

Data Contracts in the Modern Data Stack

Zack Klein  **whatnot**

What is Whatnot?



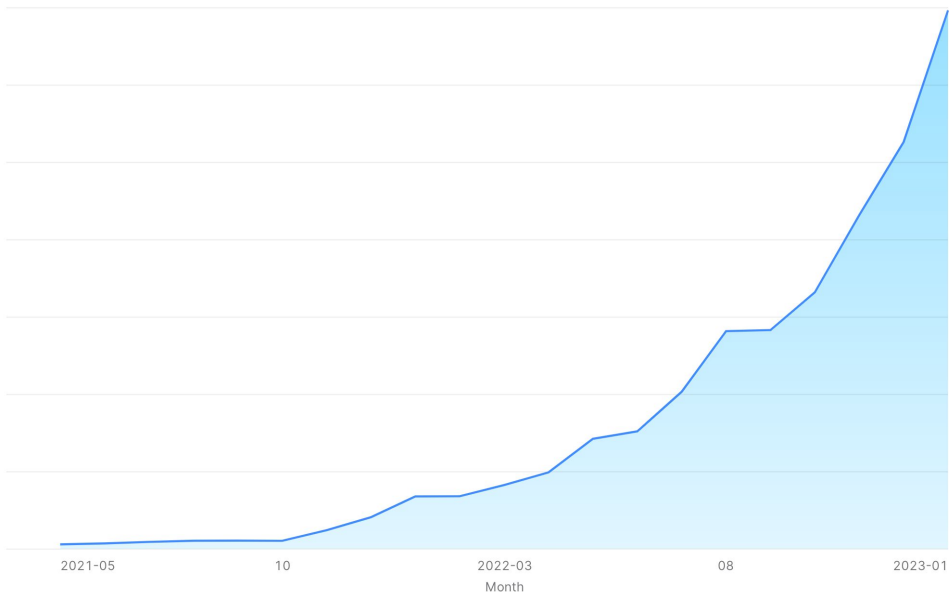
- Livestream & async e-commerce platform
- Collectibles and community-driven markets
- [Fastest growing marketplace in the United States.](#)



Our data stack

- We write a lot – check out our [blog](#)
- Data stack v3 – Snowflake, AWS, Dagster, DBT, Segment, Kafka.
- Started ~2 years ago when the company was ~20 employees. Now we are ~400!
- Today we'll be talking about **data contracts in our events** system.
- Over the past ~2 years, the number of events we've sent per month has increased **over 100x**, with an average MoM increase of **~30%**

Events sent from Whatnot Systems



The problems

- Major problems:
 - Wild west – inconsistent/huge number of tables, fields, etc.
 - Starting from scratch each time we wanted to make new events.
 - No ownership/accountability – difficult and slow to fix issues.
- Whatnot is super data-driven – we saw this as a big risk, and decided to invest heavily to improve it.

<input type="checkbox"/> ACTIVITIES_TAB_TAP	<input type="checkbox"/> DROPS_TAB_TAP	<input type="checkbox"/> LIVE_ITEM_LIST_AUCTION_TAP
<input type="checkbox"/> APPLICATION_BACKGROUNDED	<input type="checkbox"/> DROP_SHARE_TAP	<input type="checkbox"/> LIVE_ITEM_LIST_AVAILABLE_TAP
<input type="checkbox"/> APPLICATION_INSTALLED	<input type="checkbox"/> EXPLORE_FEED_CLOSED	<input type="checkbox"/> LIVE_ITEM_LIST_BUY_NOW_CONFIRM_TAP
<input type="checkbox"/> APPLICATION_OPENED	<input type="checkbox"/> EXPLORE_FEED_OPENED	<input type="checkbox"/> LIVE_ITEM_LIST_BUY_NOW_TAP
<input type="checkbox"/> APPLICATION_UPDATED	<input type="checkbox"/> GO_LIVE_TAP	<input type="checkbox"/> LIVE_ITEM_LIST_CANCEL_TAP
<input type="checkbox"/> APP_OPEN	<input type="checkbox"/> GRADING_DESCRIPTION_ADDED	<input type="checkbox"/> LIVE_ITEM_LIST_GIVEAWAY_TAP
<input type="checkbox"/> APP_STORE_UPSELL_SHOWN	<input type="checkbox"/> GRADING_DESCRIPTION_UPDATED	<input type="checkbox"/> LIVE_ITEM_LIST_PURCHASED_TAP
<input type="checkbox"/> CALENDAR_ADD_TAP	<input type="checkbox"/> GRADING_SUBMITTED	<input type="checkbox"/> LIVE_ITEM_LIST_SEARCH_TAP
<input type="checkbox"/> CALENDAR_PERMISSION_DENIED	<input type="checkbox"/> GRADING_TOGGLE	<input type="checkbox"/> LIVE_ITEM_LIST_SOLD_TAP
<input type="checkbox"/> CARET_TAP	<input type="checkbox"/> HOME_TAB_CATEGORIES_TAP	<input type="checkbox"/> LIVE_ITEM_LIST_TAP
<input type="checkbox"/> CATEGORY_EXPAND_TAP	<input type="checkbox"/> HOME_TAB_COUNTRY_TAP	<input type="checkbox"/> LIVE_PAYMENT_ADDRESS_CANCEL_TAP
<input type="checkbox"/> CATEGORY_FOLLOW_TAP	<input type="checkbox"/> HOME_TAB_FOR_YOU_TAP	<input type="checkbox"/> LIVE_PAYMENT_ADDRESS_SAVE_TAP
<input type="checkbox"/> CATEGORY_TAP	<input type="checkbox"/> HOME_TAB_TAP	<input type="checkbox"/> LIVE_PAYMENT_ADDRESS_TAP
<input type="checkbox"/> CATEGORY_UNFOLLOW_ALERT_CANCEL_TAP	<input type="checkbox"/> IDENTIFIES	<input type="checkbox"/> LIVE_PAYMENT_PAYMENT_SELECT_TAP
<input type="checkbox"/> CATEGORY_UNFOLLOW_ALERT_IMPRESSION	<input type="checkbox"/> IMPRESSION	<input type="checkbox"/> LIVE_PAYMENT_TAP
<input type="checkbox"/> CATEGORY_UNFOLLOW_GOT_IT_TAP	<input type="checkbox"/> INSTALL_ATTRIBUTED	<input type="checkbox"/> LIVE_PINNED_BUY_NOW_CONFIRM_TAP
<input type="checkbox"/> CHAT_TAGGING_SETTING_TAP	<input type="checkbox"/> INVITE_FROM_CONTACTS_TAP	<input type="checkbox"/> LIVE_PINNED_BUY_NOW_TAP
<input type="checkbox"/> CHAT_TAGGING_SET_SETTING_TAP	<input type="checkbox"/> LIFECYCLE_V2	<input type="checkbox"/> LIVE_QUICK_ADD_CATEGORY_TAP
<input type="checkbox"/> CLIENT_SYSTEM_MEASUREMENT_EVENT	<input type="checkbox"/> LISTING_SAVE_TAP	
<input type="checkbox"/> CLIP_SHARE_TAP		

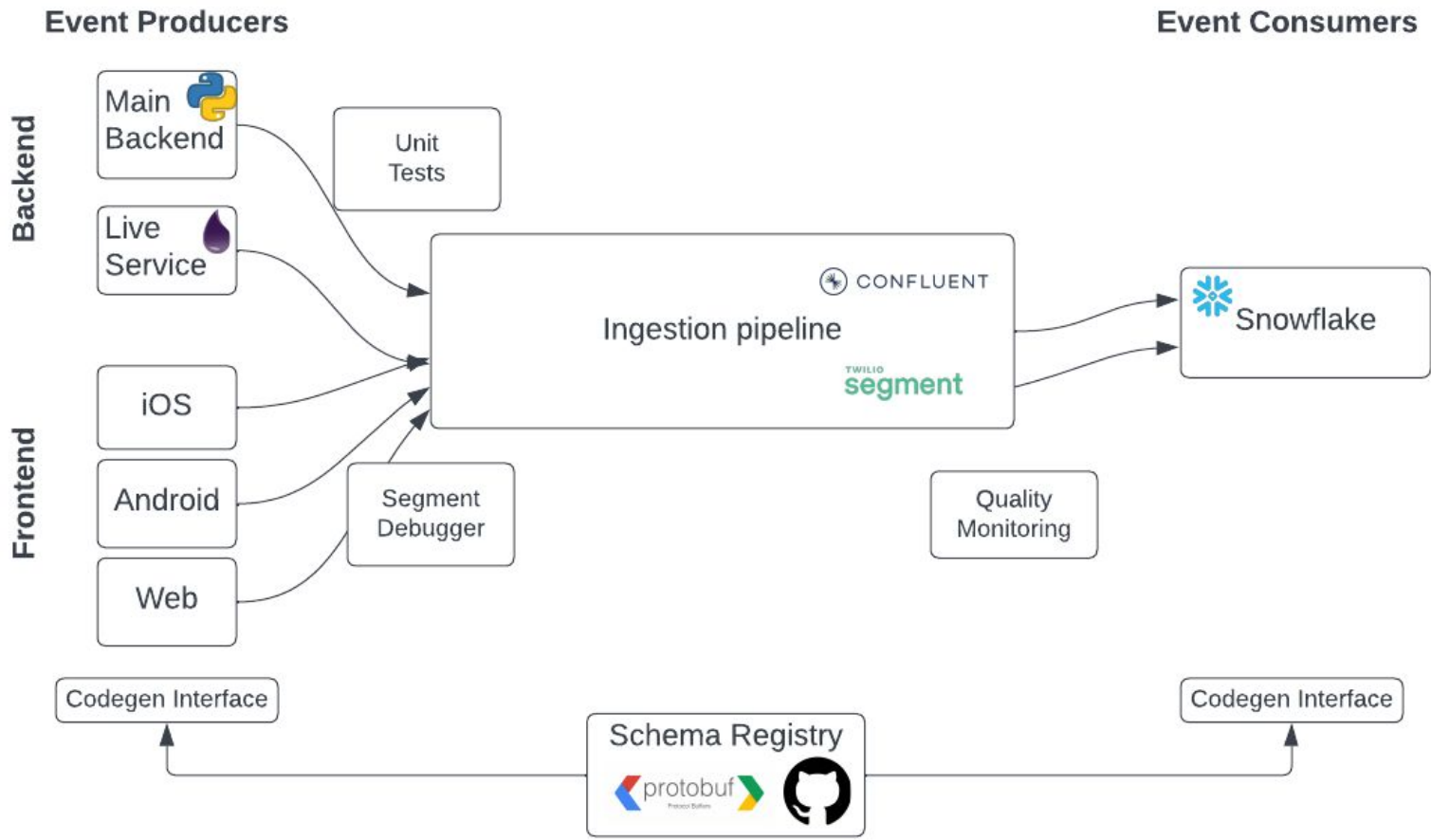


What we did

A complete overhaul of the events system

- Significant investment in the unification of our events into one “data highway”.
- Conformed all existing events into *Actor Action Object* model
- We accomplished this by building 4 components:
 - Interface
 - Schema
 - Pipeline
 - Exposure
- Leaned heavily on code generation and PR review process to allow us to move fast with high quality.





Example: a user follows another user



Step 1: Declare schema

```
message UserFollowedUserEvent {  
    google.protobuf.Int64Value follower_id = 1;  
    google.protobuf.Int64Value followee_id = 2;  
}
```

```
UserFollowedUserEvent user_followed_user = 17 [  
    (event_metadata).description.value = "Fires when a user starts following other user.",  
    (event_metadata).team_owner.value = "Foundations"  
];
```



Step 2: Implement producer

```
def user_followed_user(  
    *,  
    follower_id: int,  
    followee_id: int,  
):  
    event = UserFollowedUserEvent()  
    set_nullable_value(event.follower_id, follower_id)  
    set_nullable_value(event.followee_id, followee_id)
```

```
user_followed_user(follower_id=user_id, followee_id=user_id_to_follow)
```



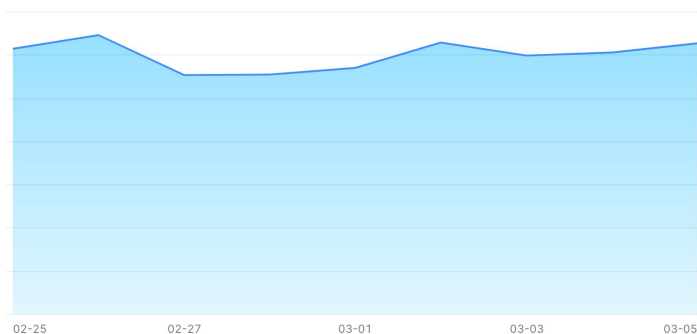
Step 3: Query

☐ BACKEND_EVENTS

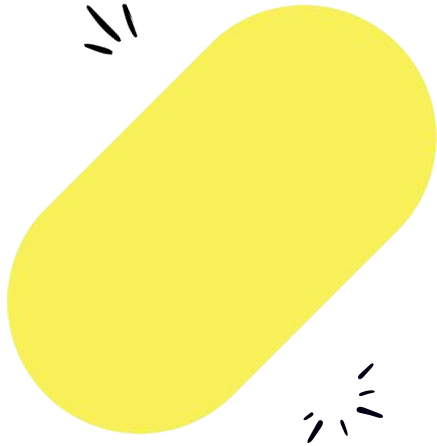
☐ FRONTEND_EVENTS

```
select *  
from backend__events  
where event_name = 'user_followed_user'  
limit 10;
```

```
select date_trunc(day, event_timestamp) dt  
      , count(*)  
from backend__events  
where 1=1  
and event_name = 'user_followed_user'  
and event_timestamp >= '2023-02-25'  
and event_timestamp < sysdate()::date  
group by 1  
order by 1 desc;
```



Maintaining quality



We catch errors at various stages of development:

- **Before anything is implemented**
 - Automated checks (using Python unit tests and open source protobuf tooling) on the schema to ensure backwards compatibility, schema validity, naming conventions, etc..
- **During development**
 - Easy path to write unit tests asserting semantic quality of the data.
- **After we've shipped**
 - In-flight monitoring using DataDog metrics and alerting on schema issues.
 - QA checks that run in the warehouse using custom observability tooling.



Learnings

- Code generation is A Good Thing.
- The earlier the better.
- The more focus on the users of the system (event producers and the data consumers), the better.



Thank you!

