Elevating Experiences with Server-Sent Events

A Journey from Polling to Real-Time Vibes

Melhin Ahammad

Melhin Ahammad

Over a decade as a software engineer, or let's just say I've encountered my fair share of fumbles while striving to build software.

I lead one of the teams at CheMondis, a marketplace for chemicals. (chemondis.com)

Always tinkering with something; I’m the proud owner of a lot of incomplete projects.

GitHub: @melhin

Caveats

Django

Mandatory Definitions

Polling

WebSockets

Server Sent Events

Event Stream Format

```
: this is a test stream

----------

data: some text

----------

data: another message

----------

data: with two lines
```

```
event: userconnect

data: {"username": "bobby", "time":
"02:33:48"}

----------

event: userdisconnect

data: {"username": "bobby", "time":
"02:34:23"}
```

Lets Jump Right In

A Simple Counter Streaming View

```python
async def stream_timer(request: HttpRequest, *args, **kwargs):
    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0

        try:
            logging.info(f"{connection_id}: Connecting to stream")
            while True:
                events_count += 1
                event = "event: new\n"
                event += f"data: {events_count}\n\n"
                logging.info(f"{connection_id}: Sent events. {events_count}")
                yield event
                await asyncio.sleep(1)

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

A Simple Counter Streaming View

Async Generator

- Sleeps for a second
- Increments count
- Yields count

```python
async def stream_timer(request: HttpRequest, *args, **kwargs):
    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0

        try:
            logging.info(f"{connection_id}: Connecting to stream")
            while True:
                events_count += 1
                event = "event: new\n"
                event += f"data: {events_count}\n\n"
                logging.info(f"{connection_id}: Sent events. {events_count}")
                yield event
                await asyncio.sleep(1)

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

A Simple Counter Streaming View

Forming Of Event Data

- Event Name: New
- Data: Count

```
async def stream_timer(request: HttpRequest, *args, **kwargs):
    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0

        try:
            logging.info(f"{connection_id}: Connecting to stream")
            while True:
                events_count += 1
                event = "event: new\n"
                event += f"data: {events_count}\n\n"
                logging.info(f"{connection_id}: Sent events. {events_count}")
                yield event
                await asyncio.sleep(1)

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

A Simple Counter Streaming View

StreamingHttpResponse

- Is the mechanism Django uses to hold the connection and send data
- Sync (WSGI) usually holds the connection and worker
- Async(ASGI) uses the event loop and doesn’t hold up a worker

```
async def stream_timer(request: HttpRequest, *args, **kwargs):
    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0

        try:
            logging.info(f"{connection_id}: Connecting to stream")
            while True:
                events_count += 1
                event = "event: new\n"
                event += f"data: {events_count}\n\n"
                logging.info(f"{connection_id}: Sent events. {events_count}")
                yield event
                await asyncio.sleep(1)

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

A Simple Counter Streaming View

Disconnection

- Signals to the View of Disconnection
- Place to have Cleanup

```
async def stream_timer(request: HttpRequest, *args, **kwargs):
    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = uuid.uuid4()
        events_count = 0

        try:
            logging.info(f"{connection_id}: Connecting to stream")
            while True:
                events_count += 1
                event = "event: new\n"
                event += f"data: {events_count}\n\n"
                logging.info(f"{connection_id}: Sent events. {events_count}")
                yield event
                await asyncio.sleep(1)

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

Developing on Local

```
uvicorn your-app-name.asgi:application --port 8002 --reload --timeout-graceful-shutdown 0
```

```
❯ uvicorn sse_liveqa.asgi:application --port 8002 --reload --timeout-graceful-
shutdown 0 --reload-include "*.html"


INFO:     Will watch for changes in these directories: ['<directory/name>']
INFO:     Uvicorn running on http://127.0.0.1:8002 (Press CTRL+C to q
uit)
INFO:     Started reloader process [77859] using WatchFiles
INFO:     Started server process [77861]
INFO:     Waiting for application startup.
INFO:     ASGI 'lifespan' protocol appears unsupported.
INFO:     Application startup complete.
INFO:     127.0.0.1:63564 - "GET /timer/ HTTP/1.1" 200 OK
2024-05-12 08:16:59,542 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Connecting to stream
2024-05-12 08:16:59,543 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 1
2024-05-12 08:17:00,544 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 2
2024-05-12 08:17:01,545 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 3
2024-05-12 08:17:02,548 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 4
2024-05-12 08:17:03,550 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 5
2024-05-12 08:17:04,551 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 6
2024-05-12 08:17:05,553 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 7
2024-05-12 08:17:06,555 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 8
2024-05-12 08:17:07,556 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 9
2024-05-12 08:17:08,558 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 10
2024-05-12 08:17:09,560 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Sent events. 11
2024-05-12 08:17:09,733 INFO 8622275264 77861 views root cfa67479-227
9-48ae-9c0f-f5382a2635e5: Disconnected after events. 11
```

```
❯ curl -i -N  127.0.0.1:8002/timer/
HTTP/1.1 200 OK
date: Sun, 12 May 2024 08:16:58 GMT
server: uvicorn
Content-Type: text/event-stream
X-Frame-Options: DENY
Vary: origin
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin
Cross-Origin-Opener-Policy: same-origin
Transfer-Encoding: chunked

event: new
data: 1

event: new
data: 2

event: new
data: 3

event: new
data: 4

event: new
data: 5

event: new
data: 6

event: new
data: 7

event: new
data: 8

event: new
data: 9

event: new
data: 10

event: new
data: 11

^C
```

Why are we embarking on this journey?

Our primary focus revolves around delivering substantial value to the users

Over the years we have build a stable a synchronous Django application

Explore, Explore, Explore

Real-time interactions enrich modern web application experience.

Web Applications Requiring Real-time Functionality

Messaging Apps

Collaborative Tools

Live Streaming Platforms

Online Gaming

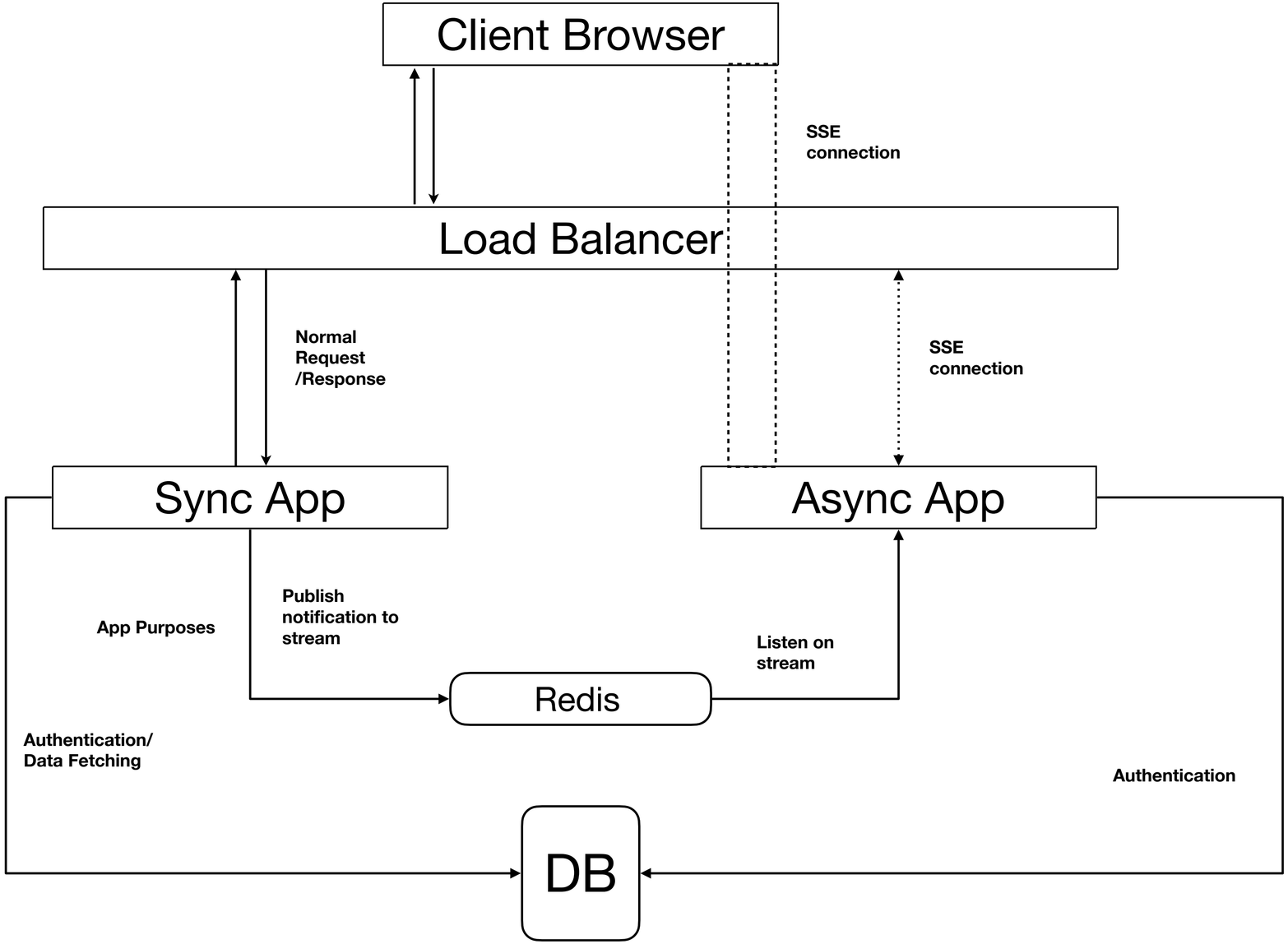
Real Time Dashboards

Notifications

How would we do this ?

- Keep the synchronous app running as it is
- Add listeners to interactions that you would like to be notified
- Publish the interactions to a common channel
- Run an asynchronous app for only realtime application
- This app listens to the common channel

Async Django: The practical guide you've been *awaiting* for by Carlton Gibson


Client Browser
SSE
connection
Load Balancer
Normal
Request
/Response
SSE
connection
Sync App
Async App
App Purposes
Publish
notification to
stream
Listen on
stream
Redis
Authentication/
Data Fetching
Authentication
DB

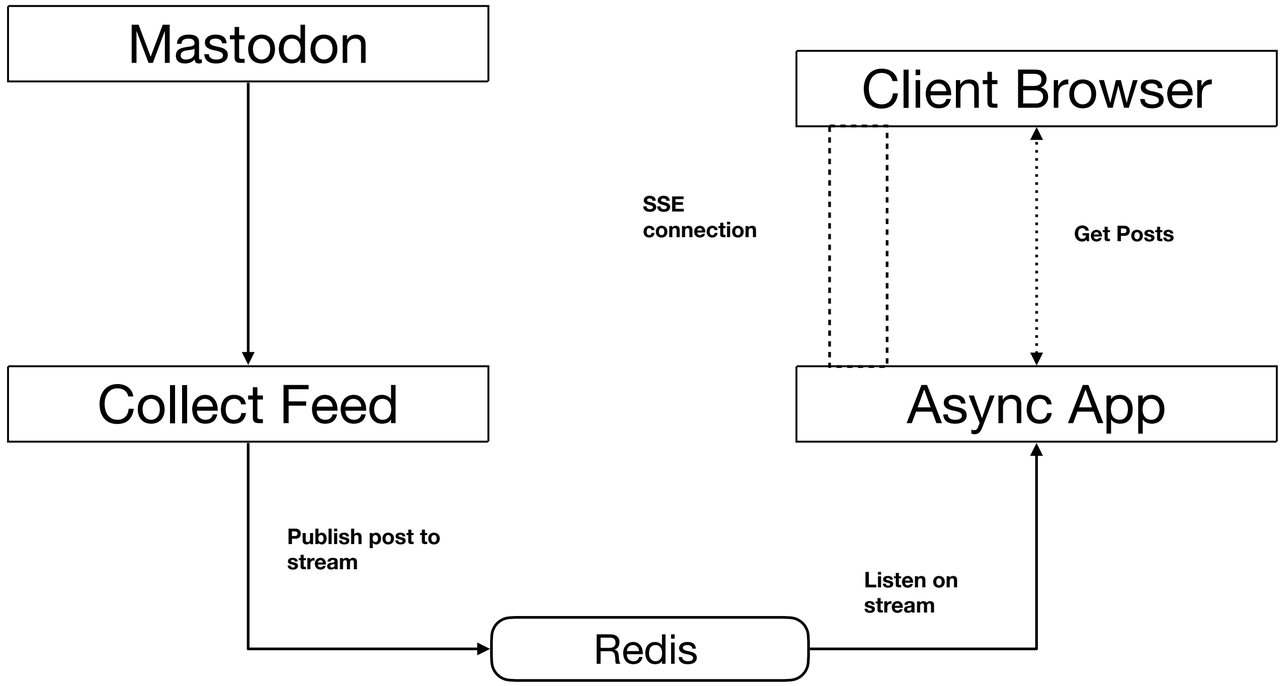
Lets look at another Example

Fediverse

Consume Mastodon stream updates, process, and redistribute in real-time.

Breaking this down in steps

- Stream Consumption: Collect Mastodon stream updates via the sync app
- Processing: Parse the update and store it in a different data structure
- Distribute: Send processed updates to all the connections to the async app

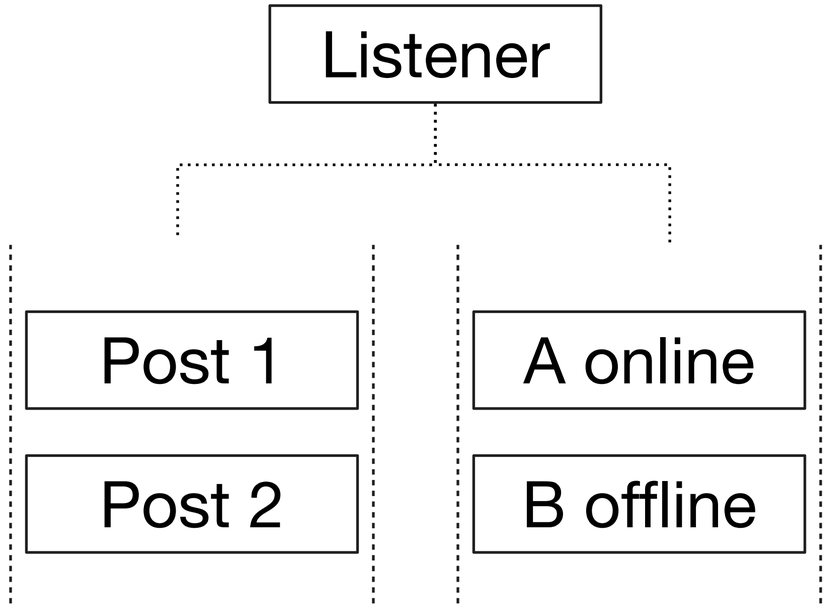

Mastodon
Client Browser
SSE
connection
Get Posts
Collect Feed
Async App
Publish post to
stream
Listen on
stream
Redis

Redis as a Common Channel

Redis Streams

Listening to multiple Redis streams

- Listener: listens on multiple streams
- For example: Post stream and Status stream
- As any when messages come in the stream the listener picks them up



How can this be done ?

Listening on streams together

```
    async def listen_on_multiple_streams(
        self,
        last_id_returned: str,
        timeout=LISTEN_TIMEOUT,
    ):
        logger.info("Fetching from stream")
        aredis = await
self.connection_factory.get_connection()
        if not last_id_returned:
            last_id_returned = "$"
        return await aredis.xread(
            count=1,
            streams={
                POST_STREAM: last_id_returned,
                ONLINE_STREAM: last_id_returned,
            },
            block=timeout,
        )
```

View

```python
@require_http_methods(["GET"])
async def stream_new_activity_and_presence(request: HttpRequest, *args, **kwargs):

    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = str(uuid.uuid4())
        events_count = 0
        listener = PostService()
        last_id_returned = None
        await listener.send_status_to_stream(
            message=OnlineStatus(user=connection_id, status="online")
        )

        try:
            while True:
                message = await listener.listen_on_multiple_streams(
                    last_id_returned=last_id_returned,
                )
                if message:
                    last_id_returned = message[0][1][0][0]
                    if message[0][0].decode("utf-8") == ONLINE_STREAM:
                        dumped_data = message[0][1][0][1][b"v"].decode("utf-8")
                        event = "event: status\n"
                    else:
                        dumped_data = json.dumps(
                            {"new_message_id": last_id_returned.decode("utf-8")}
                        )
                        event = "event: new-notification\n"
                    event += f"data: {dumped_data}\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sent events. {events_count}")
                    yield event
                else:
                    event = "event: heartbeat\n"
                    event += "data: ping\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sending heartbeats")
                    yield event

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            await listener.send_status_to_stream(
                message=OnlineStatus(user=connection_id, status="offline")
            )
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

- Sends online status on connection

```
@require_http_methods(["GET"])
async def stream_new_activity_and_presence(request: HttpRequest, *args, **kwargs):

    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = str(uuid.uuid4())
        events_count = 0
        listener = PostService()
        last_id_returned = None
        await listener.send_status_to_stream(
            message=OnlineStatus(user=connection_id, status="online")
        )

        try:
            while True:
                message = await listener.listen_on_multiple_streams(
                    last_id_returned=last_id_returned,
                )
                if message:
                    last_id_returned = message[0][1][0][0]
                    if message[0][0].decode("utf-8") == ONLINE_STREAM:
                        dumped_data = message[0][1][0][1][b"v"].decode("utf-8")
                        event = "event: status\n"
                    else:
                        dumped_data = json.dumps(
                            {"new_message_id": last_id_returned.decode("utf-8")}
                        )
                        event = "event: new-notification\n"
                    event += f"data: {dumped_data}\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sent events. {events_count}")
                    yield event
                else:
                    event = "event: heartbeat\n"
                    event += "data: ping\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sending heartbeats")
                    yield event

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            await listener.send_status_to_stream(
                message=OnlineStatus(user=connection_id, status="offline")
            )
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

- Sends online status on connection
- Listens on multiple streams

```
@require_http_methods(["GET"])
async def stream_new_activity_and_presence(request: HttpRequest, *args, **kwargs):

    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = str(uuid.uuid4())
        events_count = 0
        listener = PostService()
        last_id_returned = None
        await listener.send_status_to_stream(
            message=OnlineStatus(user=connection_id, status="online")
        )

        try:
            while True:
                message = await listener.listen_on_multiple_streams(
                    last_id_returned=last_id_returned,
                )
                if message:
                    last_id_returned = message[0][1][0][0]
                    if message[0][0].decode("utf-8") == ONLINE_STREAM:
                        dumped_data = message[0][1][0][1][b"v"].decode("utf-8")
                        event = "event: status\n"
                    else:
                        dumped_data = json.dumps(
                            {"new_message_id": last_id_returned.decode("utf-8")}
                        )
                        event = "event: new-notification\n"
                    event += f"data: {dumped_data}\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sent events. {events_count}")
                    yield event
                else:
                    event = "event: heartbeat\n"
                    event += "data: ping\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sending heartbeats")
                    yield event

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            await listener.send_status_to_stream(
                message=OnlineStatus(user=connection_id, status="offline")
            )
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

- Sends online status on connection
- Listens on multiple streams
- Based in which stream we send the respective event

```python
@require_http_methods(["GET"])
async def stream_new_activity_and_presence(request: HttpRequest, *args, **kwargs):

    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = str(uuid.uuid4())
        events_count = 0
        listener = PostService()
        last_id_returned = None
        await listener.send_status_to_stream(
            message=OnlineStatus(user=connection_id, status="online")
        )

        try:
            while True:
                message = await listener.listen_on_multiple_streams(
                    last_id_returned=last_id_returned,
                )
                if message:
                    last_id_returned = message[0][1][0][0]
                    if message[0][0].decode("utf-8") == ONLINE_STREAM:
                        dumped_data = message[0][1][0][1][b"v"].decode("utf-8")
                        event = "event: status\n"
                    else:
                        dumped_data = json.dumps(
                            {"new_message_id": last_id_returned.decode("utf-8")}
                        )
                        event = "event: new-notification\n"
                    event += f"data: {dumped_data}\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sent events. {events_count}")
                    yield event
                else:
                    event = "event: heartbeat\n"
                    event += "data: ping\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sending heartbeats")
                    yield event

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            await listener.send_status_to_stream(
                message=OnlineStatus(user=connection_id, status="offline")
            )
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

- Sends online status on connection
- Listens on multiple streams
- Based in which stream we send the respective event
- Sends a heart beat when there is no message

```
@require_http_methods(["GET"])
async def stream_new_activity_and_presence(request: HttpRequest, *args, **kwargs):

    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = str(uuid.uuid4())
        events_count = 0
        listener = PostService()
        last_id_returned = None
        await listener.send_status_to_stream(
            message=OnlineStatus(user=connection_id, status="online")
        )

        try:
            while True:
                message = await listener.listen_on_multiple_streams(
                    last_id_returned=last_id_returned,
                )
                if message:
                    last_id_returned = message[0][1][0][0]
                    if message[0][0].decode("utf-8") == ONLINE_STREAM:
                        dumped_data = message[0][1][0][1][b"v"].decode("utf-8")
                        event = "event: status\n"
                    else:
                        dumped_data = json.dumps(
                            {"new_message_id": last_id_returned.decode("utf-8")}
                        )
                        event = "event: new-notification\n"
                    event += f"data: {dumped_data}\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sent events. {events_count}")
                    yield event
                else:
                    event = "event: heartbeat\n"
                    event += "data: ping\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sending heartbeats")
                    yield event

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            await listener.send_status_to_stream(
                message=OnlineStatus(user=connection_id, status="offline")
            )
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

- Sends online status on connection
- Listens on multiple streams
- Based in which stream we send the respective event
- Sends a heart beat when there is no message
- On disconnection send an event

```python
@require_http_methods(["GET"])
async def stream_new_activity_and_presence(request: HttpRequest, *args, **kwargs):

    async def streamed_events() -> AsyncGenerator[str, None]:
        """Listen for events and generate an SSE message for each event"""
        connection_id = str(uuid.uuid4())
        events_count = 0
        listener = PostService()
        last_id_returned = None
        await listener.send_status_to_stream(
            message=OnlineStatus(user=connection_id, status="online")
        )

        try:
            while True:
                message = await listener.listen_on_multiple_streams(
                    last_id_returned=last_id_returned,
                )
                if message:
                    last_id_returned = message[0][1][0][0]
                    if message[0][0].decode("utf-8") == ONLINE_STREAM:
                        dumped_data = message[0][1][0][1][b"v"].decode("utf-8")
                        event = "event: status\n"
                    else:
                        dumped_data = json.dumps(
                            {"new_message_id": last_id_returned.decode("utf-8")}
                        )
                        event = "event: new-notification\n"
                    event += f"data: {dumped_data}\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sent events. {events_count}")
                    yield event
                else:
                    event = "event: heartbeat\n"
                    event += "data: ping\n\n"
                    events_count += 1
                    logging.info(f"{connection_id}: Sending heartbeats")
                    yield event

        except asyncio.CancelledError:
            logging.info(f"{connection_id}: Disconnected after events. {events_count}")
            await listener.send_status_to_stream(
                message=OnlineStatus(user=connection_id, status="offline")
            )
            raise

    return StreamingHttpResponse(streamed_events(), content_type="text/event-stream")
```

FrontEnd

```
<script src="https://unpkg.com/htmx.org@1.9.10/dist/ext/sse.js"></script>
  <script>
    const source = new EventSource("{{ stream_server }}", {
      withCredentials: true,
    });

    source.addEventListener("new-notification", (event) => {
      const payload = JSON.parse(event.data);
      fetch(`/streams/new/${payload.new_message_id}/`)
        .then((response) => response.text())
        .then((data) => {
          existing = notifications.innerHTML;
          notifications.innerHTML = data + existing;
          htmx.process(notifications);
        });
    });
```

FrontEnd

```
<script src="https://unpkg.com/htmx.org@1.9.10/dist/ext/sse.js"></script>
  <script>
    const source = new EventSource("{{ stream_server }}", {
      withCredentials: true,
    });

    source.addEventListener("new-notification", (event) => {
      const payload = JSON.parse(event.data);
      fetch(`/streams/new/${payload.new_message_id}/`)
        .then((response) => response.text())
        .then((data) => {
          existing = notifications.innerHTML;
          notifications.innerHTML = data + existing;
          htmx.process(notifications);
        });
    });
```

FrontEnd

```
<script src="https://unpkg.com/htmx.org@1.9.10/dist/ext/sse.js"></script>
  <script>
    const source = new EventSource("{{ stream_server }}", {
      withCredentials: true,
    });

    source.addEventListener("new-notification", (event) => {
      const payload = JSON.parse(event.data);
      fetch(`/streams/new/${payload.new_message_id}/`)
        .then((response) => response.text())
        .then((data) => {
          existing = notifications.innerHTML;
          notifications.innerHTML = data + existing;
          htmx.process(notifications);
        });
    });
```

FrontEnd

```
<script src="https://unpkg.com/htmx.org@1.9.10/dist/ext/sse.js"></script>
  <script>
    const source = new EventSource("{{ stream_server }}", {
      withCredentials: true,
    });

    source.addEventListener("new-notification", (event) => {
      const payload = JSON.parse(event.data);
      fetch(`/streams/new/${payload.new_message_id}/`)
        .then((response) => response.text())
        .then((data) => {
          existing = notifications.innerHTML;
          notifications.innerHTML = data + existing;
          htmx.process(notifications);
        });
    });
```

HTMX(If you store state)

```
<div hx-ext="sse" sse-connect="{{ stream_server }}/content/
notifications/">
  <h2>New Mastodon Posts</h2>
    <div hx-get="{% url 'content:new-content' %}" hx-
trigger="sse:new-notification" hx-swap="afterbegin">
    </div>
</div>
```

Why not Just use Websockets ?

- Bidirectional realtime connection is not needed most of the cases
- Changes in infra to support Web Socket
- Compressions are not supported out of the Box

Interesting talk by SSE vs WebSockets vs Long Polling. Martin Chaov. JS Fest 2018

Is everything really as Rosy as it seems?

Caveats

- Settings
- Middleware
- Installed Apps
- Connection Limitation

Middleware’s

ThreadPool Executor

Takeaways

- Server Sent Events
- Redis Streams
- Fediverse

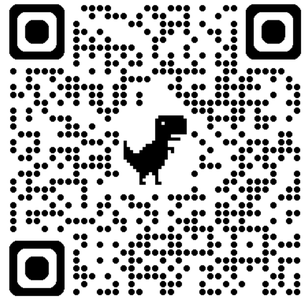
Some Performance stats (Only as reference)

- For purely **timer** on a free render instance i.e 512 Mb 0.1 cpu we are able to reach approx **500 connections**
- For the **stream connection** with an external given Aiven redis db of 1 Cpu / 1GB Ram we get approx **150-200 connections**

Thank You

All the code examples provided in the presentation is available in https://github.com/melhin/django-sse-liveqa

SSE vs WebSockets vs Long Polling. Martin Chaov. JS Fest 2018 - YouTube



"Just Add Await: Retrofitting Async Into Django" - Andrew Godwin

Async Django: The practical guide you've been *awaiting* for by Carlton Gibson