

Component and flows

Please direct specific questions to people named below. Please also refer to technical expertise areas [owners](#).
If all else fails, reach out to [Nishi Ningegowda](#) or [Eric Sellin](#)

What each component does, how they communicate, what is the communication from the customer point of view (each component within the cluster)?

We have a number of services running inside a EKS k8s cluster:

- **Monolith** - Ruby on Rails (RoR) application - provides both internal and external APIs, as well as the Event Dashboard used by event organisers. Also responsible for authentication of attendees using the platform. Data storage via Aurora PostgreSQL. Caching via RedisLabs. Job scheduling via Sidekiq Enterprise. Produces messages (attendee activity) to AWS MSK Kafka bus.
- **Analytics** - [Turerkan Ince](#) - RoR application - consumes messages from Kafka bus + data points from attendee web/mobile apps to aggregate in separate database (also Aurora PostgreSQL) - keeps track of time spent by attendees in various parts of an event - generates CSV reports upon request from monolith - provides data API to render analytics metrics/charts inside Event Dashboard.
- **Registrations** - [Paco Sanchez](#) - React webapp - allows event organisers to configure ticketing for their events, and renders the registration widget allowing attendees to register for events
- **App store** - React webapp - allows event organisers to install and configure 3rd-party apps for their events (e.g. polls, audio translations, Q&A, etc)
- **Page Builder (aka Canvas)** - React webapp - allows event organisers to create custom landing page for their events with a rich-text editor - renders landing page for attendees before they register for an event
- **Captions** - API providing client-side credentials to use Azure translations service - will be merged with monolith due to small size
- **Longwave** - calls webhooks to provide event organisers with real-time attendee activity in their event (e.g. registrations)
- **Onsite** - [Gurel Kaynak](#) - Python (Django/DRF) application - consumes/produces messages from/to Kafka bus - storage in separate database (also Aurora PostgreSQL) - provides API for our Organizer mobile application, allowing QR code scanning badge printing for physical events. Provides API for our Attendee App allowing lead retrieval scanning.
- **Live** - Rust application - real-time tracking for online presence of attendees - powers the "green dot" in the attendee webapp and mobile app (see below)
- **GraphQL** - Internal only - provides GraphQL endpoints for the Organizer Dashboard (see below)

We have a number of web properties served via CloudFront distributions + S3 buckets origin

- **Attendee webapp** - [Ross Francis](#) - React webapp used by attendees when they join a virtual event - comprises the following parts:
 - Reception - general info about the event incl. schedules and speakers
 - Stage - watch a live or pre-recorded stream, participate in chat/polls/Q&A, use 3rd-party apps
 - Sessions - similar to Google Meet, but viewers can decide whether or not they want to appear on screen
 - Networking - pairs 2 people over a direct video call to facilitate connection discovery at an event
 - Expo - virtual trade show - each vendor has their own virtual booth where they can share video and chat with attendees
 - Replay - post-event replay of recorded stage or sessions stream
- **Organiser dashboard** - [Ben Embery](#) - React webapp - allows event organisers to manage their account (incl. team members and billing) and create new events
- **Event dashboard** - [Ben Embery](#) - React webapp - this is gradually replacing the legacy Event Dashboard implemented in the RoR monolith - we use reverse-proxying in Istio in order to serve selected content from this webapp instead of the monolith
- **Pre-event check** - [Herson Lopez](#) - provides connectivity and A/V quality checks for speakers participating in streams
- **Slices** - some of our FE code is split into "micro front-ends", i.e. pieces of functionality that can be built and deployed separately

We have a web property that is served through cloudflare workers

- **Checkout** - [Ben Embery](#) - React webapp (Remix) used by customers that want to purchase or upgrade our lower-tier self serve plans

We have a separate EKS k8s cluster dedicated to video and AI:
More info can be found [here](#)

- **MAR/MTS** - [Yuriy Orlov](#) - Media Assets Registry / Media Transform Service - storage/transform of media assets (videos, captions, manifests) - will be combined into 1 service in the future
- **RTC** - [Gustavo Garcia](#) - WebRTC-based audio/video stack used to power sessions
- **AI** - [Gustavo Garcia](#) - AI based API powering AI features in Events

We use a mix of Fargate/Lambda/EC2 for the following:

- **Video API and mixer** - [Armando Garcia-Mendoza](#) - creates recordings from stages or sessions (web browser screen capture)
More information available [here](#)

We have 2 mobile apps, each available on both iOS and Android:

- **Attendee mobile app** - [Kirami Kacan](#) (iOS) and [Mustafa Berk](#) (Android) - replicates the functionality of the attendee webapp above - allows attendees to join events, watch stage streams, participate in sessions, use chat/Q&A/polls, etc
- **Organizer mobile app** - [Kirami Kacan](#) (iOS) and [Mustafa Berk](#) (Android) - allows staff to scan attendee QR codes and print badges for onsite (physical) events

More information: [Events Technical Overview](#)



hopin-erd.pdf

Hopin Analytics



hopin-analytics-erd (1).pdf

Video API

