッチした情報が
取り出せる

Viewsみたいなもん

```
application/views/ws/consumers/kanban_consumer.py
:
:
class KanbanConsumer(BaseJsonConsumer):
:
    async def connect(self):
        if not self.scope['user'].is_authenticated:
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
:
```

ChannelLayerのgroup_addを
実行し、他のConsumerとの
通信ができるようにする

Viewsみたいなもん

```
application/views/ws/consumers/kanban_consumer.py
:
:
class KanbanConsumer(BaseJsonConsumer):
:
    async def connect(self):
        if not self.scope['user'].is_authenticated:
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
:
```

接続を拒否する

接続を受け入れる

```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):  
  
    async def connect(self):  
        # accept or close  
        if YourCondition:  
            await self.accept()  
        else:  
            await self.close()  
  
    async def receive_json(self, content, **kwargs):  
        # receive message  
        print(content)  
  
        # echo back  
        await self.send_json(content)
```

```
class MyConsumer(AsyncJsonWebSocketConsumer or JsonWebSocketConsumer)
```

```
async def connect(self):
```

```
    # accept or close
```

```
    if YourCondition:
```

```
        await self.accept()
```

```
    else:
```

```
        await self.close()
```

```
async def receive_json(self, content, **kwargs):
```

```
    # receive message
```

```
    print(content)
```

```
    # echo back
```

```
    await self.send_json(content)
```

(Async)?JsonWebSocketConsumerを
継承して実装する

```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):
```

```
    async def connect(self):  
        # accept or close  
        if YourCondition:  
            await self.accept()  
        else:  
            await self.close()
```

初回接続時に
呼ばれる
Accept or close を
呼び出す

```
    async def receive_json(self, content, **kwargs):  
        # receive message  
        print(content)  
  
        # echo back  
        await self.send_json(content)
```

```
class MyConsumer(AsyncJsonWebSocketConsumer or JsonWebSocketConsumer):
```

```
    async def connect(self):
```

```
        # accept or close
```

```
        if YourCondition:
```

```
            await self.accept()
```

```
        else:
```

```
            await self.close()
```

クライアントからの
メッセージ受信時に呼ばれる

```
    async def receive_json(self, content, **kwargs):
```

```
        # receive message
```

```
        print(content)
```

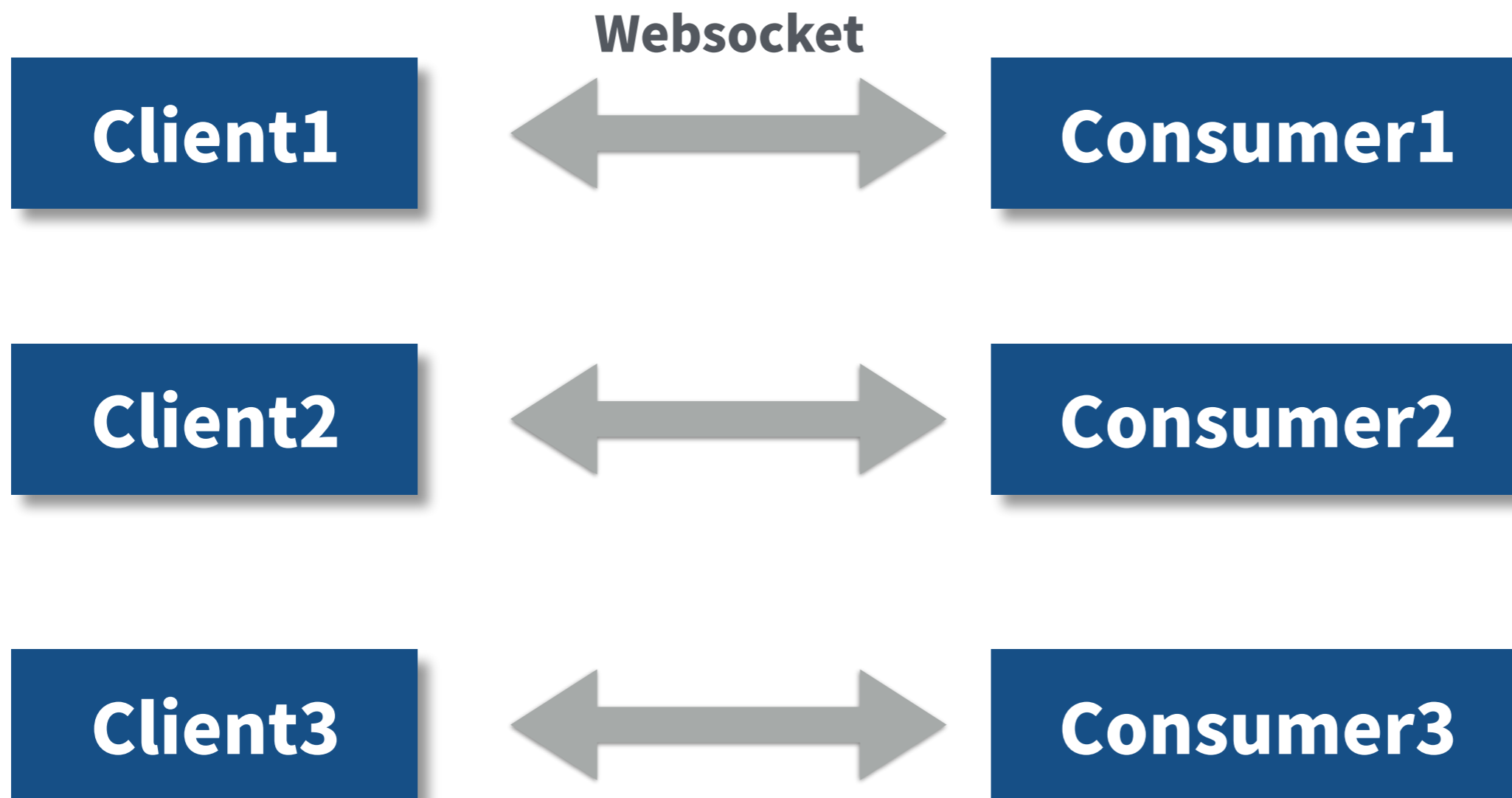
```
        # echo back
```

```
        await self.send_json(content)
```

```
class MyConsumer(AsyncJsonWebSocketConsumer or JsonWebSocketConsumer):  
  
    async def connect(self):  
        # accept or close  
        if YourCondition:  
            await self.accept()  
        else:  
            await self.close()  
  
    async def receive_json(self, content, **kwargs):  
        # receive message  
        print(content)  
  
        # echo back  
        await self.send_json(content)
```

クライアントへ
メッセージを返送する

ClientとConsumer間はWebSocketで 送受信ができるようになった



Consumer1の変更をClient2,3に伝えるには？

card_list = [A]

Client1



card_list = [A]

Consumer1

card_list = [A]

Client2



card_list = [A]

Consumer2

card_list = [A]

Client3



card_list = [A]

Consumer3

Consumer1の変更をClient2,3に伝えるには？

card_list = [A, B]

Client1

Add card B

card_list = [A, B]

Consumer1

card_list = [A]

Client2

card_list = [A]

Consumer2

card_list = [A]

Client3

card_list = [A]

Consumer3



Consumer1の変更をClient2,3に伝えるには？

card_list = [A, B]

Client1

Add card B

card_list = [A, B]

Consumer1

card_list = [A]

Client2

card_list = [A]

Consumer2

card_list = [A]

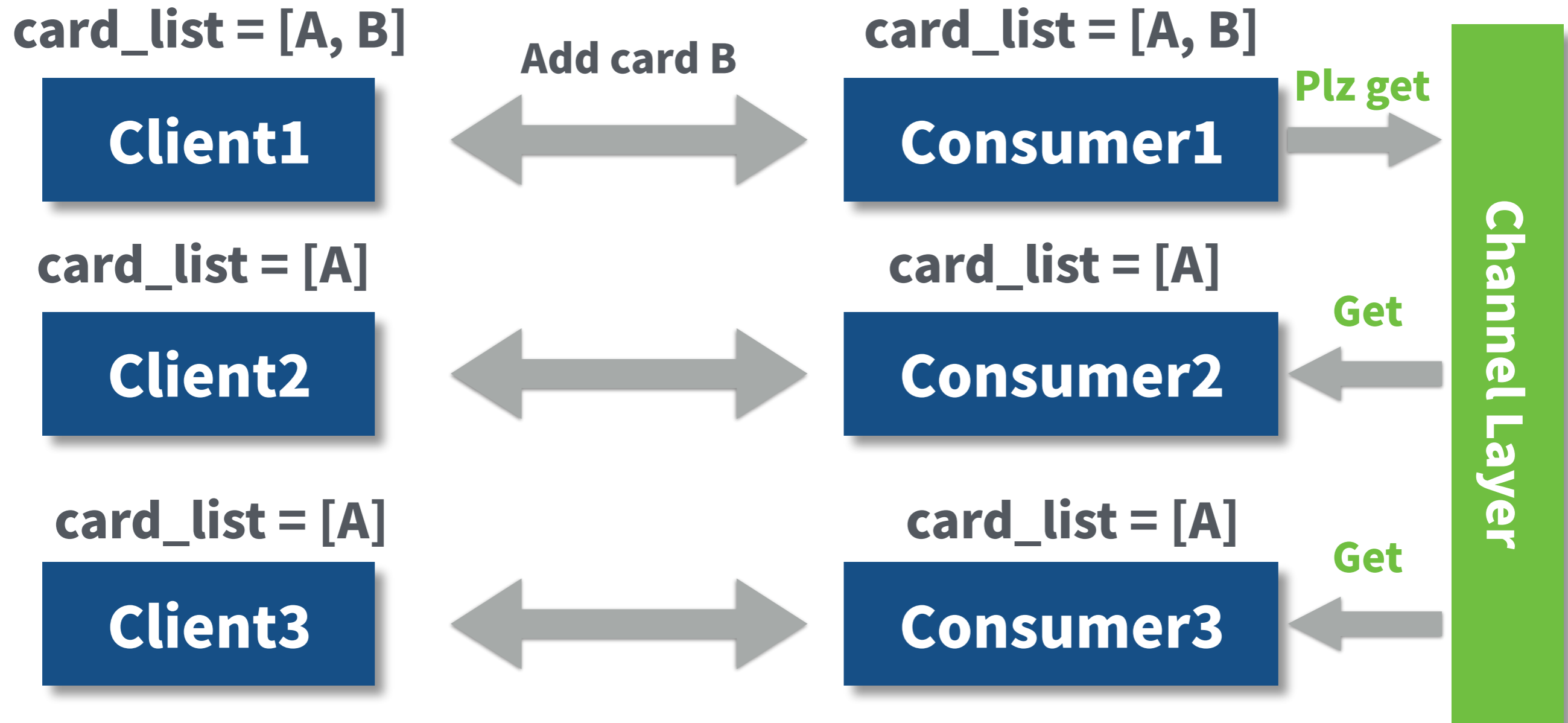
Client3

card_list = [A]

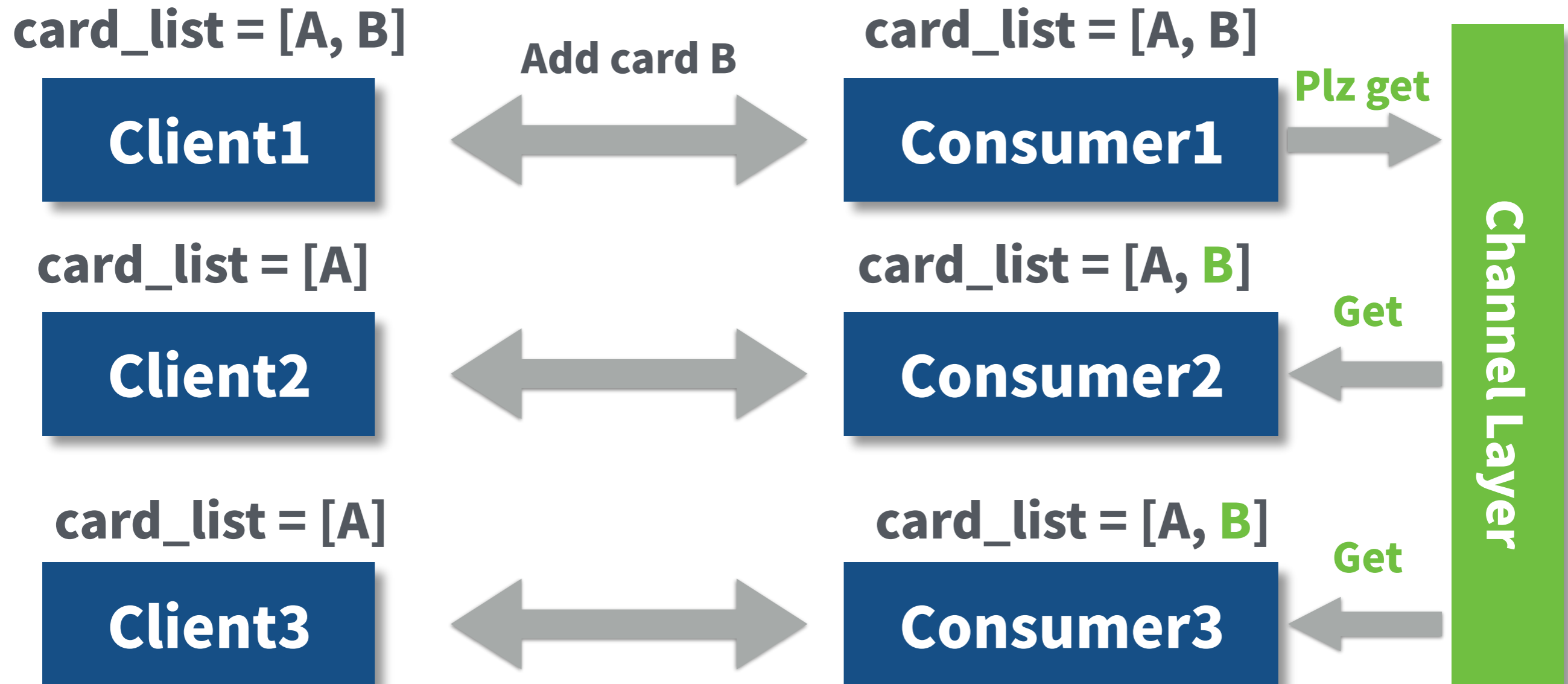
Consumer3

Channel Layer

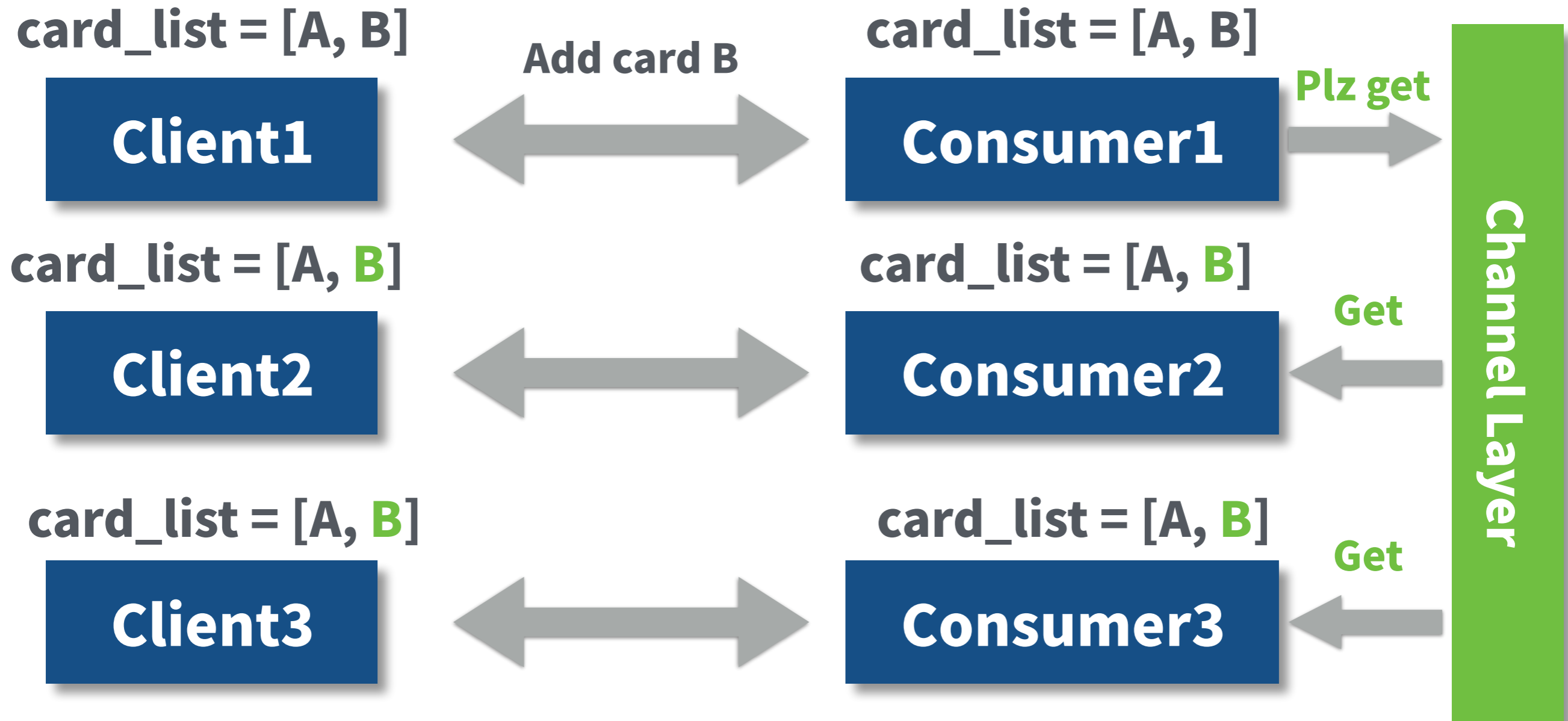
Consumer1の変更をClient2,3に伝えるには？



Consumer1の変更をClient2,3に伝えるには？



Consumer1の変更をClient2,3に伝えるには？



```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        :
        self.group_name = 'group_1'
        await self.channel_layer.group_add(
            self.group_name,
            self.channel_name # auto injection
        )

    async def receive_json(self, content, **kwargs):
        :
        # broadcast other consumers
        await self.channel_layer.group_send(
            self.group_name,
            {
                'type': 're_send',
            }
        )

    async def re_send(self, *args, **kwargs):
        await self.send_json('new_content')
```

```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):
```

```
    async def connect(self):
```

```
        :
        self.group_name = 'group_1'
        await self.channel_layer.group_add(
            self.group_name,
            self.channel_name # auto injection
        )
```

connect時に
処理内容を共有する
グループへ登録

```
    async def receive_json(self, content, **kwargs):
```

```
        :
        # broadcast other consumers
        await self.channel_layer.group_send(
            self.group_name,
            {
                'type': 're_send',
            }
        )
```

```
    async def re_send(self, *args, **kwargs):
        await self.send_json('new_content')
```

```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        :
        self.group_name = 'group_1'
        await self.channel_layer.group_add(
            self.group_name,
            self.channel_name # auto injection
        )

    async def receive_json(self, content, **kwargs):
        :
        # broadcast other consumers
        await self.channel_layer.group_send(
            self.group_name,
            {
                'type': 're_send',
            }
        )

    async def re_send(self, *args, **kwargs):
        await self.send_json('new_content')
```

group_sendで
ChannelLayerを
通じて他のConsumerに
リクエストする


```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        :
        self.group_name = 'group_1'
        await self.channel_layer.group_add(
            self.group_name,
            self.channel_name # auto injection
        )

    async def receive_json(self, content, **kwargs):
        :
        # broadcast other consumers
        await self.channel_layer.group_send(
            self.group_name,
            {
                'type': 're_send',
            }
        )
```

typeで指定したメソッドが
各Consumerで実行される

```
async def re_send(self, *args, **kwargs):
    await self.send_json('new_content')
```

```
application/views/ws/consumers/kanban_consumer.py
```

```
def __init__(self, *args, **kwargs):
    :
    self.action_map = {
        'update_card_order': self.update_card_order,
        'update_pipe_line_order': self.update_pipe_line_order,
        'add_pipe_line': self.add_pipe_line,
        'add_card': self.add_card,
        'rename_pipe_line': self.rename_pipe_line,
        'delete_pipe_line': self.delete_pipe_line,
        'delete_board': self.delete_board,
        'rename_board': self.rename_board,
        'broadcast_board_data': self.broadcast_board_data,
        'broadcast_board_data_without_requester': self.broadcast_board_data_without_requester,
    }
```

```
application/views/ws/consumers/base_consumer.py
```

```
async def receive_json(self, content, **kwargs):
```

```
    """
```

```
    Typeに応じた処理を呼び出して実行する
```

```
:param dict content:
```

```
:param kwargs:
```

```
:return:
```

```
    """
```

```
    action = self.action_map.get(content['type'])
```

```
    if not action:
```

```
        raise ConsumerException('{} is not a valid action_type'.format(content['type']))
```

```
    await action(content)
```

content.typeに応じたメソッドを
呼び出すようにreceive_jsonを
オーバーライド

カンバンとVue

**Vueがカンバンに向いてるということではなく
単に私がVueでつくったので紹介だけ**



カンバンに必要な機能

ドラッグアンドドロップでの
直感的な操作



クライアントサイドの
JS頑張る



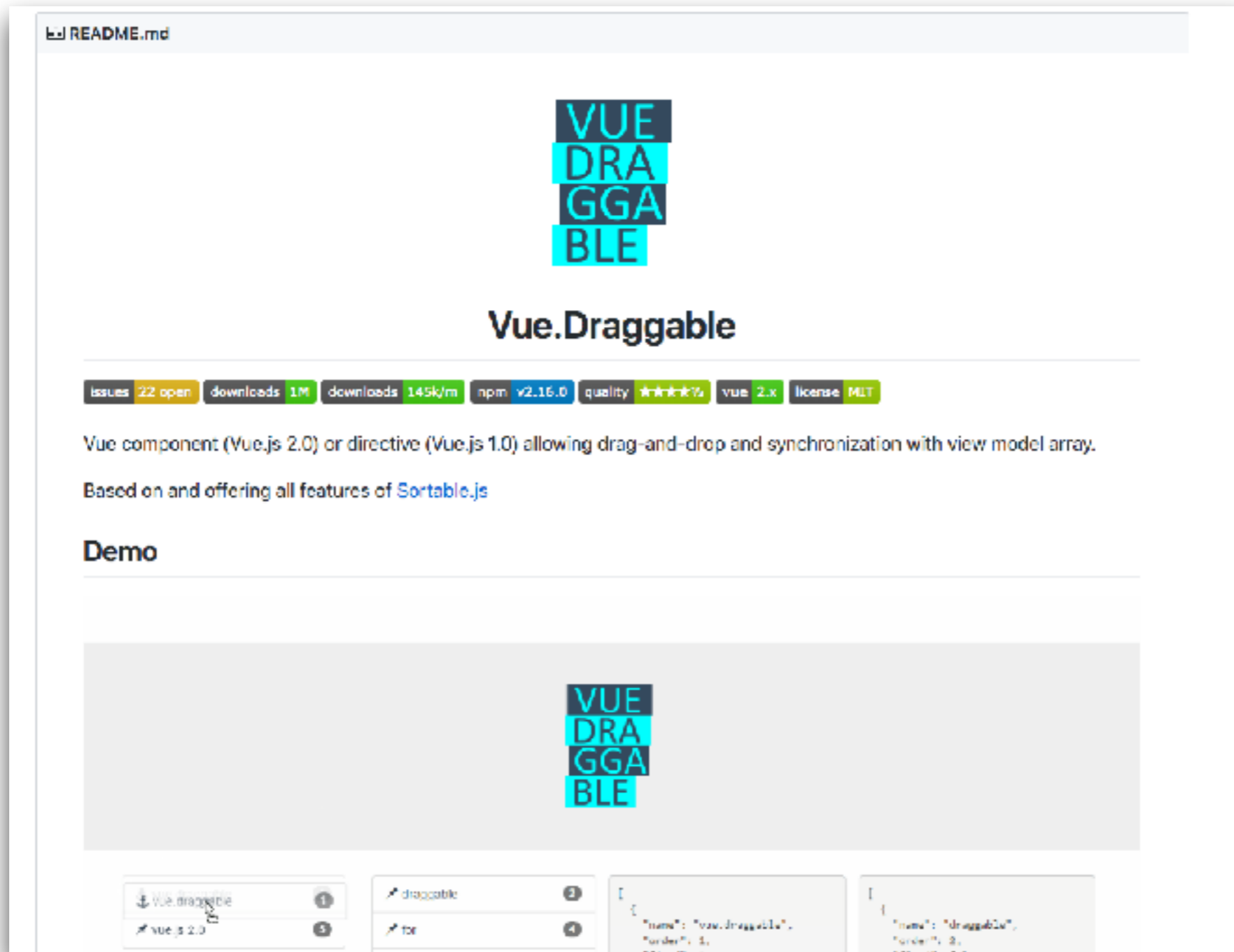
リアルタイムなデータの
反映



KANBAN

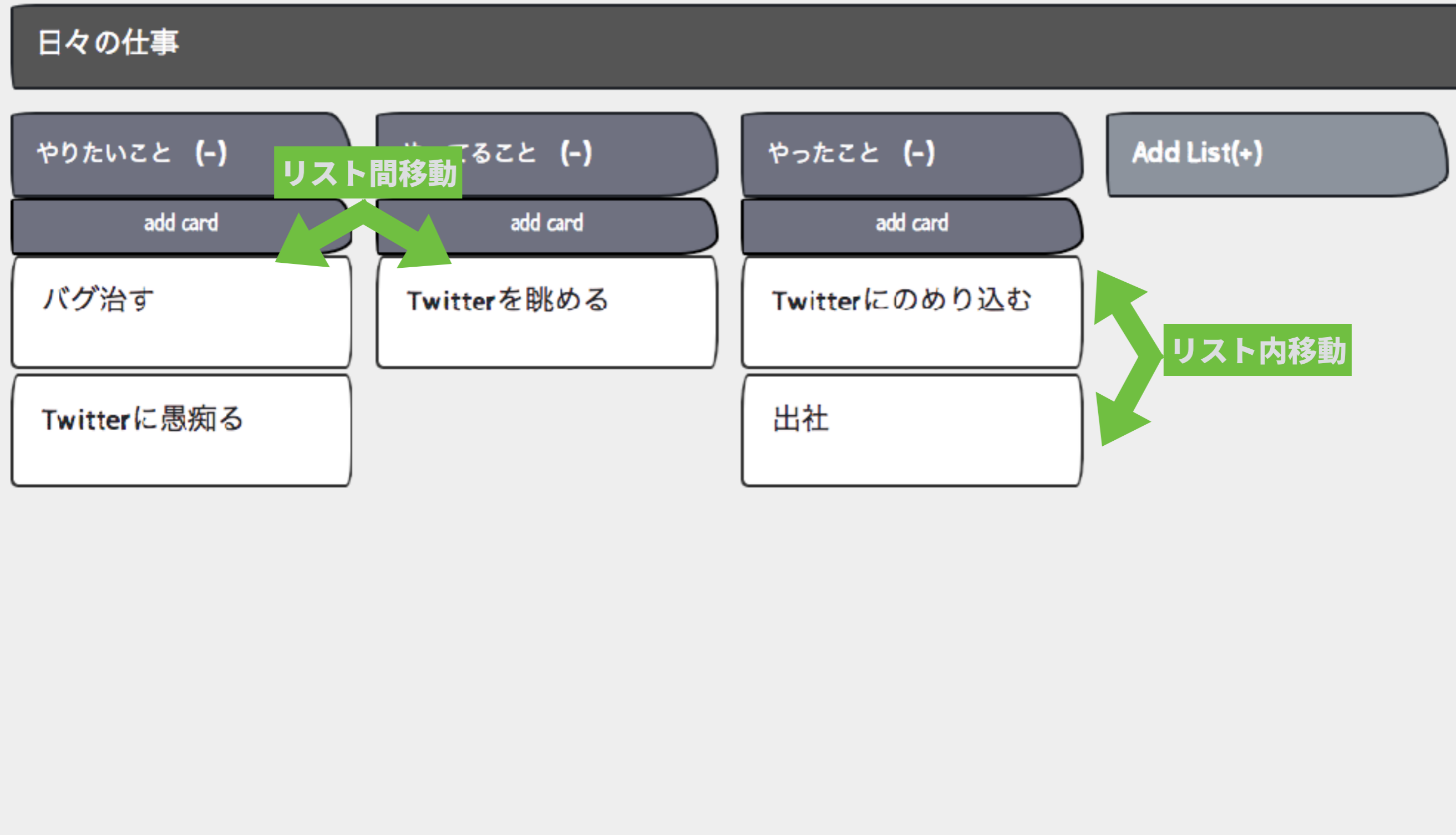
ISUCON8

やりたいこと (-)	やってること (-)	やったこと (-)	できなかったこと (-)	Add List(+)
add card	add card	add card	add card	
参加ブログまとめる		打ち上げ		
後悔		予選参加		
		決勝進出		



<https://github.com/SortableJS/Vue.Draggable>

- **Sortable.jsのラッパーで使いやすい**
- **リスト内はもちろん、リスト間のD&Dも容易**
- **D&D完了時に、並び順のデータがちゃんと取れる**



The screenshot shows the GitHub repository page for 'nathantsoi/vue-native-websocket'. At the top, the repository name is displayed with navigation icons for Unwatch (13), Unstar (244), and Fork (53). Below this, there are tabs for Code, Issues (23), Pull requests (2), ZenHub, Projects (0), Wiki, and Insights. The repository description is 'native websocket with vuex integration'. A summary bar shows 110 commits, 2 branches, 10 releases, and 24 contributors. Below the summary, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The commit history is listed below, showing the latest commit by 'weglov' (release v2.0.11) 2 days ago, and several other commits related to 'Fix custom mutations feature + test' and 'Specs (#6)'.

Commit	Message	Time
weglov	release v2.0.11	2 days ago
dist	Fix custom mutations feature + test	3 days ago
src	Fix custom mutations feature + test	3 days ago
test/unit	Fix custom mutations feature + test	3 days ago
.eslintrc.js	native websocket with json parsing, vuex integration and namespacing	a year ago
.gitignore	Specs (#6)	a year ago
.npmrc	Specs (#6)	a year ago
.travis.yml	Specs (#6)	a year ago
CHANGELOG.md	release v2.0.11	2 days ago
PUBLISH.md	add test step to PUBLISH.md	10 months ago

<https://github.com/nathantsoi/vue-native-websocket>

Commits on May 31, 2018

Update README.md ...

 justerror committed on 31 May ✓

Verified



4f9acd0



Commits on May 7, 2018

Feature handle skip scheme ws url ...

 denzow committed on 7 May ✓




c0f0209



Commits on Apr 18, 2018

update build


 weglov committed on 18 Apr ✓



649485e



Merge pull request #48 from saspallow/master ...

 weglov committed on 18 Apr ✓

Verified



45c763f



Commits on May 31, 2018

Update README.md ...

 justerror committed on 31 May ✓

Verified



4f9acd0



Commits on May 7, 2018

Feature handle skip scheme ws url ...

 denzow committed on 7 May ✓




c0f0209



Commits on Apr 18, 2018

update build


 weglov committed on 18 Apr ✓



649485e



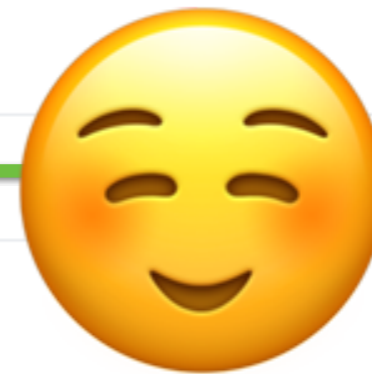
Merge pull request #48 from saspallow/master ...

 weglov committed on 18 Apr ✓

Verified



45c763f



- Vue側から簡単にWebsocketが使える
- Store(Vuex)とも連携が簡単
- Serverからmutation/actionが呼び出せる

```
Vue.use(VueNativeSock, 'ws://localhost:9090', {  
  store,  
})
```

- `sendObj`でJSONを簡単に投げられる

```
socket.sendObj({  
  type: 'add_card',  
  pipeLineId,  
  cardTitle,  
});
```

- シンプルな用途であれば使いやすくして便利
- 自動再接続や手動再接続のメソッドもある

- 接続先の変更とかはちょっとだるい

```
// 切断
Vue.prototype.$disconnect();

// InstallされているVueNativeSockを一旦削除する
const index = Vue._installedPlugins.indexOf(VueNativeSock);
if (index > -1) {
  Vue._installedPlugins.splice(index, 1);
}
// 新しいURLで再インストールする
Vue.use(VueNativeSock, `/new_ws_endpoint/`, {
  connectManually: true,
  reconnection: true,
  reconnectionAttempts: 5,
  reconnectionDelay: 3000,
  format: 'json',
  store,
});
// 新しいURLへ接続する
Vue.prototype.$connect();
```

データの流れ



```
<Draggable
  class="card-container"
  :options="options"
  v-model="wrappedCardList"
>
  <Card v-for="card in wrappedCardList"
    class="item"
    v-show="card.isShown"
    :card="card"
    :key="card.cardId"
  />
</Draggable>
:
computed: {
  wrappedCardList: {
    get() {
      return this.pipeLine.cardList;
    },
    set(value) {
      console.log(value);
      this.updateCardOrder({
        pipeLineId: this.pipeLine.pipeLineId,
        cardList: value,
      });
    },
  },
},
```

Draggableで
囲んだ要素(Card)が
D&D可能に

```
<Draggable
  class="card-container"
  :options="options"
  v-model="wrappedCardList"
>
  <Card v-for="card in wrappedCardList"
    class="item"
    v-show="card.isShown"
    :card="card"
    :key="card.cardId"
  />
</Draggable>

:
computed: {
  wrappedCardList: {
    get() {
      return this.pipeline.cardList;
    },
    set(value) {
      console.log(value);
      this.updateCardOrder({
        pipelineId: this.pipeline.pipelineId,
        cardList: value,
      });
    },
  },
},
```

D&Dの更新内容は
V-modelに同期される
作り

```
<Draggable
  class="card-container"
  :options="options"
  v-model="wrappedCardList"
>
  <Card v-for="card in wrappedCardList"
    class="item"
    v-show="card.isShown"
    :card="card"
    :key="card.cardId"
  />
</Draggable>
:
computed: {
  wrappedCardList: {
    get() {
      return this.pipeline.cardList;
    },
    set(value) {
      console.log(value);
      this.updateCardOrder({
        pipelineId: this.pipeline.pipelineId,
        cardList: value,
      });
    },
  },
},
```

D&D完了時にsetが
呼ばれる

```
[
  {cardId: 1, title: ....},
  {cardId: 2, title: ....},
  {cardId: 3, title: ....},
]
```

Vuexに処理を
依頼する

```
<Draggable
  class="card-container"
  :options="options"
  v-model="wrappedCardList"
>
  <Card v-for="card in wrappedCardList"
    class="item"
    v-show="card.isShown"
    :card="card"
    :key="card.cardId"
  />
</Draggable>
:
computed: {
  wrappedCardList: {
    get() {
      return this.pipeline.cardList;
    },
    set(value) {
      console.log(value);
      this.updateCardOrder({
        pipelineId: this.pipeline.pipelineId,
        cardList: value,
      });
    },
  },
},
```

```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {
  const socket = getters.getSocket;
  socket.sendObj({
    type: 'update_card_order',
    pipeLineId,
    cardIdList: cardList.map(x => x.cardId),
  });
  commit('updateCardOrder', { pipeLineId, cardList });
},

// MUTATION
updateCardOrder(state, { pipeLineId, cardList }) {
  const targetPipeLine = state.boardData.pipeLineList
    .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
  targetPipeLine.cardList = cardList;
},
```



```
// ACTION
```

```
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {  
  const socket = getters.getSocket;  
  socket.sendObj({  
    type: 'update_card_order',  
    pipeLineId,  
    cardIdList: cardList.map(x => x.cardId),  
  });  
  commit('updateCardOrder', { pipeLineId, cardList });  
},
```

```
// MUTATION
```

```
updateCardOrder(state, { pipeLineId, cardList }) {  
  const targetPipeLine = state.boardData.pipeLineList  
    .find(pipeLine => pipeLine.pipeLineId === pipeLineId);  
  targetPipeLine.cardList = cardList;  
},
```

VueComponentから
ここが呼ばれる。

```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {
  const socket = getters.getSocket;
  socket.sendObj({
    type: 'update_card_order',
    pipeLineId,
    cardIdList: cardList.map(x => x.cardId),
  });
  commit('updateCardOrder', { pipeLineId, cardList });
},

// MUTATION
updateCardOrder(state, { pipeLineId, cardList }) {
  const targetPipeLine = state.boardData.pipeLineList
    .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
  targetPipeLine.cardList = cardList;
},
```

VueNativeWebsocketを使って、Django側に更新を依頼(後述)

```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {
  const socket = getters.getSocket;
  socket.sendObj({
    type: 'update_card_order',
    pipeLineId,
    cardIdList: cardList.map(x => x.cardId),
  });
  commit('updateCardOrder', { pipeLineId, cardList });
}
```

投げると同時に投げた
クライアント側では更新が
行われた体で表示を更新

```
// MUTATION
updateCardOrder(state, { pipeLineId, cardList }) {
  const targetPipeLine = state.boardData.pipeLineList
    .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
  targetPipeLine.cardList = cardList;
},
```

```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {
  const socket = getters.getSocket;
  socket.sendObj({
    type: 'update_card_order',
    pipeLineId,
    cardIdList: cardList.map(x => x.cardId),
  });
  commit('updateCardOrder', { pipeLineId, cardList });
},

// MUTATION
updateCardOrder(state, { pipeLineId, cardList }) {
  const targetPipeLine = state.boardData.pipeLineList
    .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
  targetPipeLine.cardList = cardList;
},
```

```
class KanbanConsumer(BaseJsonConsumer):  
  
    def __init__(self, *args, **kwargs):  
        :  
        self.action_map = {  
            'update_card_order': self.update_card_order,  
            :  
        }  
        :  
        :
```

```
class KanbanConsumer(BaseJsonConsumer):  
  
    def __init__(self, *args, **kwargs):  
        :  
        self.action_map = {  
            'update_card_order': self.update_card_order,  
            :  
        }  
        :
```

type: 'update_card_order'は
self.update_card_orderを
呼び出しするようにマップ

```
class KanbanConsumer(BaseJsonConsumer):  
  
    async def update_card_order(self, content):  
        """  
        ボード内のカードの並び順を更新する  
        """  
        pipe_line_id = content['pipeLineId']  
        card_id_list = content['cardIdList']  
        await database_sync_to_async(kanban_sv.update_card_order)(  
            pipe_line_id,  
            card_id_list  
        )  
        await self.broadcast_board_data_without_requester()
```

```
class KanbanConsumer(BaseJsonConsumer):  
  
    async def update_card_order(self, content):  
        """  
        ボード内のカードの並び順を更新する  
        """  
        pipe_line_id = content['pipeLineId']  
        card_id_list = content['cardIdList']  
  
        await database_sync_to_async(kanban_sv.update_card_order)(  
            pipe_line_id,  
            card_id_list  
        )  
  
        await self.broadcast_board_data_without_requester()
```

DjangoORM経由でカードの
並び順を更新


```
class KanbanConsumer(BaseJsonConsumer):
```

```
    async def update_card_order(self, content):
```

```
        """
```

```
        ボード内のカードの並び順を更新する
```

```
        """
```

```
        pipe_line_id = content['pipeLineId']
```

```
        card_id_list = content['cardIdList']
```

```
        await database_sync_to_async(kanban_board.update_order)
```

```
            pipe_line_id,
```

```
            card_id_list
```

```
        )
```

```
        await self.broadcast_board_data_without_requester()
```

更新処理をした
Consumer以外に
ブロードキャスト

```
class KanbanConsumer(BaseJsonConsumer):  
  
    async def update_card_order(self, content):  
        """  
        ボード内のカードの並び順を更新する  
        """  
        pipe_line_id = content['pipeLineId']  
        card_id_list = content['cardIdList']  
        await database_sync_to_async(kanban_sv.update_  
            pipe_line_id,  
            card_id_list  
        )  
        await self.broadcast_board_data_without_requester()
```



```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {
  const socket = getters.getSocket;
  socket.sendObj({
    type: 'update_card_order',
    pipeLineId,
    cardIdList: cardList.map(x => x.cardId),
  });
  commit('updateCardOrder', { pipeLineId, cardList });
}
```

```
// MUTATION
```

```
updateCardOrder(state, { pipeLineId, cardList }) {
  const targetPipeLine = state.boardData.pipeLineList
    .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
  targetPipeLine.cardList = cardList;
},
```

処理を依頼したクライアントは
すでに更新できた体で
再描画済み

```
class KanbanConsumer(BaseJsonConsumer):
:
    async def broadcast_board_data_without_requester(self, *args):
        await self.group_send(
            self.room_group_name,
            {
                'type': 'send_board_data',
                'requester_id': self.consumer_id,
            }
        )
```

```
class KanbanConsumer(BaseJsonConsumer):  
:  
    async def broadcast_board_data_without_re  
        await self.group_send(  
            self.room_group_name,  
            {  
                'type': 'send_board_data',  
                'requester_id': self.consumer_id,  
            }  
        )
```

channel_layerのgroup_sendで
他のConsumerにsend_board_dataの
実行を依頼

```
class KanbanConsumer(BaseJsonConsumer):
:
    async def send_board_data(self, event):
        """
        ボードのデータを送る
        """
        # 自身が発火したブロードキャストなら無視する
        if event.get('requester_id') == self.consumer_id:
            return
        board_data = await database_sync_to_async(
            kanban_sv.get_board_data_board_id
        )(self.board_id)
        await self.send_data({
            'boardData': board_data,
        }, mutation='setBoardData')
```

```
class KanbanConsumer(BaseJsonConsumer):
:
    async def send_board_data(self, event):
        """
        ボードのデータを送る
        """
        # 自身が発火したブロードキャストなら無視する
        if event.get('requester_id') == self.consumer_id:
            return

        board_data = await database_sync_to_async(
            kanban_sv.get_board_data_board_id
        )(self.board_id)
        await self.send_data({
            'boardData': board_data,
        }, mutation='setBoardData')
```



Django ORM経由でボードの
構成データを再取得

```
class KanbanConsumer(BaseJsonConsumer):
:
    async def send_board_data(self, event):
        """
        ボードのデータを送る
        """
        # 自身が発火したブロードキャストなら無視する
        if event.get('requester_id') == self.consumer_id:
            return
        board_data = await database_sync_to_async(
            kanban_sv.get_board_data_board_id)(self.board_id)
        await self.send_data({
            'boardData': board_data,
        }, mutation='setBoardData')
```

新しいボードデータを
対応するクライアントに送信


```
class KanbanConsumer(BaseJsonConsumer):
:
async def send_board_data(self, event):
    """
    ボードのデータを送る
    """
    # 自身が発火したブロードキャストなら無視する
    if event.get('requester_id') == self.consumer_id:
        return
    board_data = await database_sync_to_async(
        kanban_sv.get_board_data_board_id
    )(self.board_id)
    await self.send_data({
        'boardData': board_data,
    }, mutation='setBoardData')
```

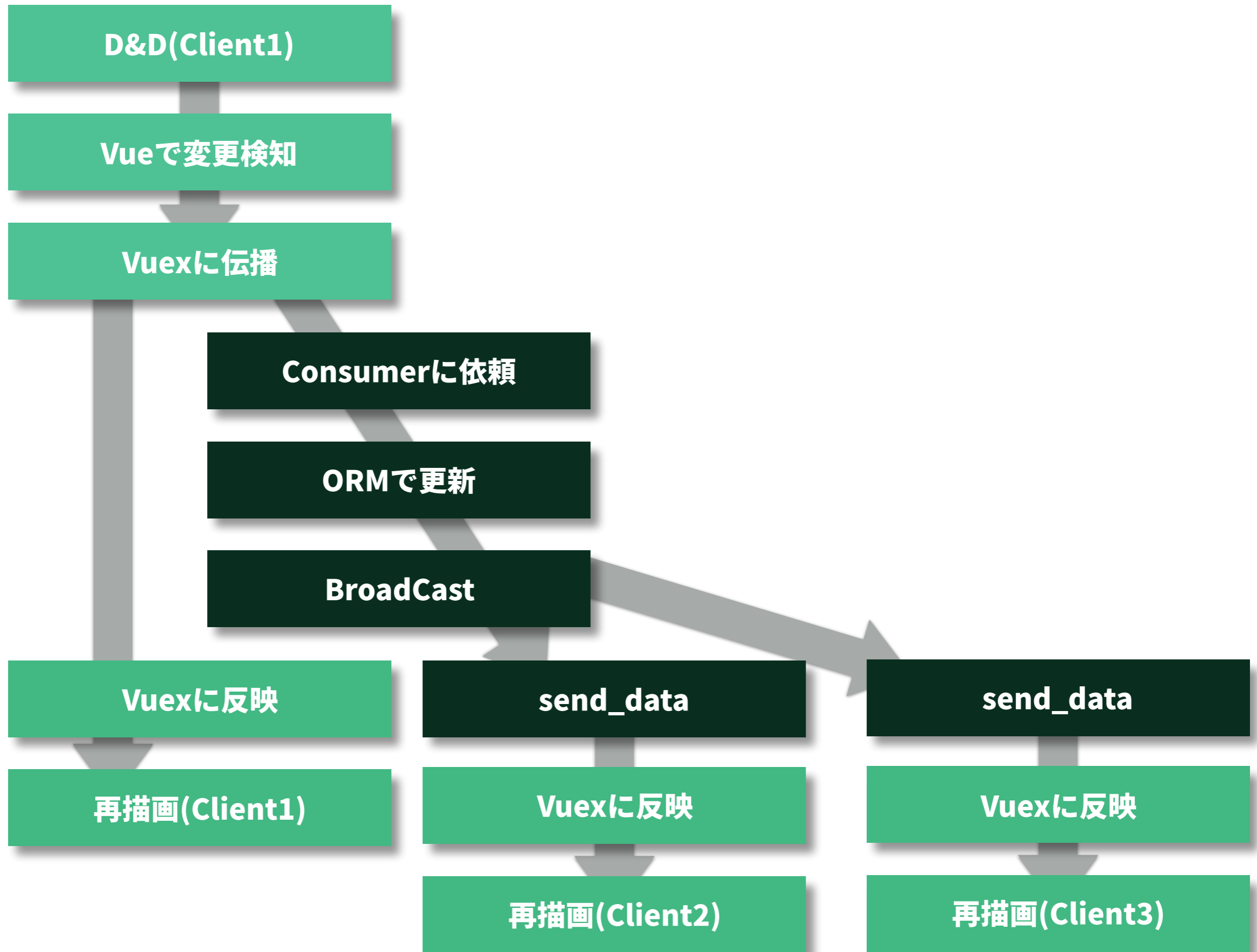
クライアント側の
mutation:setBoardDataに
引き渡す

```
const state = {
  boardData: {
    pipeLineList: [],
  },
  focusedCard: {},
  searchWord: '',
};

// MUTATION
setBoardData(state, { boardData }) {
  state.boardData = camelcaseKeys(boardData, { deep: true });
},
```

```
const state = {  
  boardData: {  
    pipeLineList: [],  
  },  
  focusedCard: {},  
  searchWord: '',  
};  
  
// MUTATION  
setBoardData(state, { boardData }) {  
  state.boardData = camelcaseKeys(boardData, { deep: true });  
},
```

返送されたデータで
更新し画面に反映



つらい話



たった5人でサービスを落とす方法

**カンバン周りの開発を終えてRelease直前に
動作確認をしてたら5人くらいでサーバごとハング**

```
class MyConsumer(AsyncJsonWebSocketConsumer or JsonWebSocketConsumer):  
  
    async def connect(self):  
        # accept or close  
        if YourCondition:  
            self.accept()  
        else:  
            self.close()  
  
    async def receive_json(self, content, **kwargs):  
        # receive message  
        print(content)  
  
        # echo back  
        await self.send_json(content)
```



```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):
```

```
    async def connect(self):
```

```
        # accept or close
```

```
        if YourCondition:
```

```
            self.accept()
```

```
        else:
```

```
            self.close()
```

```
    async def receive_json(self, content, **kwargs):
```

```
        # receive message
```

```
        print(content)
```

```
        # echo back
```

```
        await self.send_json(content)
```



たった5人でサービスを落とす方法

- **ConsumerはSync実装、Async実装の2種類**
- **Sync実装ではスレッドで動作**
- **Async実装ではコルーチンで動作**

たった5人でサービスを落とす方法

- **ConsumerはSync実装、Async実装の2種類**
- **Sync実装ではスレッドで動作**
- **Async実装ではコルーチンで動作**
- **重い処理がConsumer内にあった**
- **その処理が重なるとサーバごとハングして死**

SyncをAsyncに全面書き換え

SyncをAsyncに全面書き換え

- 各メソッドを `async def` に書き換え
- ORM周りを `database_to_async` でラップ

SyncをAsyncに全面書き換え

- 各メソッドを `async def`に書き換え
- ORM周りを `database_to_async`でラップ
- Sync実装の20倍でもサーバがハングしなくなる



静かな凡ミス

間違い探し

```
async def add_card(self, content):  
    """  
    カードの追加  
    """  
    pipe_line_id = content['pipeLineId']  
    card_title = content['cardTitle']  
    database_sync_to_async(kanban_sv.add_card)(  
        pipe_line_id,  
        card_title  
    )  
    self.broadcast_board_data()
```


間違い探し

```
async def add_card(self, content):  
    """  
    カードの追加  
    """  
    pipe_line_id = content['pipeLineId']  
    card_title = content['cardTitle']  
    await database_sync_to_async(kanban_sv.add_card)(  
        pipe_line_id,  
        card_title  
    )  
    await self.broadcast_board_data()
```

実行してもエラーが出ず

サイレントに処理されないので気が付かない



突然の

My SQL Server has gone away

間違い探し

```
async def hoge(self, event):  
    # カードを取得  
    card = await database_sync_to_async(Card.objects.get)(id=1)  
    # カードが所属するボードを取得して、その全体のデータを取得  
    board_data = await database_sync_to_async(kanban_sv.get_board_data)(  
        card.board  
    )
```

間違い探し

```
async def hoge(self, event):  
    # カードを取得  
    card = await database_sync_to_async(Card.objects.get)(id=1)  
    # カードが所属するボードを取得して、その全体のデータを取得  
    board_data = await database_sync_to_async(kanban_sv.get_board_data)(  
        card.board  
    )
```

FKをたどるだけでも、ORMの操作になる。
database_sync_to_asyncで囲わないと
接続がリークして、やがて死ぬ

<http://www.denzow.me/entry/2018/07/30/000955>



[Dd]enzow(ill)? with DB and Python

DBとか資格とかPythonとかの話をつらつらと

2018-07-30

編集

Django Channelsのdatabase_sync_to_asyncを理解しなくて死んだ話

Channels

Django

Python

Django Channelsの `@database_sync_to_async` デコレータを正しく理解しなくてえらい目('MySQL server has gone away')にあったので、確認して理解した内容を残しておきます。



あれ、Websocketで成功・失敗って・・・

- 例えばカード追加がたまに失敗するシステム
- 成功・失敗毎にクライアントにAlert出したい

```
await KanbanClient.deleteCard({
  boardId,
  cardId,
}).then(res => {
  alert('success');
}).catch(e => {
  alert('error');
});
```

成功時

失敗時

- **WebsocketはPromiseではない**
- **返答を待つという概念はない**

```
socket.sendObj({  
  type: 'add_card',  
  pipeLineId,  
  cardTitle,  
});
```

このメッセージの処理状況を
容易に管理できない

- **更新部分はAjaxで処理**
- **処理完了時に、クライアントがブロードキャストを依頼**
- **ローディング、成功失敗時の通知等が実装可能に**

```
async updateCardTitle({ commit, dispatch }, { boardId, cardId, title }) {
  const cardData = await KanbanClient.updateCardData({
    boardId,
    cardId,
    title,
  });
  dispatch('broadcastBoardData');
},
:
broadcastBoardData({ getters }) {
  console.log('call broadcastBoardData');
  const socket = getters.getSocket;
  socket.sendObj({
    type: 'broadcast_board_data',
  });
},
```

```
async updateCardTitle({ commit, dispatch }, { boardId, cardId, title }) {  
  const cardData = await KanbanClient.updateCardData({  
    boardId,  
    cardId,  
    title,  
  });  
  dispatch('broadcastBoardData');  
},  
:  
broadcastBoardData({ getters }) {  
  console.log('call broadcastBoardData');  
  const socket = getters.getSocket;  
  socket.sendObj({  
    type: 'broadcast_board_data',  
  });  
},  
},
```

Ajaxでカードを
更新(await)

```
async updateCardTitle({ commit, dispatch }, { boardId, cardId, title }) {  
  const cardData = await KanbanClient.updateCardData({  
    boardId,  
    cardId,  
    title,  
  });  
  dispatch('broadcastBoardData');  
},  
:  
broadcastBoardData({ getters }) {  
  console.log('call broadcastBoardData');  
  const socket = getters.getSocket;  
  socket.sendObj({  
    type: 'broadcast_board_data',  
  });  
},  
},
```

Ajaxが終わったら
別の処理を発火

```
async updateCardTitle({ commit, dispatch }, { boardId, cardId, title }) {  
  const cardData = await KanbanClient.updateCardData({  
    boardId,  
    cardId,  
    title,  
  });  
  dispatch('broadcastBoardData');  
},  
:
```

```
broadcastBoardData({ getters }) {  
  console.log('call broadcastBoardData');  
  const socket = getters.getSocket;  
  socket.sendObj({  
    type: 'broadcast_board_data',  
  });  
},
```

websocket経由で
consumerに
ブロードキャストを依頼



**ASGIサーバなのにChannels動かかんやん
(2018年6月頃)**

目次 README.rst

daphne

build passing pypi v2.2.2

Daphne is a HTTP, HTTP2 and WebSocket protocol server for [ASGI](#) and [ASGI-HTTP](#), developed to power Django Channels.

It supports automatic negotiation of protocols; there's no need for URL prefixing to determine WebSocket endpoints versus HTTP endpoints.

Note: Daphne 2 is not compatible with Channels 1.x applications, only with Channels 2.x and other ASGI applications. Install a 1.x version of Daphne for Channels 1.x support.

Running

Simply point Daphne to your ASGI application, and optionally set a bind address and port (defaults to localhost, port 8000):

```
daphne -b 0.0.0.0 -p 8001 django_project.asgi:application
```

<https://github.com/django/daphne>

README.md



The lightning-fast ASGI server.

build passing codecov 91% pypi package 0.3.5

Documentation: <https://www.uvicorn.org>

Requirements: Python 3.5, 3.6, 3.7

Uvicorn is a lightning-fast ASGI server implementation, using [uvloop](#) and [httptools](#).

<https://github.com/encode/uvicorn>

README.rst

Hypercorn

pipeline passed docs passing pypi v0.2.4 http 1.0,1.1,2 python 3.6 | 3.7 license MIT

Hypercorn is an ASGI web server based on the sans-io [hyper](#), [h11](#), [h2](#), and [wsproto](#) libraries and inspired by Gunicorn. Hypercorn supports HTTP/1, HTTP/2, and websockets and the ASGI 2 specification.

Hypercorn was initially part of [Quart](#) before being separated out into a standalone ASGI server. Hypercorn forked from version 0.5.0 of Quart.

Quickstart

Hypercorn can be installed via [pipenv](#) or [pip](#),

```
$ pipenv install hypercorn
$ pip install hypercorn
```

and requires Python 3.6.1 or higher.

With hypercorn installed ASGI frameworks (or apps) can be served via Hypercorn via the command line,

```
$ hypercorn module:app
```

<https://gitlab.com/pgjones/hypercorn>

- Django Channelsがちゃんと動いたのはdaphneのみ
- uvicornはAuthMiddlewareで死亡
- Hypercornはなんでか忘れたけど死亡

- Django Channelsがちゃんと動いたのはdaphneのみ
- uvicornはAuthMiddlewareで死亡
- Hypercornはなんでか忘れたけど死亡

とりあえずChannelsなら
daphne使っておけば安心(?)

まとめ

- 辛い話が最後まで続いたけど、Channelsは使いやすくていい！
- Websocketが書きやすくていい！
- 2.xのナレッジが少ないのでみんなでもOutput & Followしていこう
- ASGIでひろがるPythonに期待

Pythonやりたい人

The screenshot shows the Google Translate interface. The search bar contains the text '翻訳' (Translate). Below the search bar, there are navigation tabs: 'すべて' (All), 'ニュース' (News), 'ショッピング' (Shopping), '動画' (Video), '画像' (Image), 'もっと見る' (See more), '設定' (Settings), and 'ツール' (Tools). The search results show approximately 94,400,000 results in 0.30 seconds. The main content area displays the translation of 'We are hiring!' from English to Japanese. The English text is 'We are hiring!' with a '編集' (Edit) link. The Japanese text is '私達は雇っています！' (Watashitachi wa yatotte imasu!). There are also icons for voice input, voice output, and a copy icon. At the bottom, there are links for 'Google 翻訳で開く' (Open with Google Translate) and 'フィードバック' (Feedback).

Google

翻訳

すべて ニュース ショッピング 動画 画像 もっと見る 設定 ツール

約 94,400,000 件 (0.30 秒)

英語

We are hiring! 編集

日本語

私達は雇っています！
Watashitachi wa yatotte imasu!

Google 翻訳で開く フィードバック

➡ <https://bit.ly/2NPWCc8>