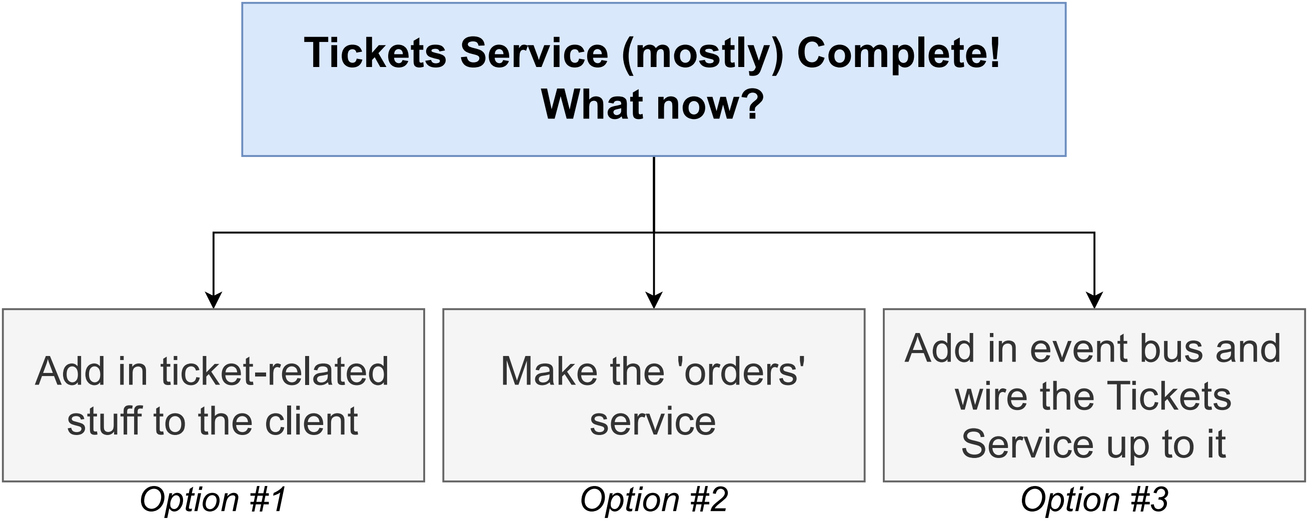
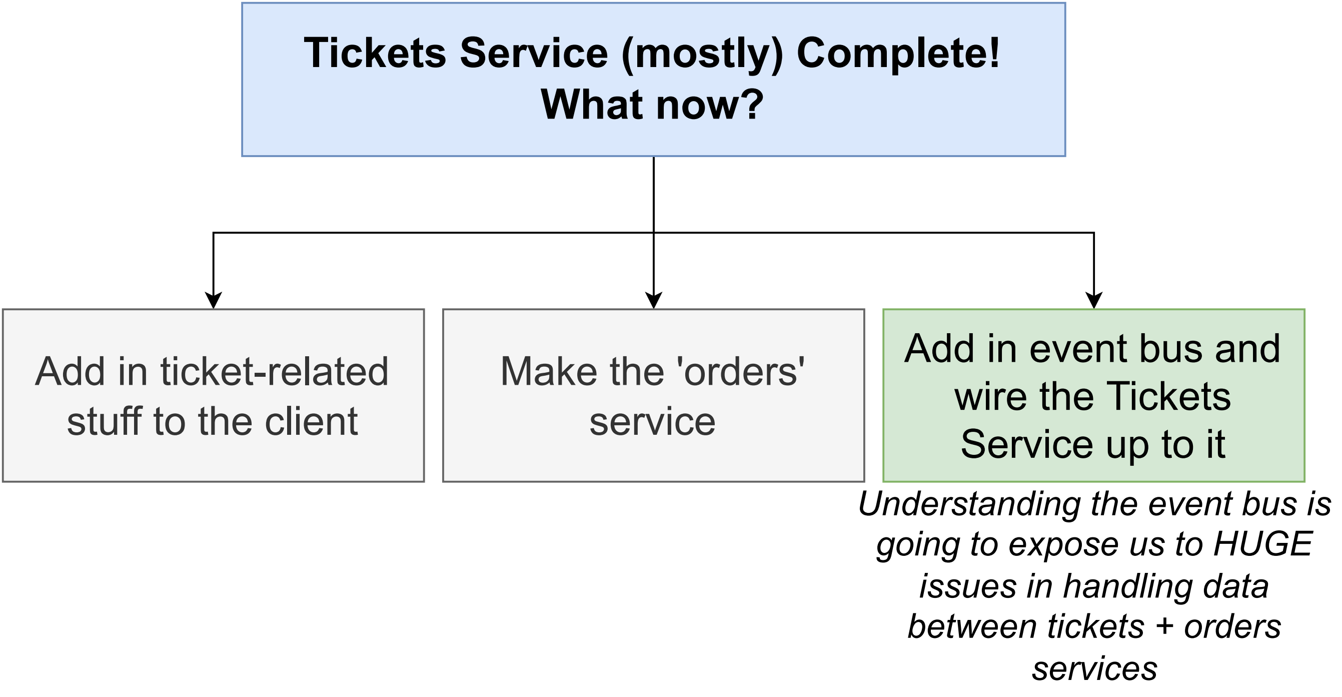
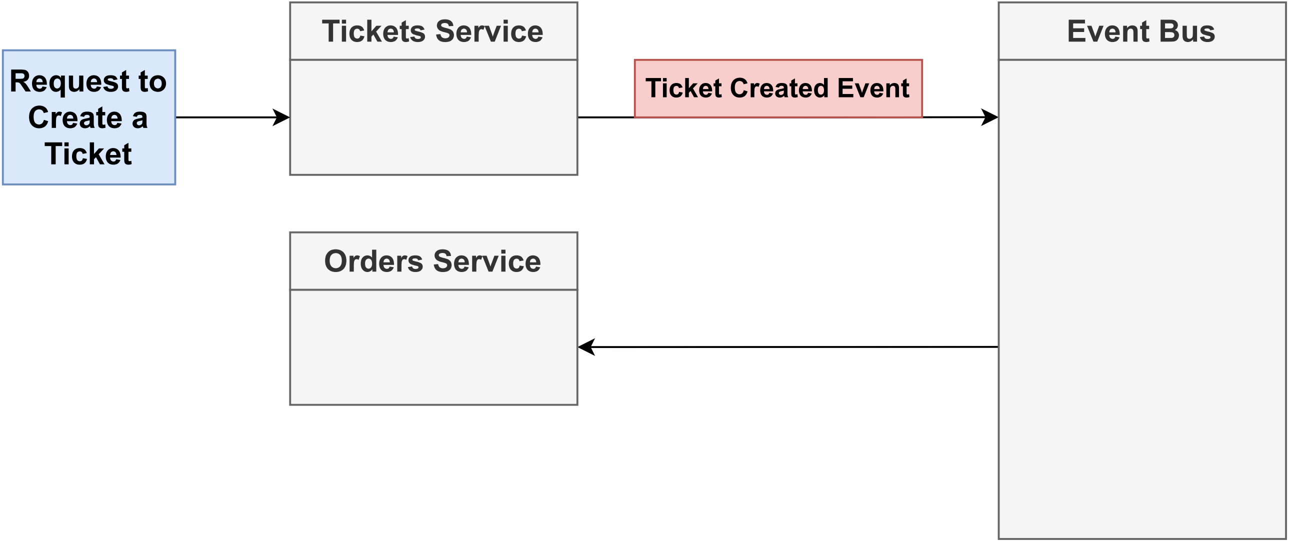
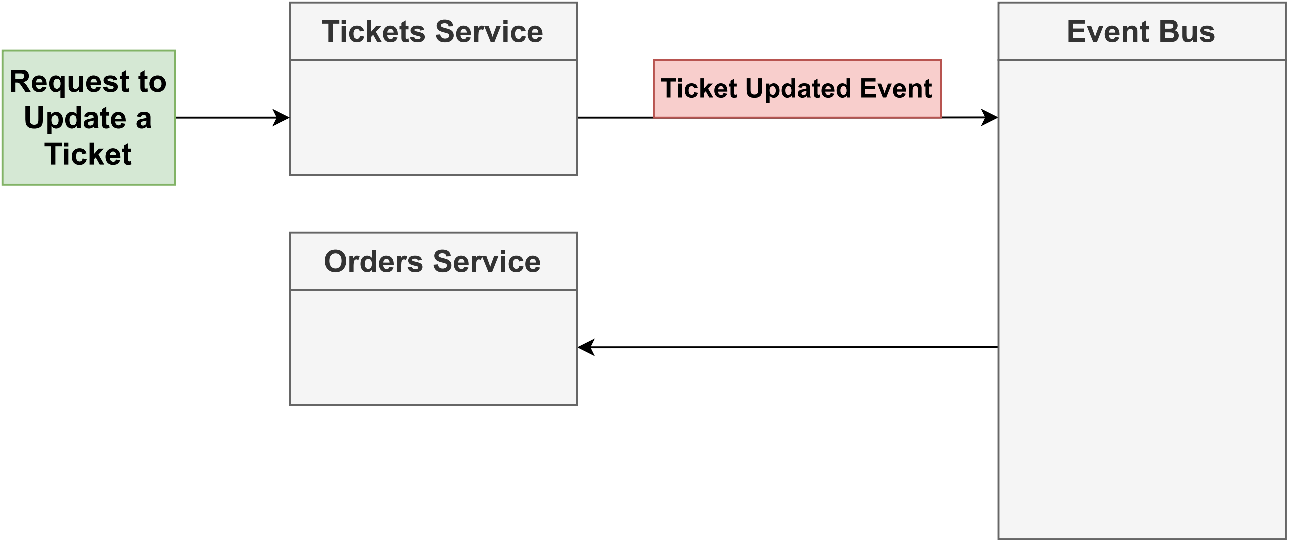

Tickets Service (mostly) Complete! What now?
Add in ticket-related stuff to the client
Make the 'orders' service
Add in event bus and wire the Tickets Service up to it
Option #1
Option #2
Option #3


Tickets Service (mostly) Complete! What now?
Add in ticket-related stuff to the client
Make the 'orders' service
Add in event bus and wire the Tickets Service up to it
Understanding the event bus is going to expose us to HUGE issues in handling data between tickets + orders services


Request to Create a Ticket
Tickets Service
Ticket Created Event
Event Bus
Orders Service


Request to Update a Ticket
Tickets Service
Ticket Updated Event
Event Bus
Orders Service

NATS Streaming Server

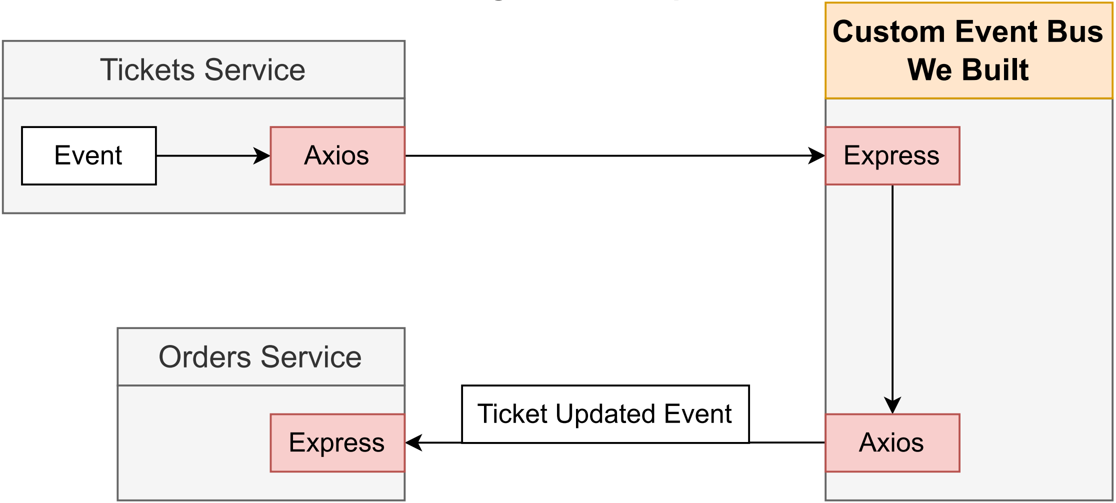
Docs at:
docs.nats.io

NATS and **NATS Streaming Server** are two different things

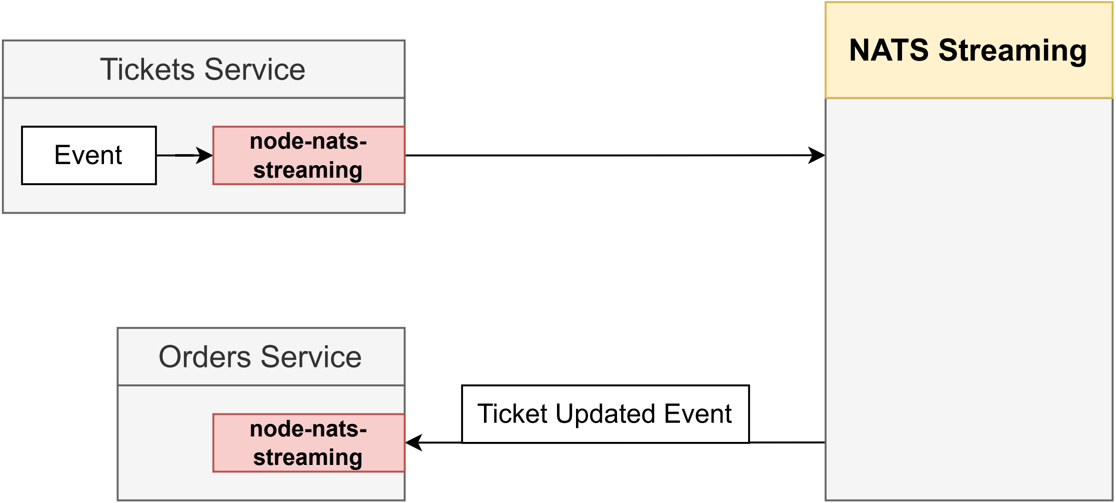
NATS Streaming implements some extraordinarily important design decisions that will affect our app

We are going to run the official 'nats-streaming' docker image in kubernetes. Need to read the image's docs

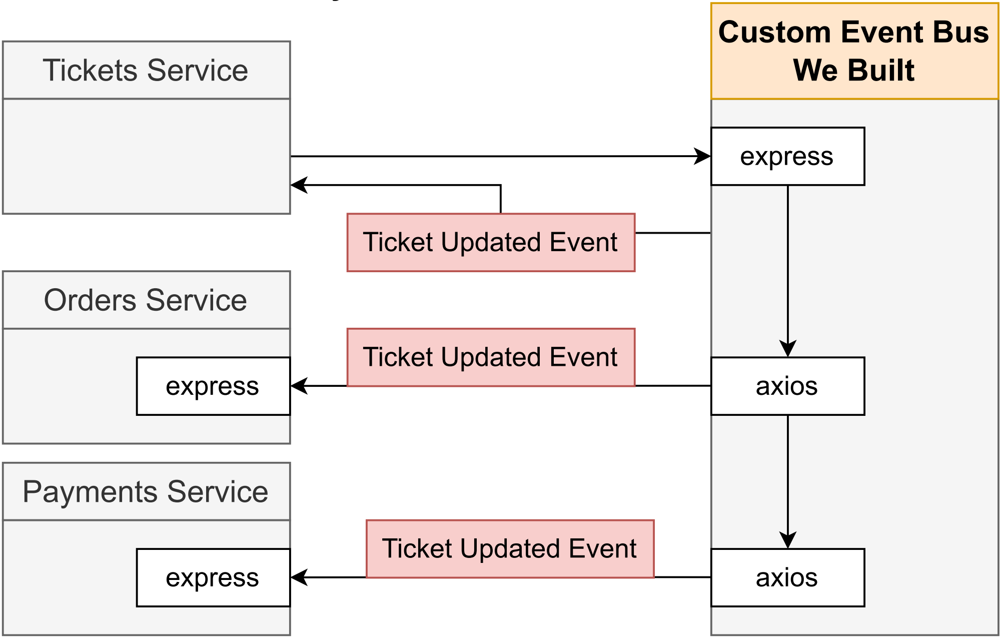
Our Custom Event Bus shared events using Axios + Express



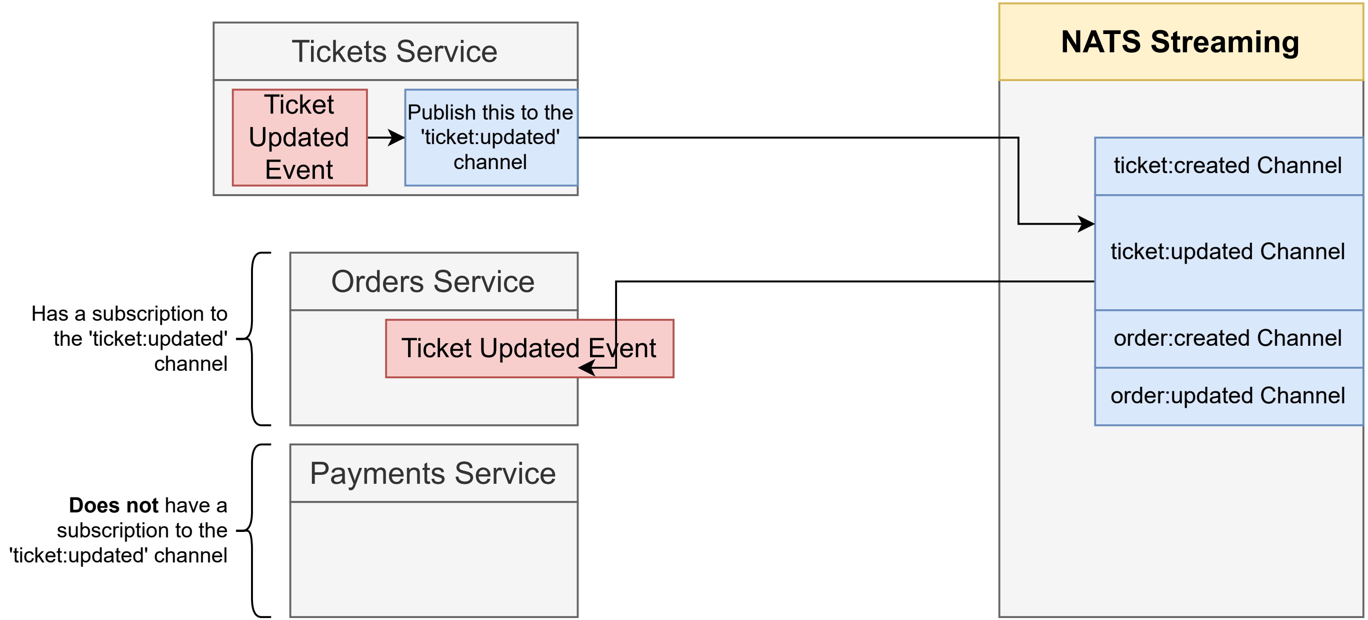
To communicate with NATS, we will use a *client library* called node-nats-streaming



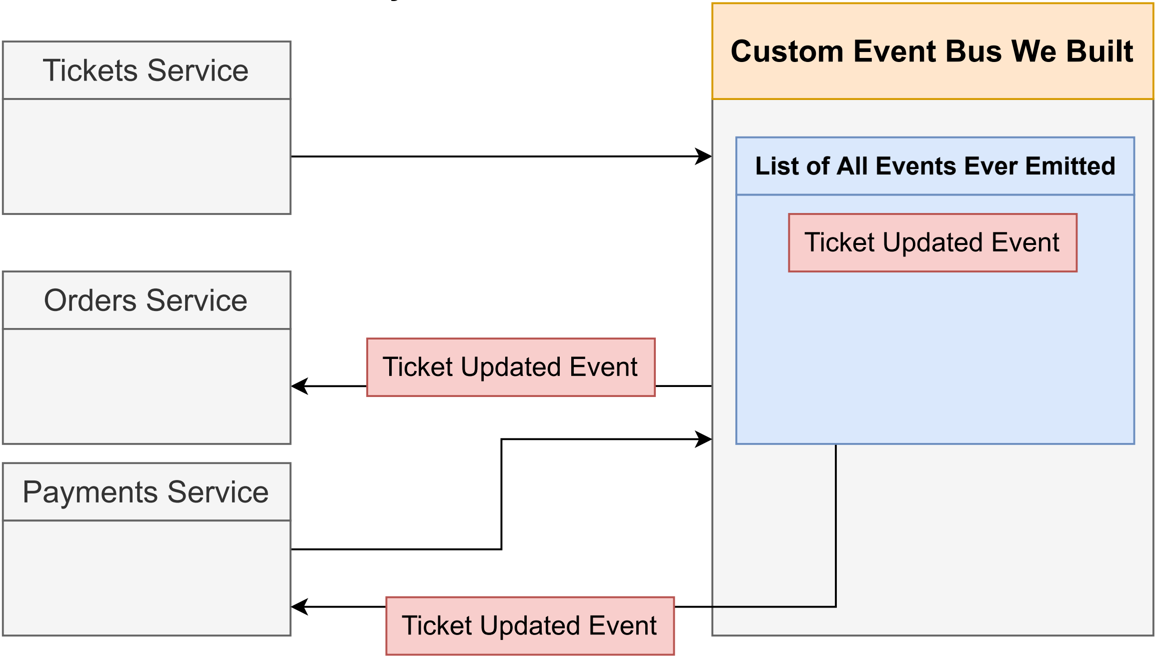
Our Custom Event Bus sent events to every service



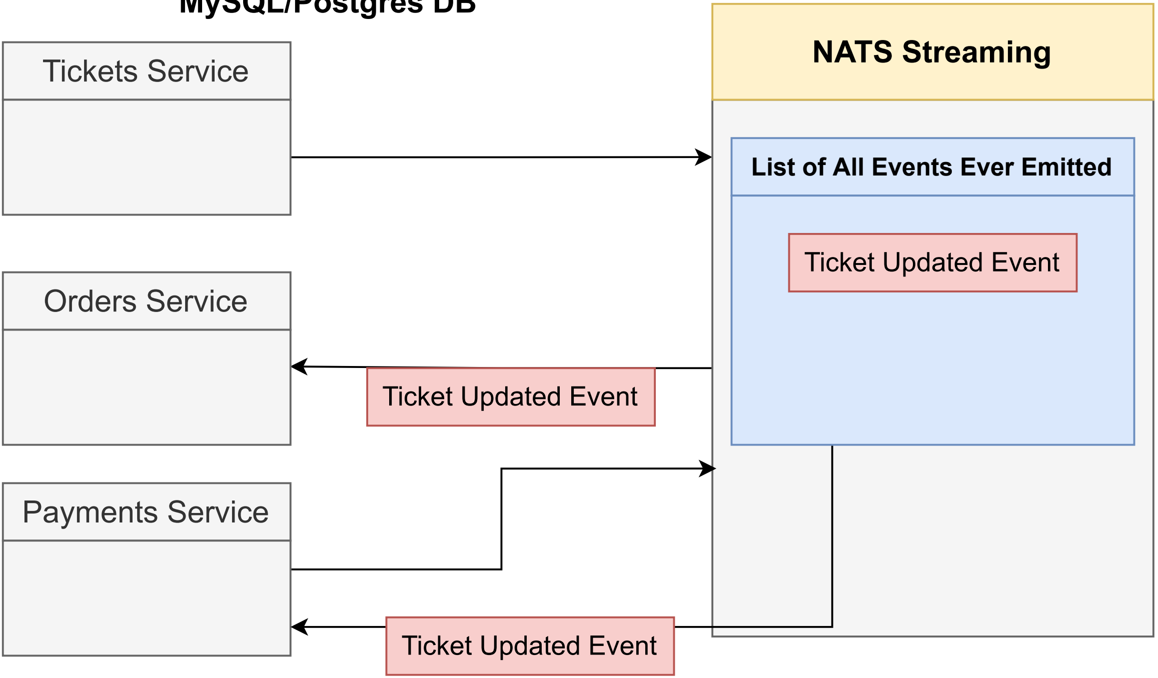
NATS Streaming requires us to subscribe to *channels.* Events are emitted to specific channels



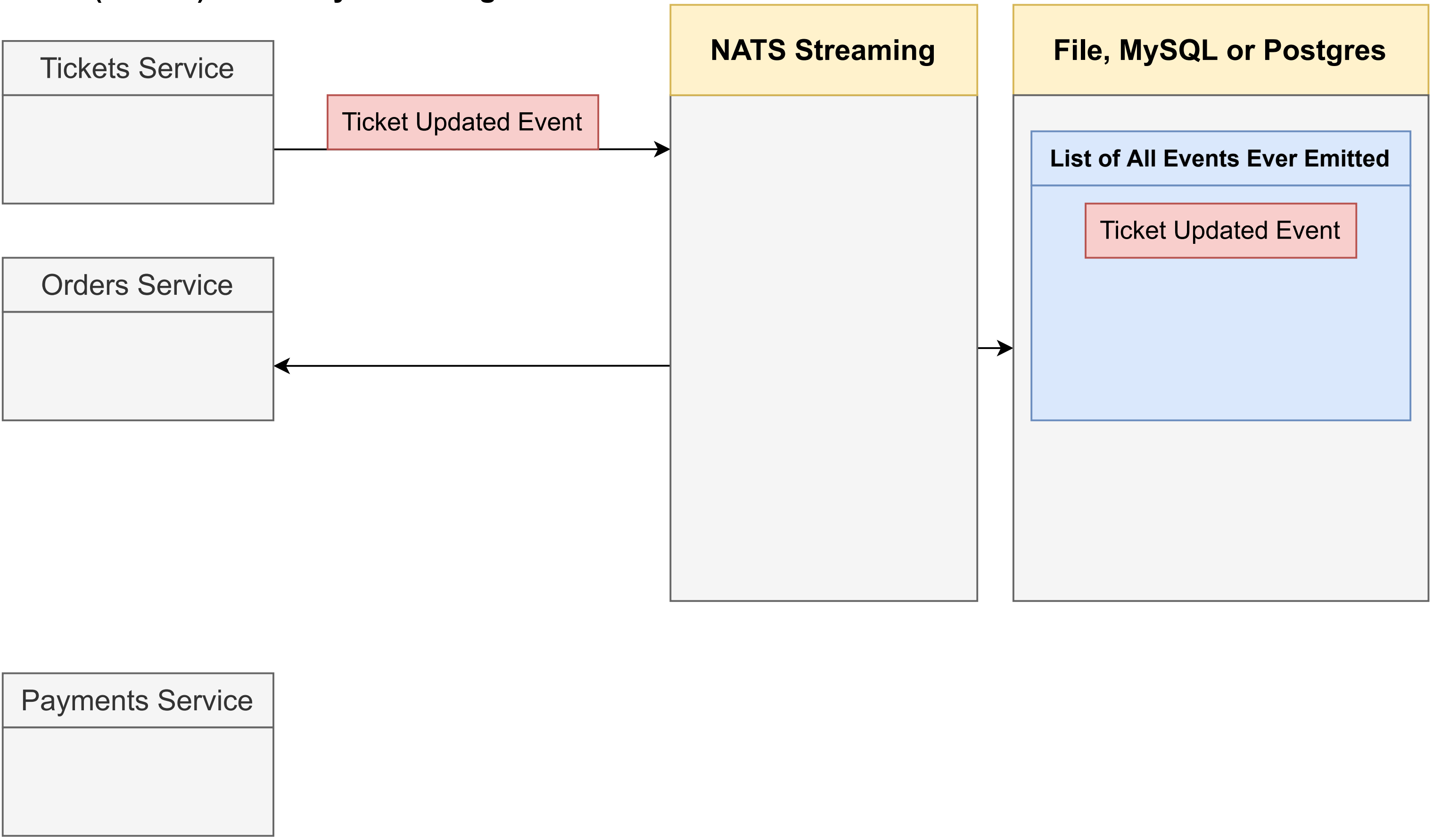
Our Event Bus stored events in memory



NATS Streaming stores all events in memory (default), flat files or in a MySQL/Postgres DB



NATS Streaming stores all events in flat files (default) or in a MySQL/Postgres DB



Short Term Goal

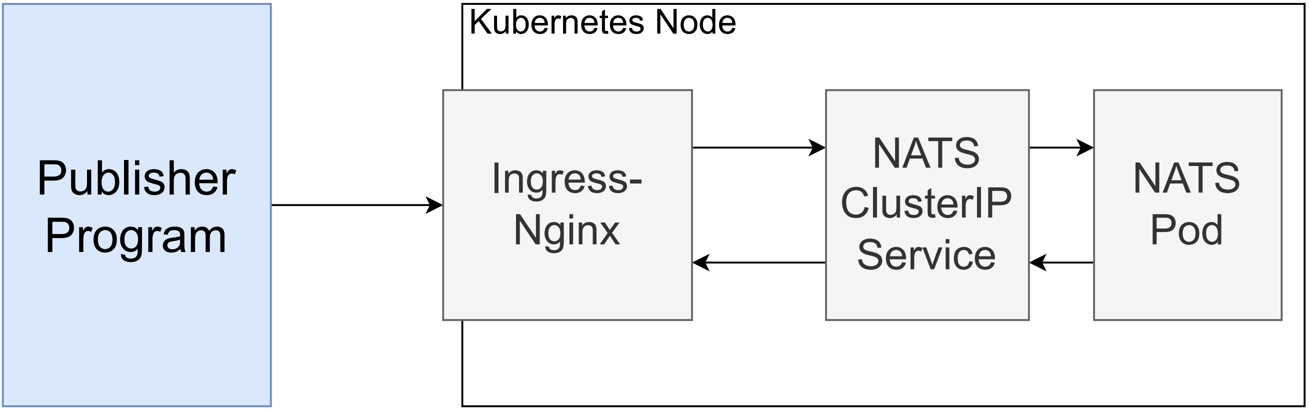
Create a new sub-project with typescript support

Install node-nats-streaming library and connect to nats streaming server

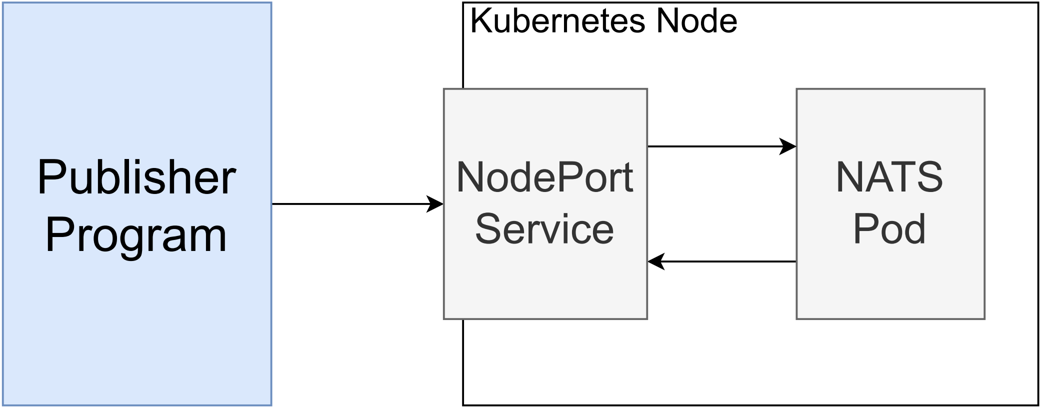
We should have *two npm scripts,* one to run code to *emit* events, and one to run code to *listen for* events

This program will be ran *outside* of kubernetes!

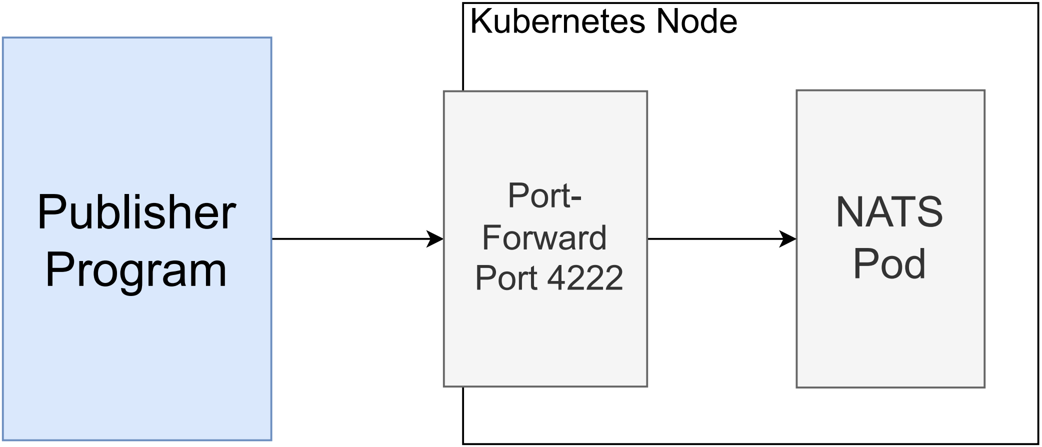
Option #1 to Connect



Option #2 to Connect



Option #3 to Connect



Quick Note

We are going to build up some pretty solid code around node-nats-streaming

↓

This code will be complex, and you will wonder 'why are we doing this' at several/many/all times

↓

The goal of this video is to highlight some big issues with that demo - these issues are why we are going to do this refactor

A Few Issues

Oh, we broke our tests

All services need to have an exactly accurate, common definition of what data is in each type of event

All services need to have a precise definition of the *subject* of each event

We need to make it *really easy and painless* to send/receive events

@sgtickets/common Lib

EventNames

TS Enum that makes it (nearly) impossible to misspell some event name

buildClient

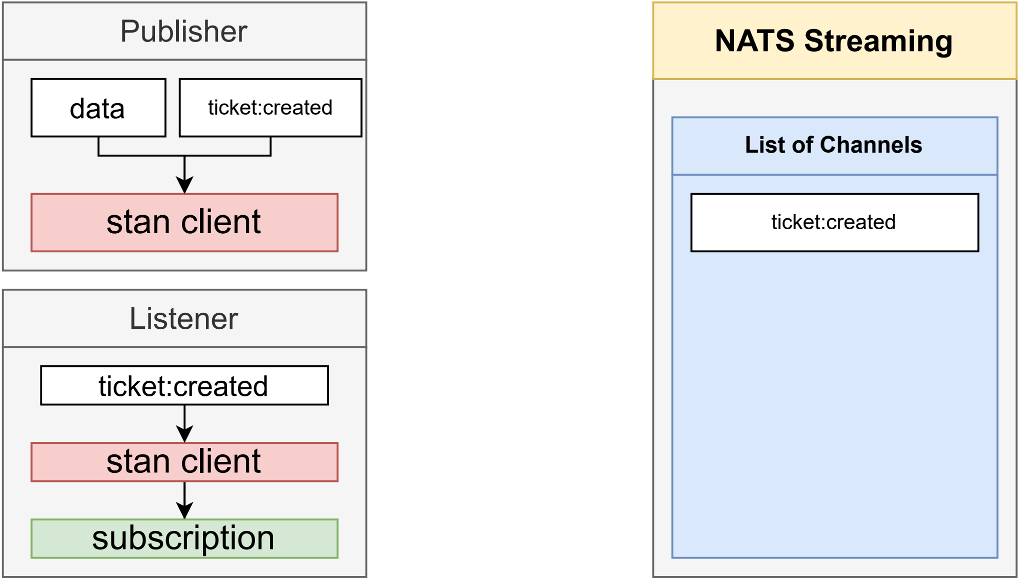
Function that makes it really easy (async/await, not callbacks) to create a new NATS client

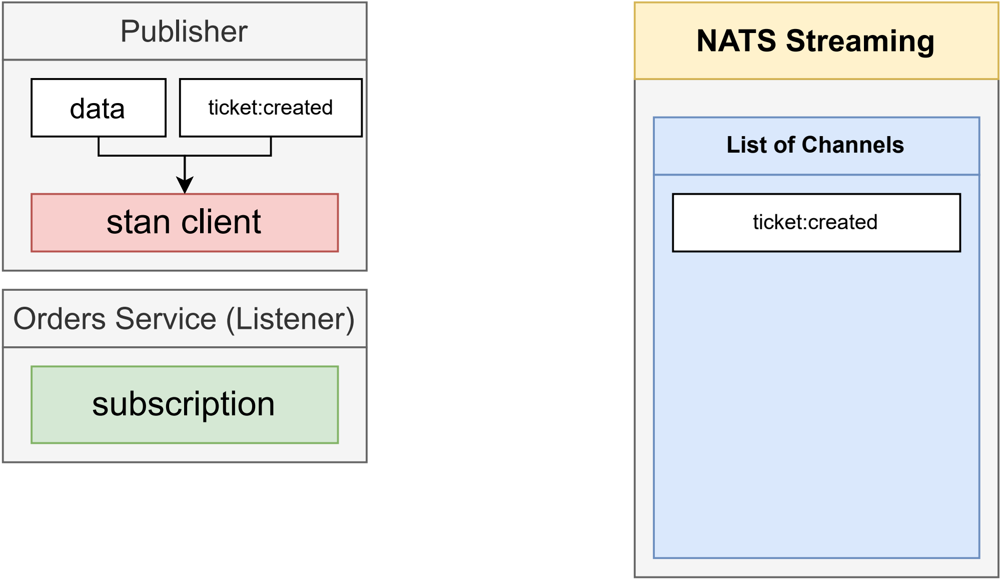
Publishers

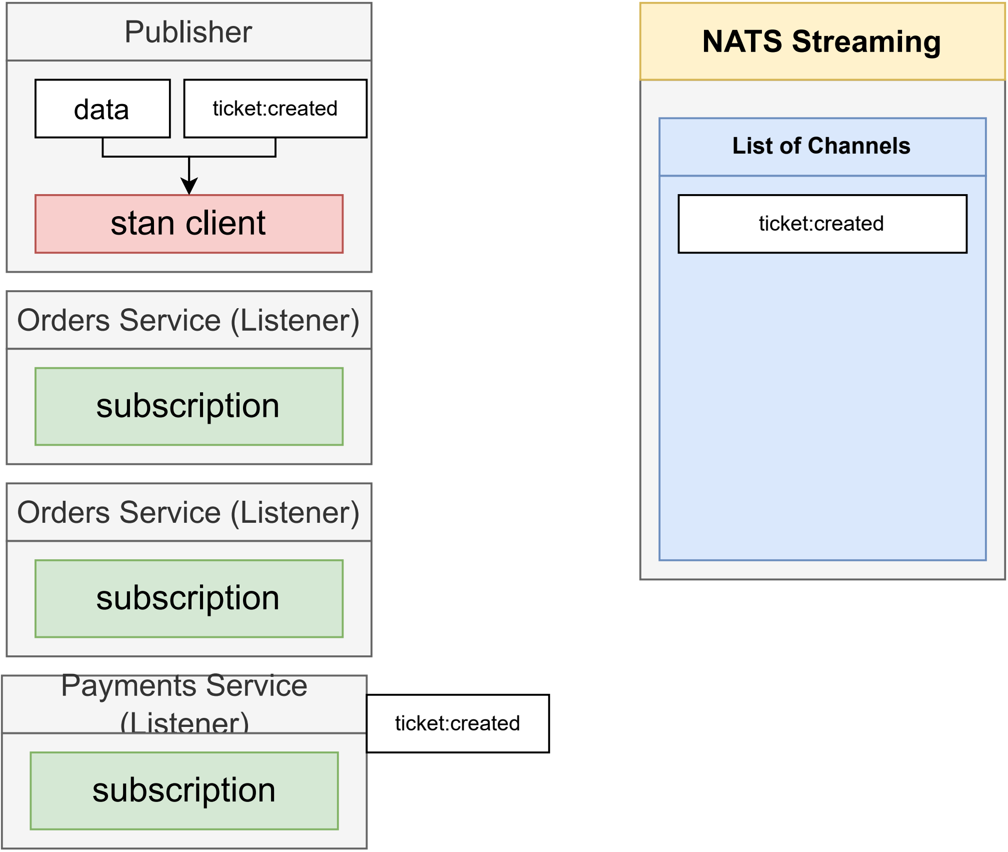
Classes that make it really easy to publish a specific kind of event

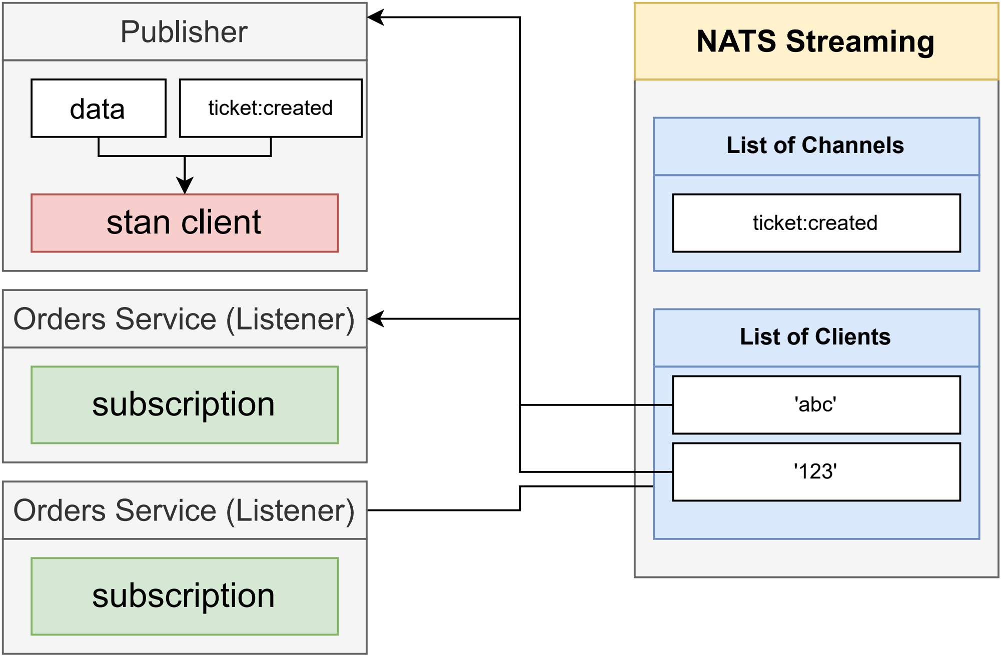
Listeners

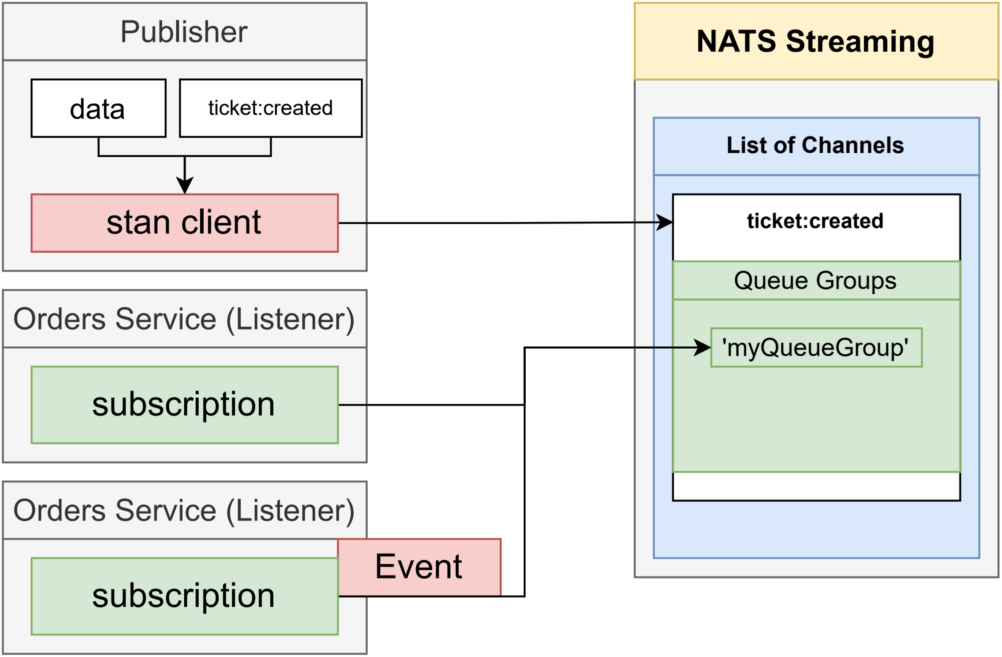
Classes that make it really easy to listen to specific events

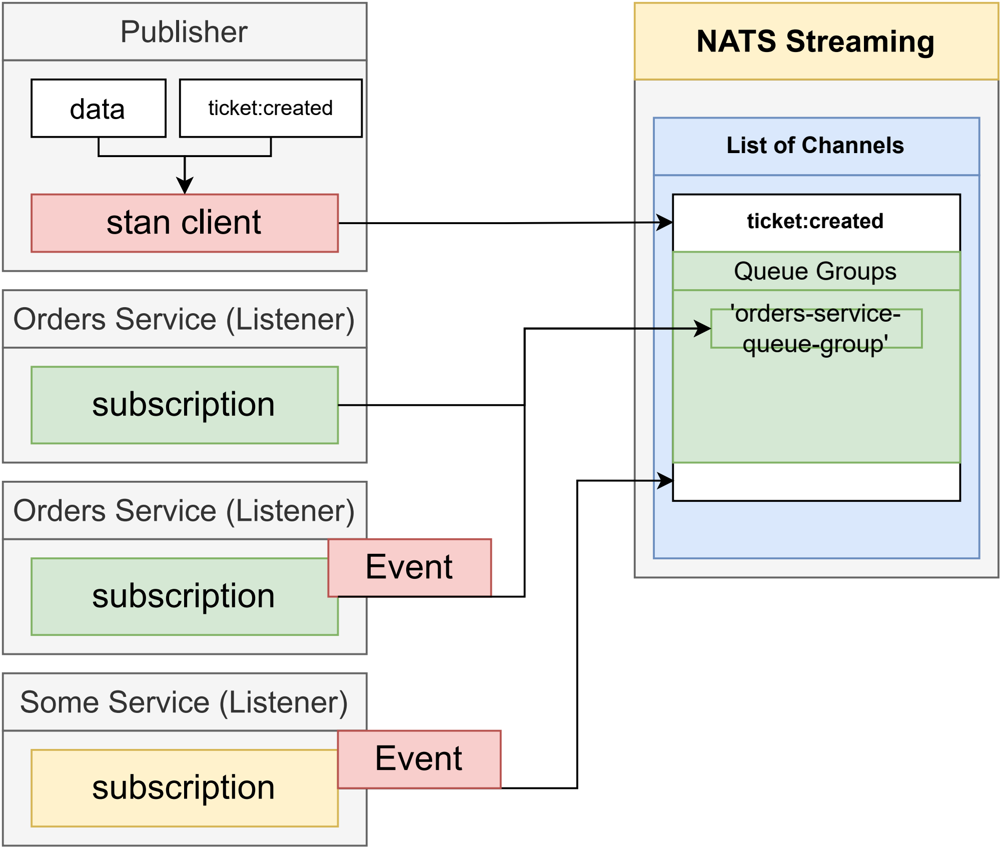

Publisher
data
ticket:created
stan client
Listener
ticket:created
stan client
subscription
NATS Streaming
List of Channels
ticket:created

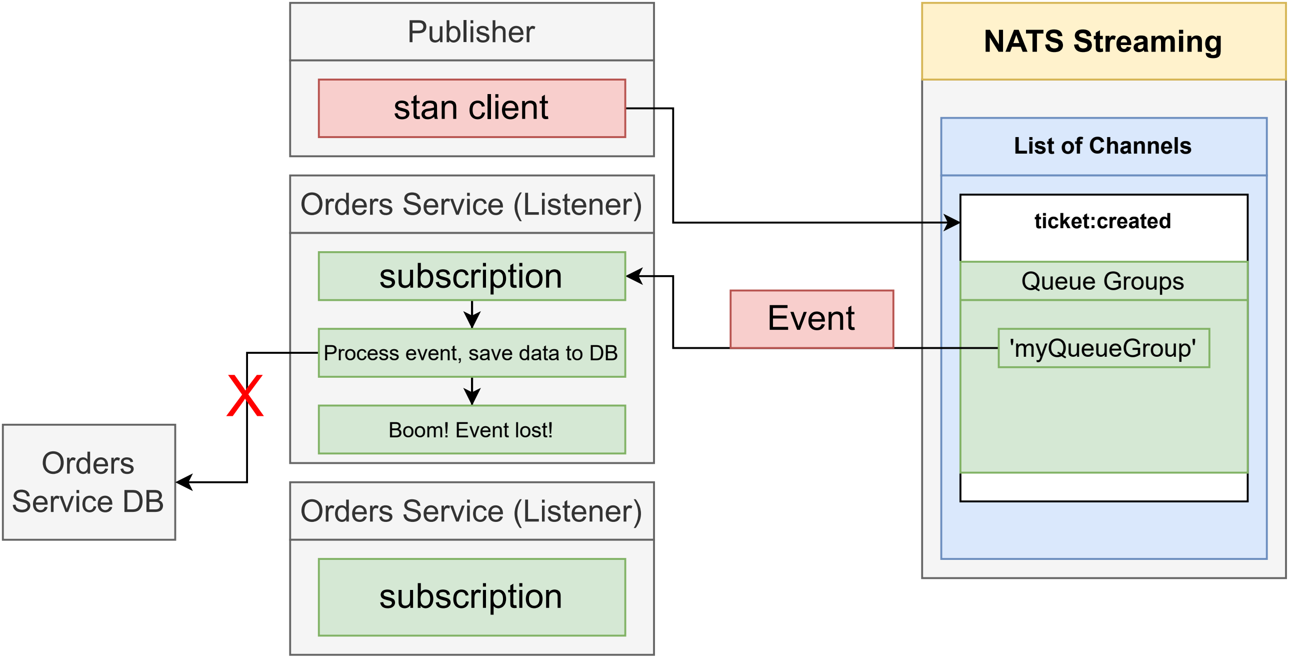

Publisher
data
ticket:created
stan client
Orders Service (Listener)
subscription
NATS Streaming
List of Channels
ticket:created

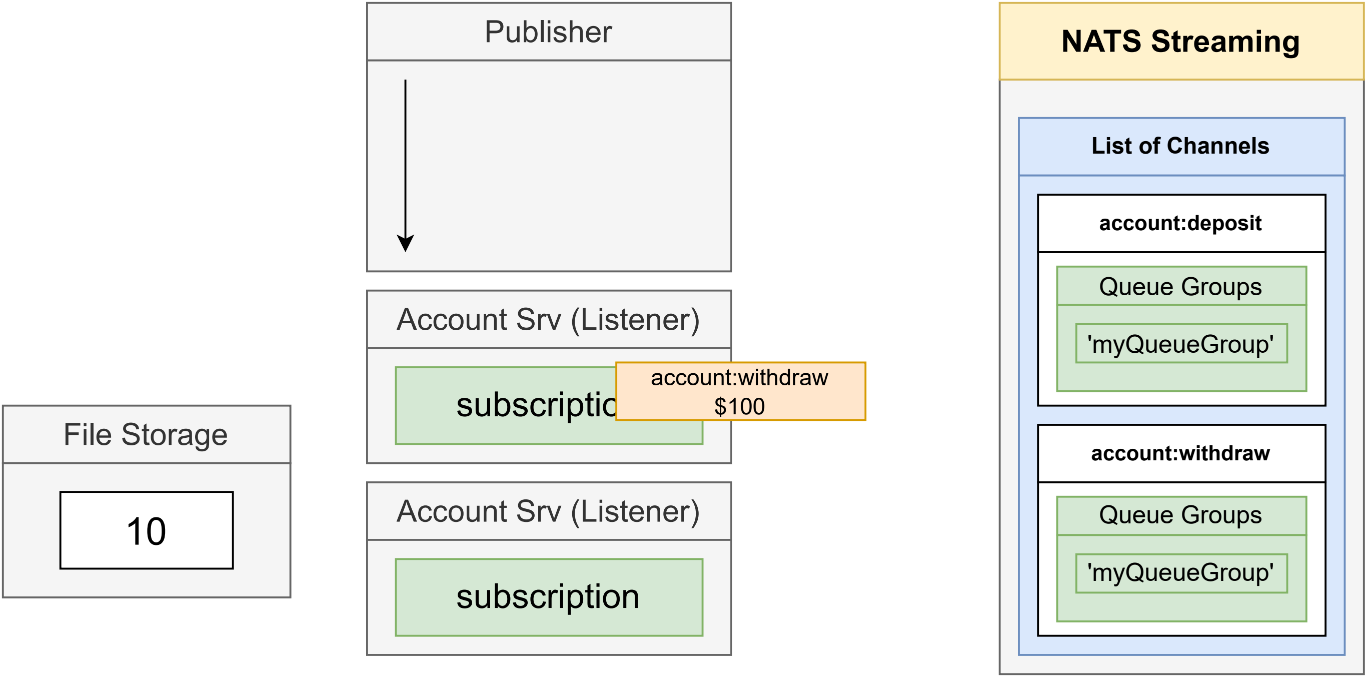

Publisher
data
ticket:created
stan client
Orders Service (Listener)
subscription
Orders Service (Listener)
subscription
Payments Service (Listener)
subscription
ticket:created
NATS Streaming
List of Channels
ticket:created


Publisher
data
ticket:created
stan client
Orders Service (Listener)
subscription
Orders Service (Listener)
subscription
NATS Streaming
List of Channels
ticket:created
List of Clients
'abc'
'123'

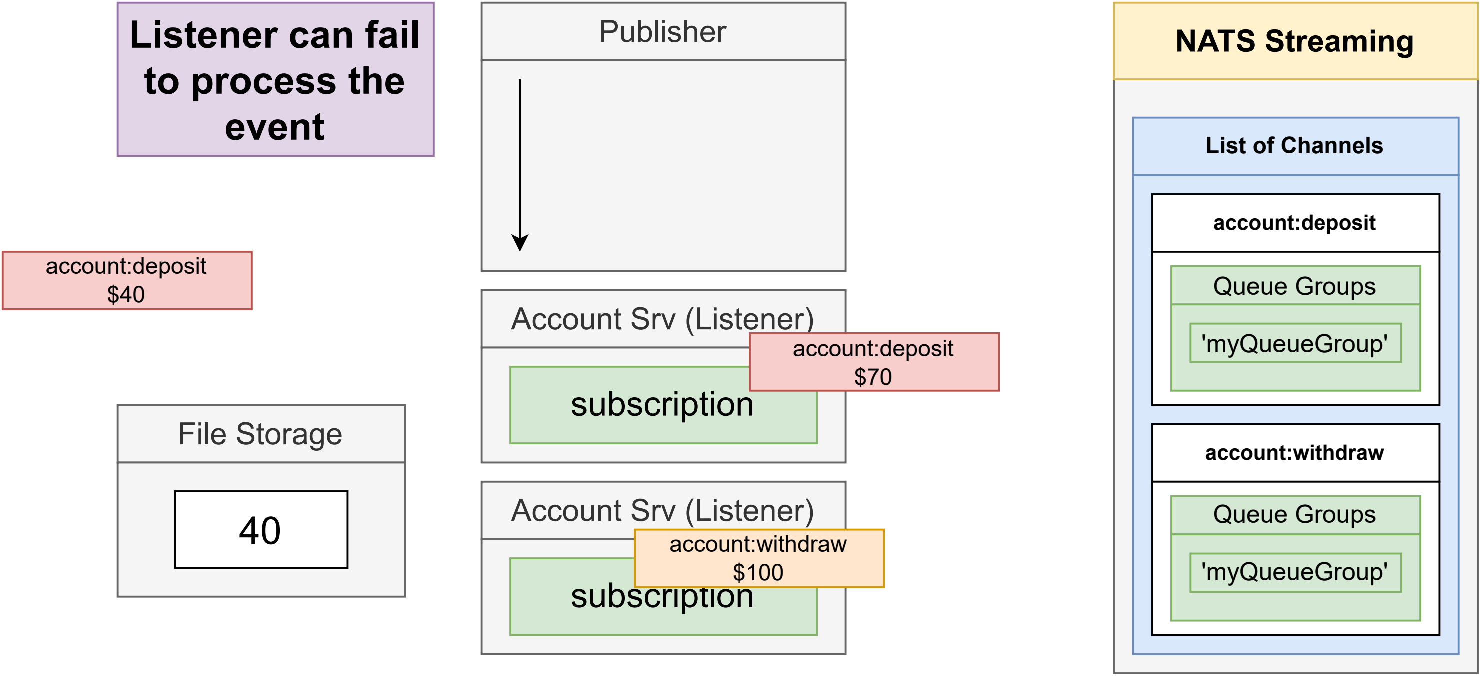

Publisher
data
ticket:created
stan client
Orders Service (Listener)
subscription
Orders Service (Listener)
subscription
Event
NATS Streaming
List of Channels
ticket:created
Queue Groups
'myQueueGroup'


Publisher
data
ticket:created
stan client
Orders Service (Listener)
subscription
Orders Service (Listener)
subscription
Event
Some Service (Listener)
subscription
Event
NATS Streaming
List of Channels
ticket:created
Queue Groups
'orders-service-queue-group'

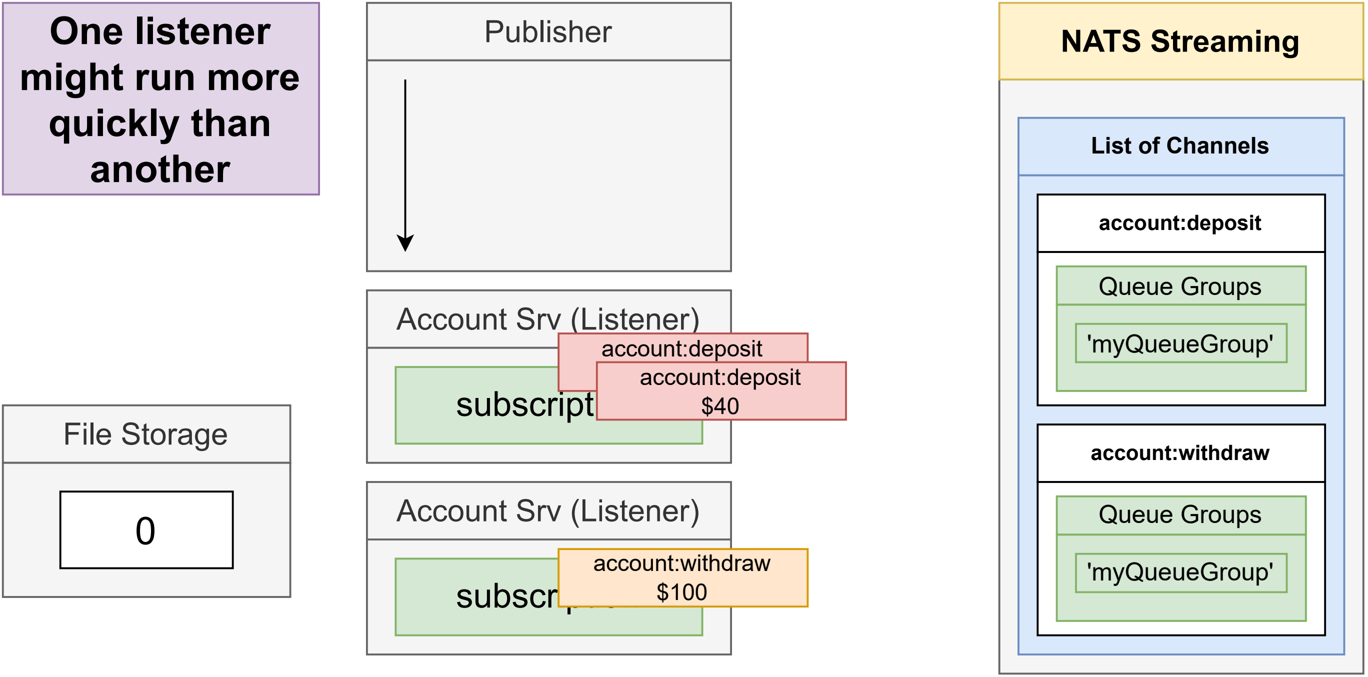

Publisher
stan client
Orders Service (Listener)
subscription
Process event, save data to DB
Boom! Event lost!
X
Orders Service DB
Orders Service (Listener)
subscription
Event
NATS Streaming
List of Channels
ticket:created
Queue Groups
'myQueueGroup'


Publisher
Account Srv (Listener)
subscriptio
account:withdraw
$100
Account Srv (Listener)
subscription
File Storage
10
NATS Streaming
List of Channels
account:deposit
Queue Groups
'myQueueGroup'
account:withdraw
Queue Groups
'myQueueGroup'

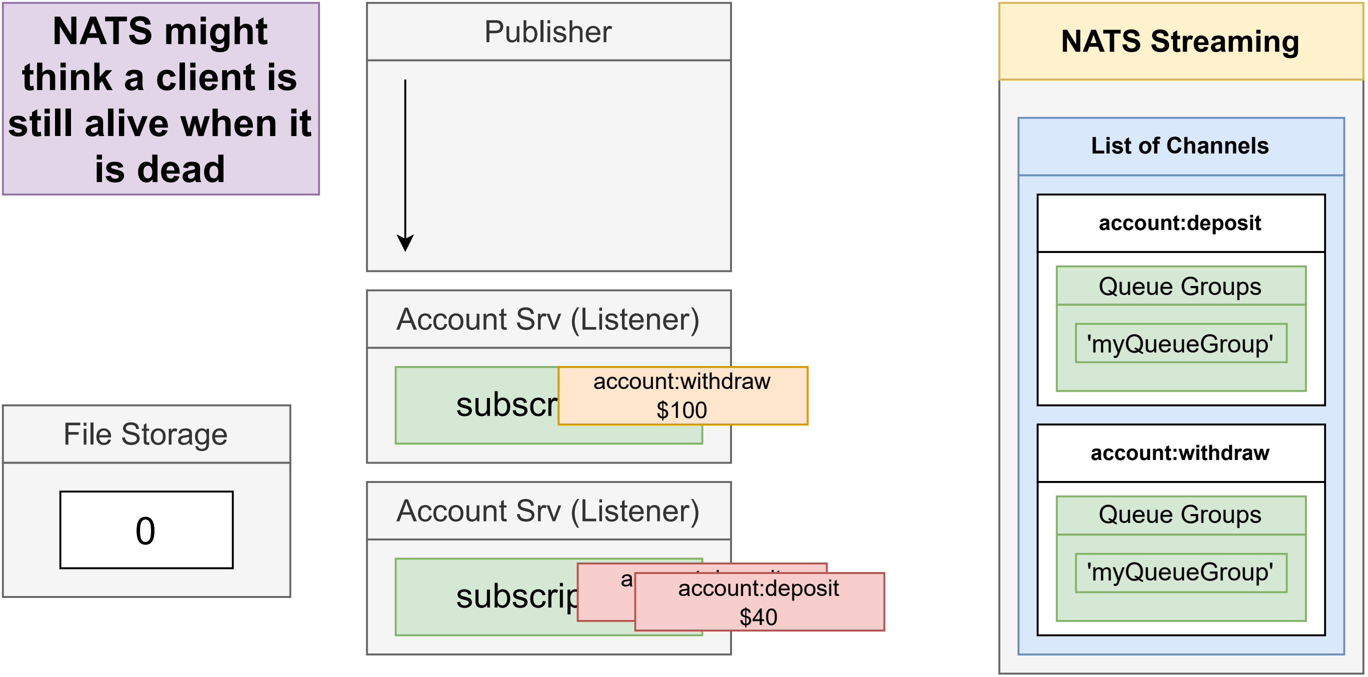
Listener can fail to process the event



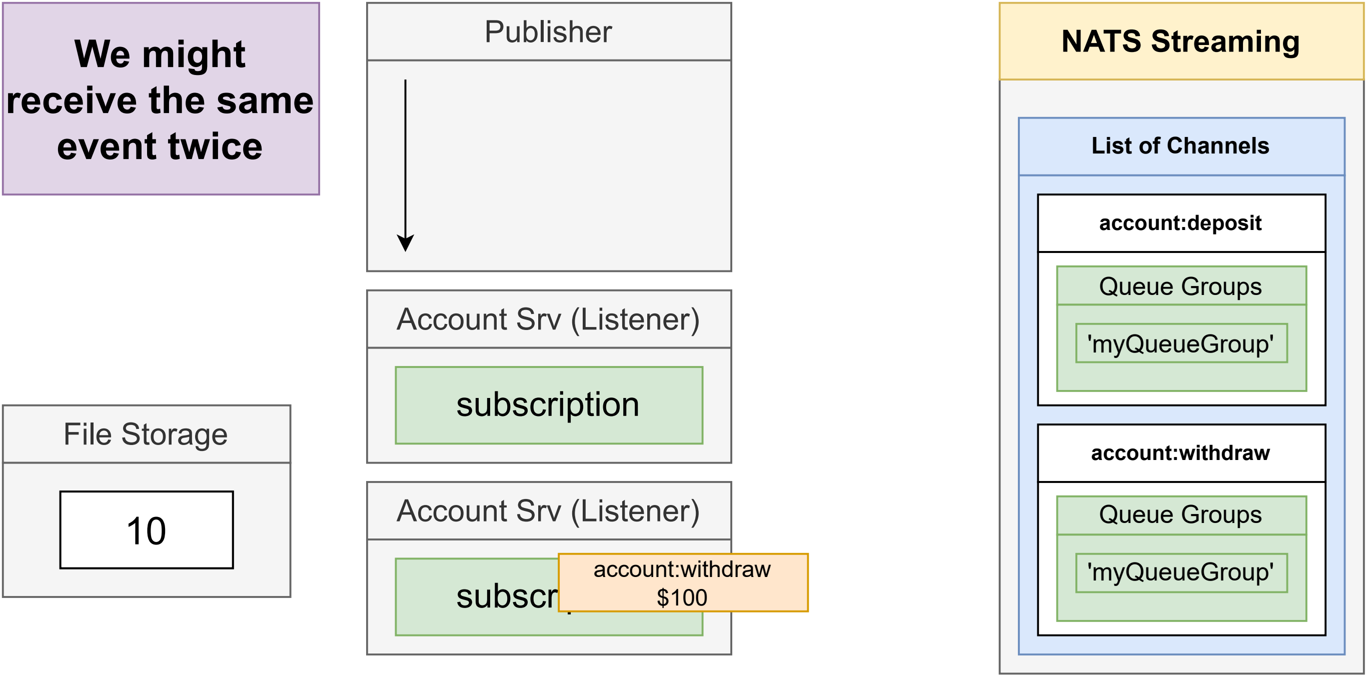
One listener might run more quickly than another

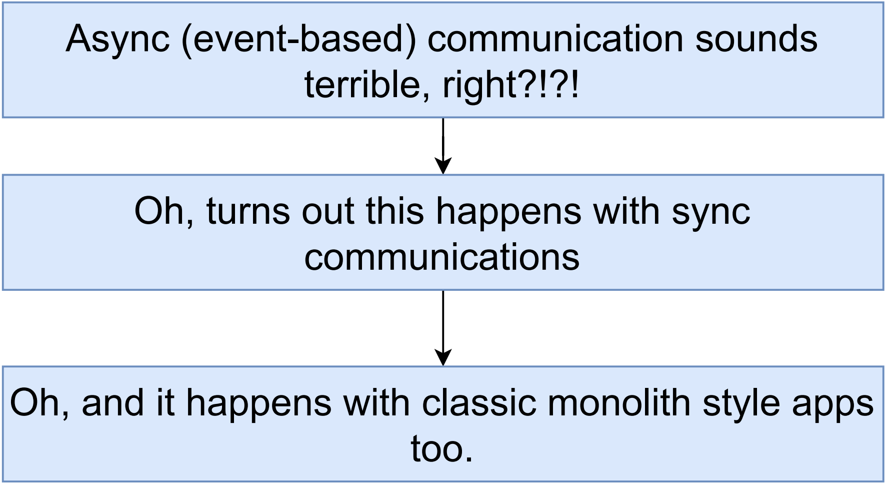


NATS might think a client is still alive when it is dead



We might receive the same event twice




Async (event-based) communication sounds terrible, right?!?!
Oh, turns out this happens with sync communications
Oh, and it happens with classic monolith style apps too.