OPENREC.tv における ライブ動画&メッセージ 配信基盤の全貌

AWS DevDay Tokyo 2017

Today's Contents

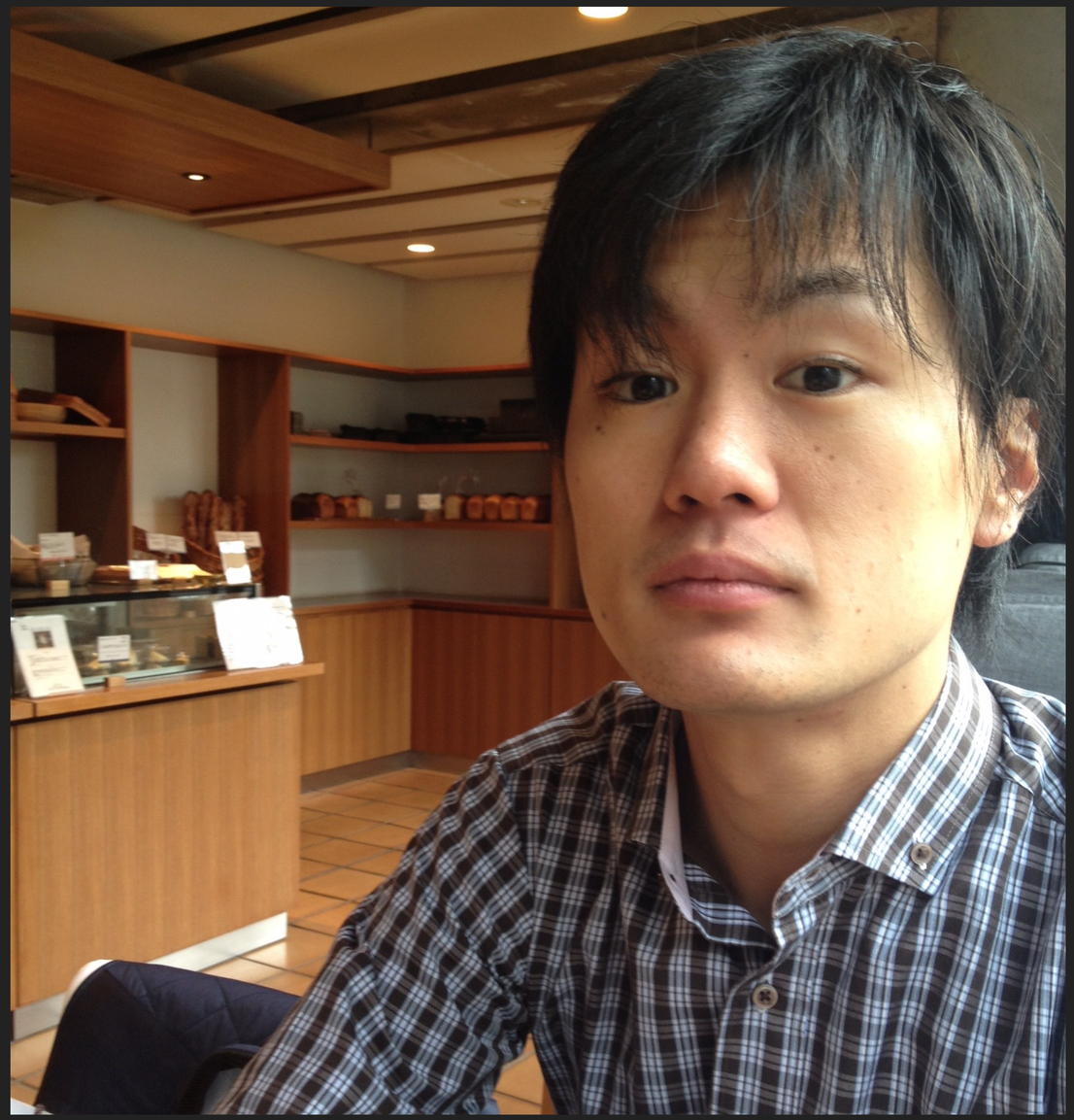
本セッションを通じて「ライブ」に求められるアーキテクティング をご紹介します

- OPENREC.tv とは
- 動画配信の基本
- 動画配信基盤のアーキテクチャと変遷
- チャットの基本
- リアルタイムメッセージ配信基盤のアーキテクチャと変遷
- サービスの成長を加速させるために

whoami

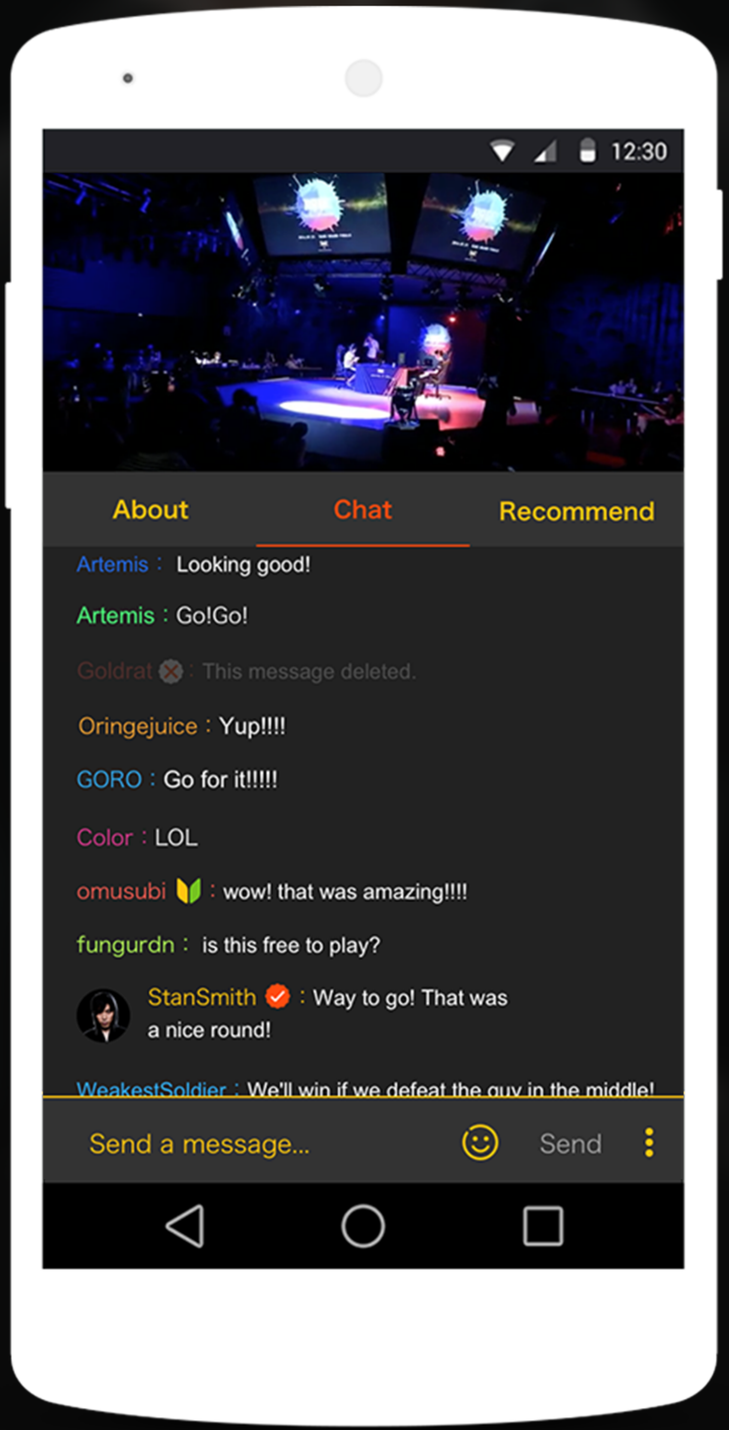
廣瀬 太郎 / Taro Hirose

- OPENREC.tv / CyberZ, Inc.
- Backend Engineer
- @uorat
- http://uorat.hatenablog.com



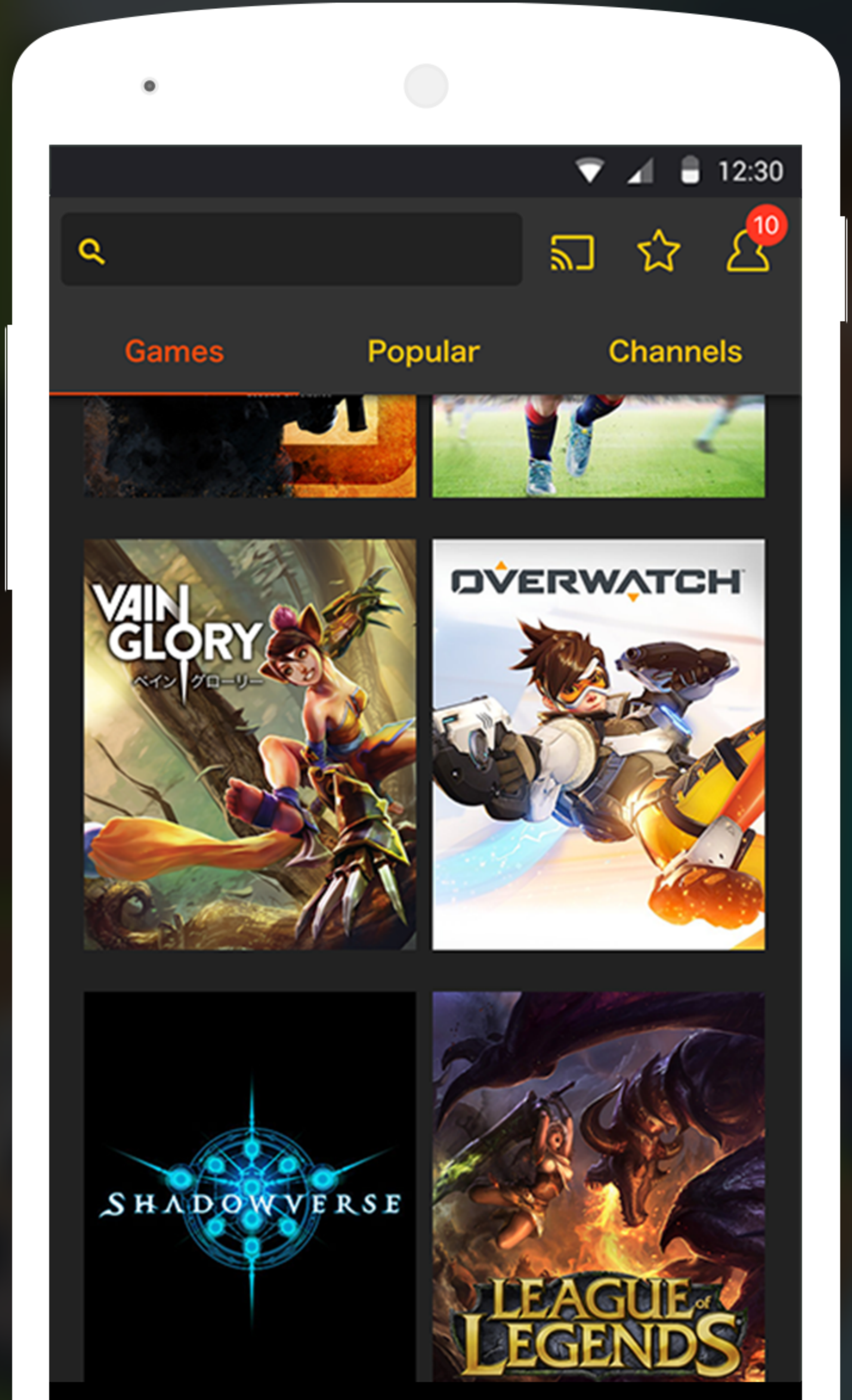
サービス紹介

OPENREC.tv

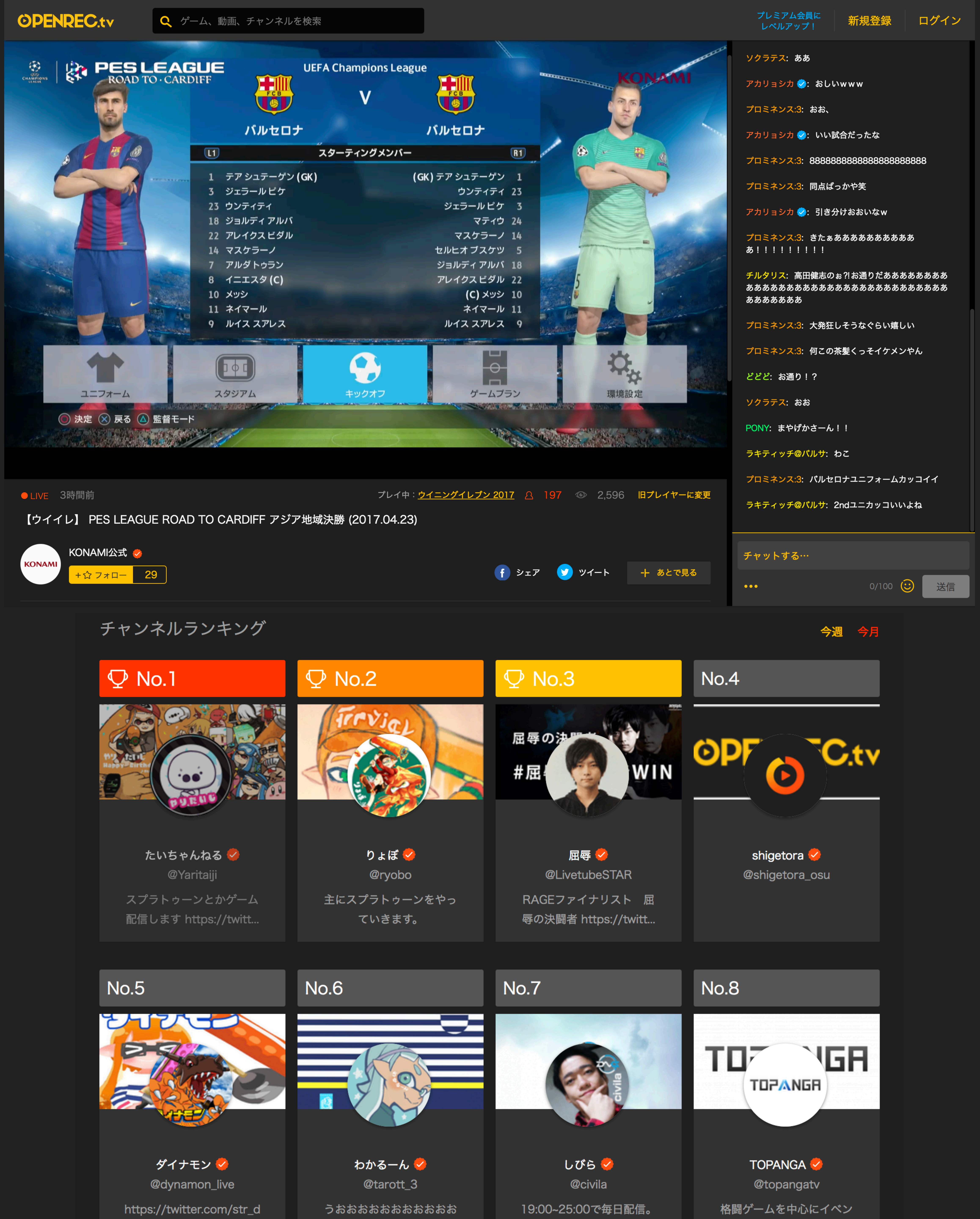


ゲームに特化した動画配信メディア

OPENREC.tv



12:30
10
Games
Popular
Channels
VAINGLORY
ベイングローリー
OVERWATCH
SHADOWVERSE
LEAGUE of LEGENDS



OPENREC.tv
ゲーム、動画、チャンネルを検索
プレミアム会員に レベルアップ！
新規登録
ログイン
PES LEAGUE ROAD TO · CARDIFF
UEFA Champions League
バルセロナ
V
バルセロナ
スターティングメンバー
ユニフォーム
スタジアム
キックオフ
ゲームプラン
環境設定
決定 戻る 監督モード
LIVE 3時間前
プレイ中：ウイニングイレブン 2017 197 2,596 旧プレイヤーに変更
【ウイイレ】 PES LEAGUE ROAD TO CARDIFF アジア地域決勝 (2017.04.23)
KONAMI公式
フォロー 29
シェア
ツイート
あとで見る
チャットする…
0/100
送信
チャンネルランキング
今週
今月
No.1
たいちゃんねる
@Yaritaiji
スプラトゥーンとかゲーム配信します https://twitt...
No.2
りょぼ
@ryobo
主にスプラトゥーンをやっていきます。
No.3
屈辱
@LivetubeSTAR
RAGEファイナリスト 屈辱の決闘者 https://twitt...
No.4
shigetora
@shigetora_osu
No.5
ダイナモン
@dynamon_live
https://twitter.com/str_d
No.6
わかるーん
@tarott_3
うおおおおおおおおおおおお
No.7
しびら
@civila
19:00~25:00で毎日配信。
No.8
TOPANGA
@topangatv
格闘ゲームを中心にイベン

e-sports 大会やオリジナル番組も配信

Service's Growth

DAUは13倍・MAUは110万人を突破

http://gamebiz.jp/?p=180677

OPENRECの急成長　DAUは13倍・MAUは110万人を突破　ライブ配信強化とスマホのeスポーツタイトル登場が契機

2017年03月17日 13時15分更新

209 いいね！　リスト ツイート　3 B!ブックマーク　0 G+1

CyberZ 企業データ

注目ワード： パタパタ!!にゃんこ　B.B.クマ！
パズル＆ドラゴンズ スーパーマリオブラザーズ エディション

RAGE vol.4 GRAND FINAL

2017.06.10 (Sat) 10:00 – 19:30

RAGE
NEW GENERATION E-SPORTS

Today's Contents

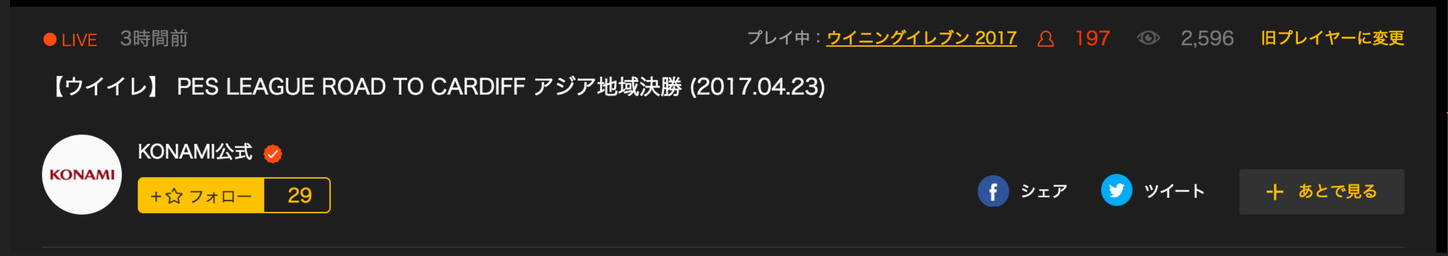
本セッションを通じて「ライブ」に求められるアーキテクティング をご紹介します

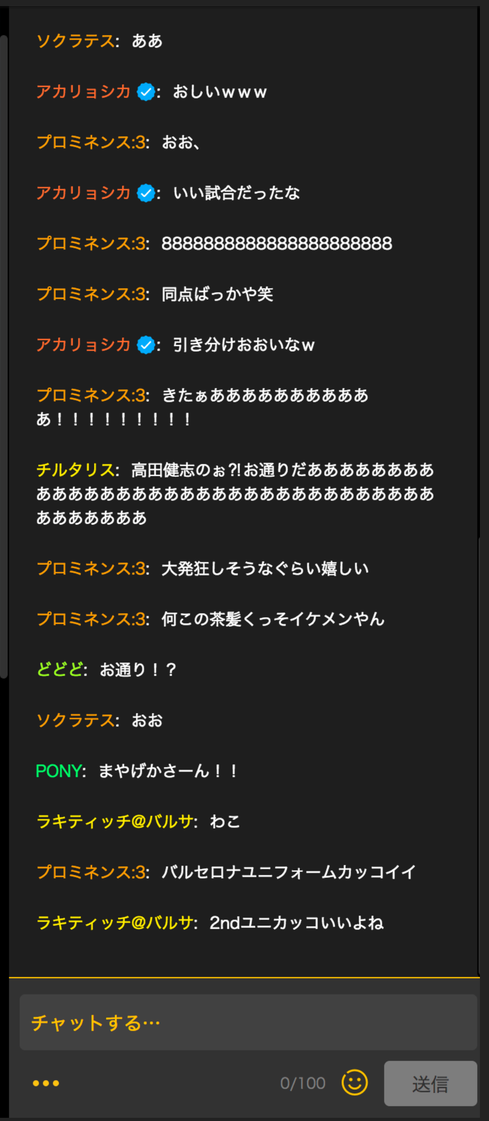
- OPENREC.tv とは
- 動画配信の基本
- 動画配信基盤のアーキテクチャと変遷
- チャットの基本
- リアルタイムメッセージ配信基盤のアーキテクチャと変遷
- サービスの成長を加速させるために

Today's Contents

本セッションを通じて「ライブ」に求められるアーキテクティング をご紹介します

- OPEN
- 動画配
- 動画配
- チャッ
- リアル
- サービ

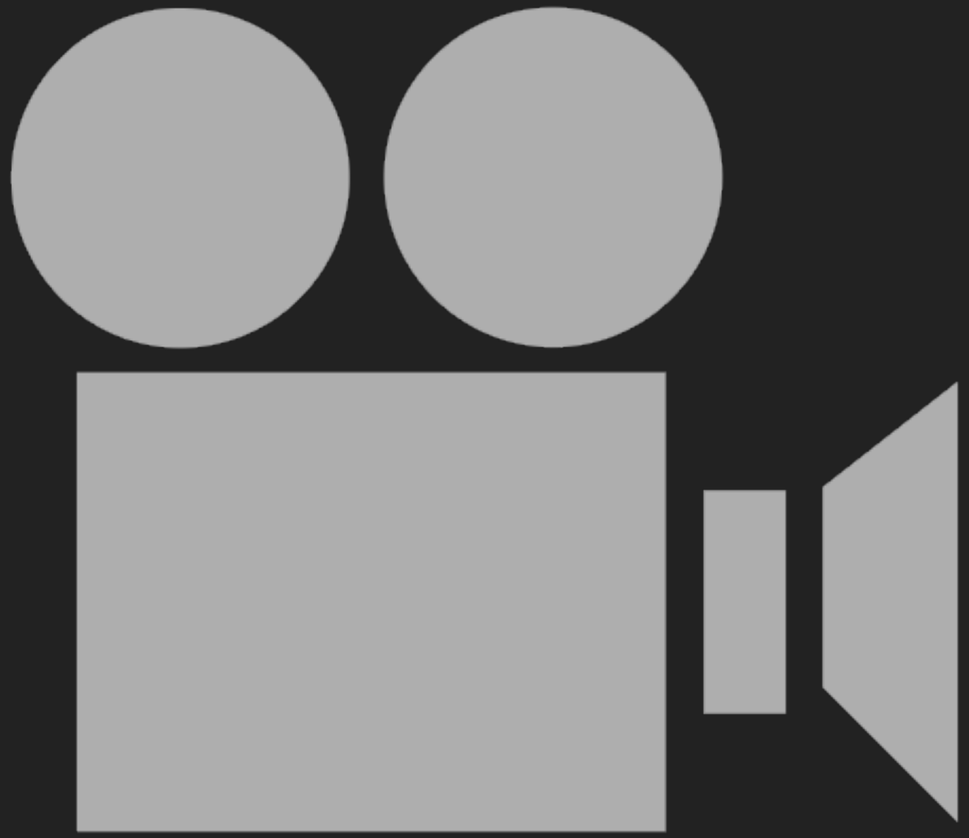




Today's Contents

本セッションを通じて「ライブ」に求められるアーキテクティング をご紹介します

- OPEN
- 動画配
- 動画配
- チャッ
- リアル
- サービ



動画配信の基本

Live Streaming

動画配信

大きく 2 種の用途がある

- オンデマンド配信
  - 視聴者が **見たい時** に動画を視聴することができる配信方法 (abbr. VOD)
    - On-Demand = 「要求に応じて」
  - 早送りや巻き戻しなどの再生制御が可能
  - e.g. 投稿動画、アーカイブ動画 / タイムシフト
- ライブ配信
  - **リアルタイム** に配信されている動画を視聴することができる配信方法
  - リアルタイム視聴なので、基本的に映像の再生制御は行えないことが多い

動画配信

配信方式も大きく 2 種ある

- Progressive Download
  - **一つのまとまったメディアデータ**をダウンロードさせる方式
    - プレイヤーはダウンロードできたところから再生する
  - シンプルだが課題あり
    - **ライブ配信非対応**
    - メディアファイルそのものを直接ダウンロードさせるためコンテンツ保護の観点で難あり

- Streaming
  - **メディアデータを細かく分割**して、順次転送 / ダウンロードさせる方式
    - Progressive Download と似たように、ダウンロードと並行して再生する
  - 視聴中にデータが生成され続ける**ライブ配信に対応可能**
    - 勿論オンデマンド配信も可能

動画配信

配信方式も大きく 2 種ある

- Progressive Download
  - 一つのまとまったメディアデータをダウンロードさせる方式
    - プレイヤーはダウンロードできたところから再生する
  - シンプルだが課題あり
    - ライブ配信非対応
    - メディアファイルそのものを直接ダウンロードさせるためコンテンツ保護の観点で難あり

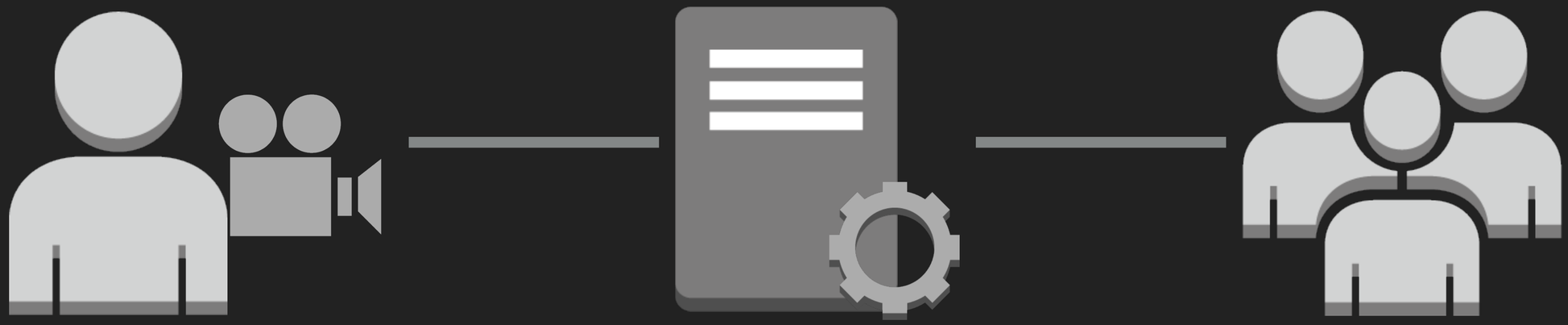
VOD / LIVE いずれも

Streaming 配信がトレンド

- Streaming
  - **メディアデータを細かく分割**して、順次転送 / ダウンロードさせる方式
    - Progressive Download と似たように、ダウンロードと並行して再生する
  - 視聴中にデータが生成され続ける**ライブ配信に対応可能**
    - 勿論オンデマンド配信も可能

Broadcaster / Viewer

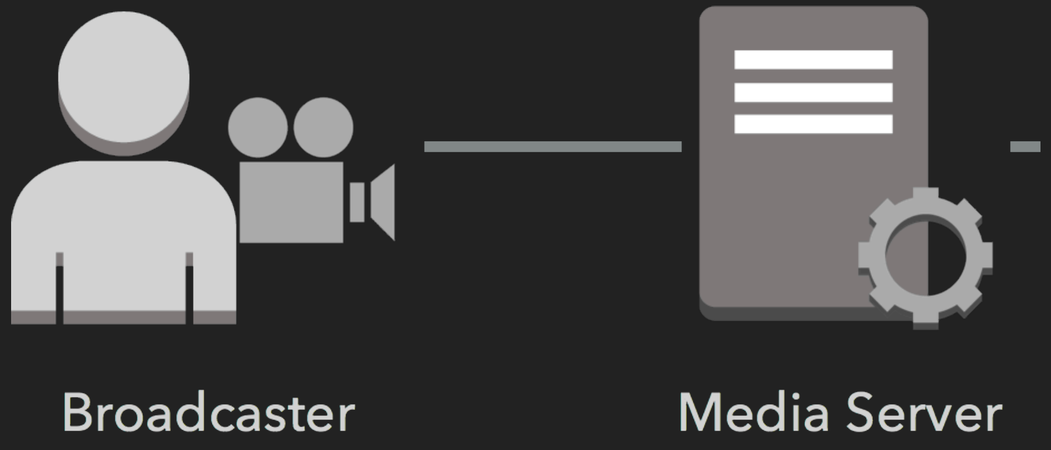
Live Streaming 配信について 配信者側と視聴者側に立って見てみる



Live Streaming: Broadcaster Side

RTMP w/ H.264/AAC の組み合わせが主流

- RTMP = Real-Time Messaging Protocol
  - 持続接続型プロトコル
  - 動画・音声データを分割して転送
- エンコード
  - クライアント側で映像 & 音声データを圧縮
  - 映像 & 音声データの圧縮方式は様々ある
    - 動画コーデック: H.264/MPEG-4 AVC, H.265/HEVC, ...
    - 音声コーデック: MP3, AAC, WMA, ...



Live Streaming: Broadcaster Side

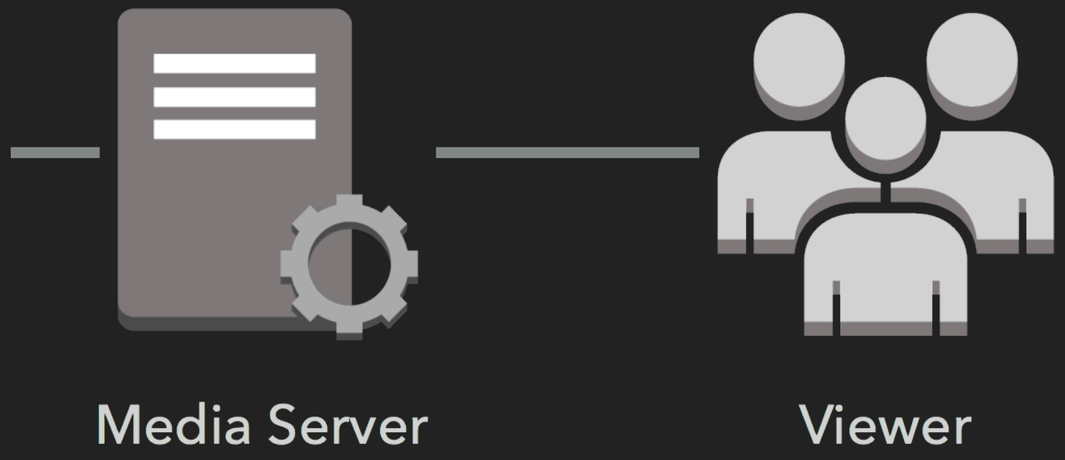
よく使われる配信ソフトウェア

Live Streaming: Viewer Side

HTTP ストリーミング型が主流

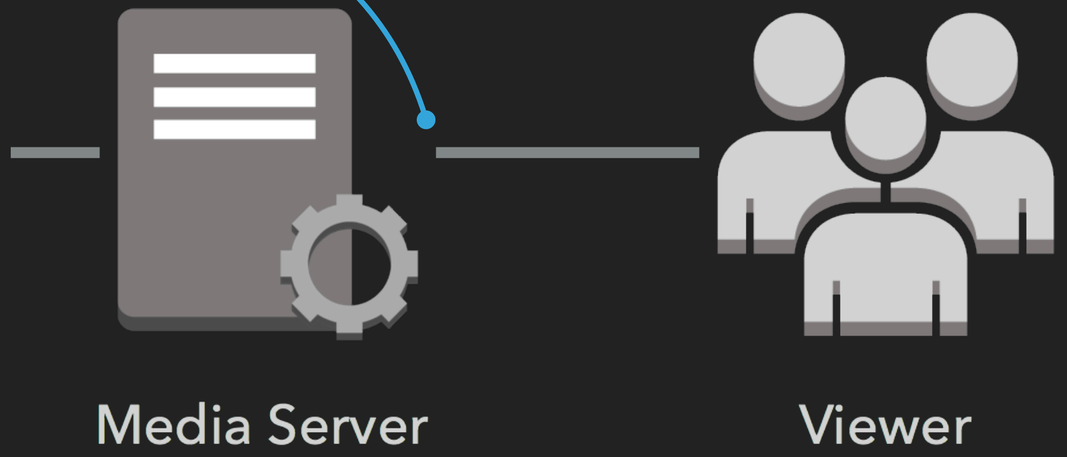
- ストリーミング型
  - 専用プレイヤーやプラグインが必要となる
    - RTMP w/ Adobe Flash
    - RTSP/RTP w/ Windows Media, ReadMedia, QuickTime ...etc
    - MMS w/ Windows Media
  - 大量ユーザーへの配信には大量の Edge Server が必要となる

- **HTTP ストリーミング型**
  - 分割された軽量なメディアファイルを **HTTP** で配信
  - ブラウザ / デバイス標準で再生可能（なことが多い）
    - HLS
    - MPEG-DASH
    - HDS
    - Smooth Streaming
  - **一般的な Caching 技術**を応用可能
    - 言い換えればただの HTTP ファイルダウンロード



Live Streaming: Viewer Side

HTTP ストリーミング型が主流



- **HTTP ストリーミング型**
  - 分割された軽量なメディアファイルを **HTTP** で配信
  - ブラウザ / デバイス標準で再生可能（なことが多い）
    - HLS
    - MPEG-DASH
    - HDS
    - Smooth Streaming
  - **一般的な Caching 技術**を応用可能
    - 言い換えればただの HTTP ファイルダウンロード

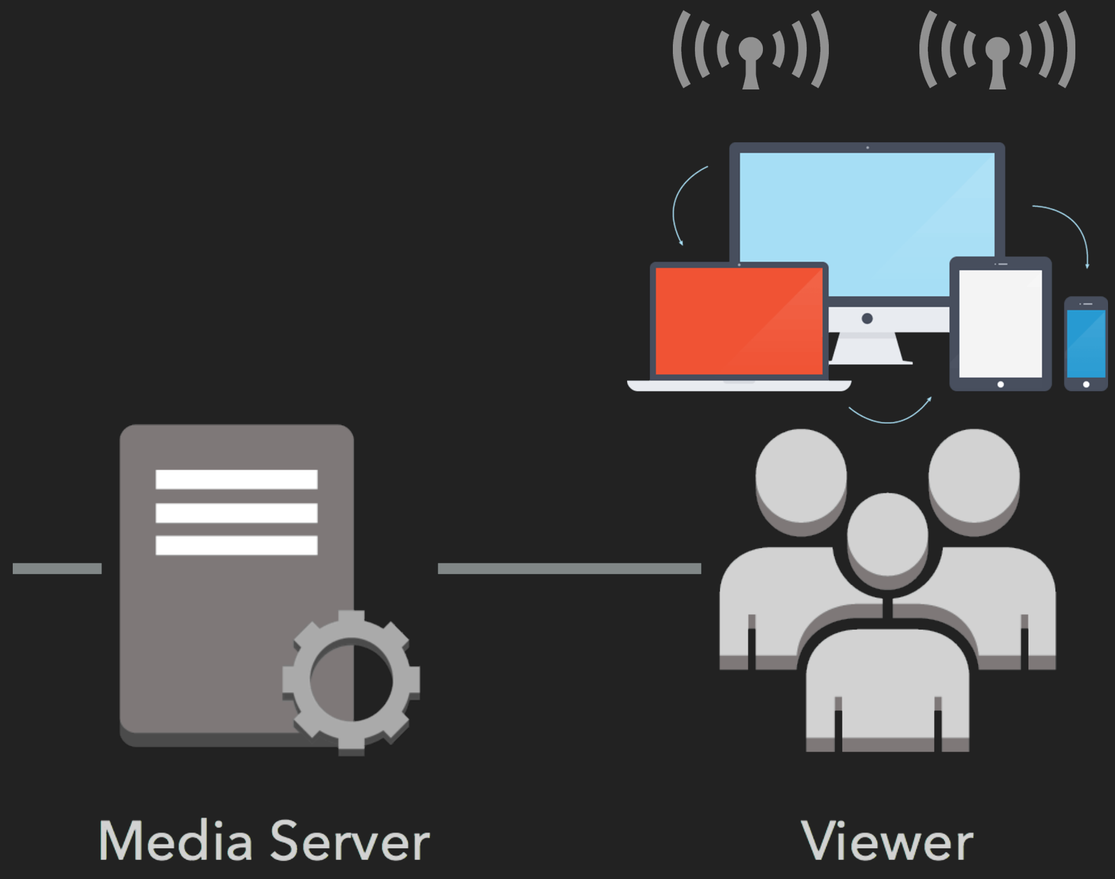
Live Streaming: Viewer Side

Mobile first, Multi Device.

Live Streaming: Viewer Side

Adaptive Bitrate Streaming

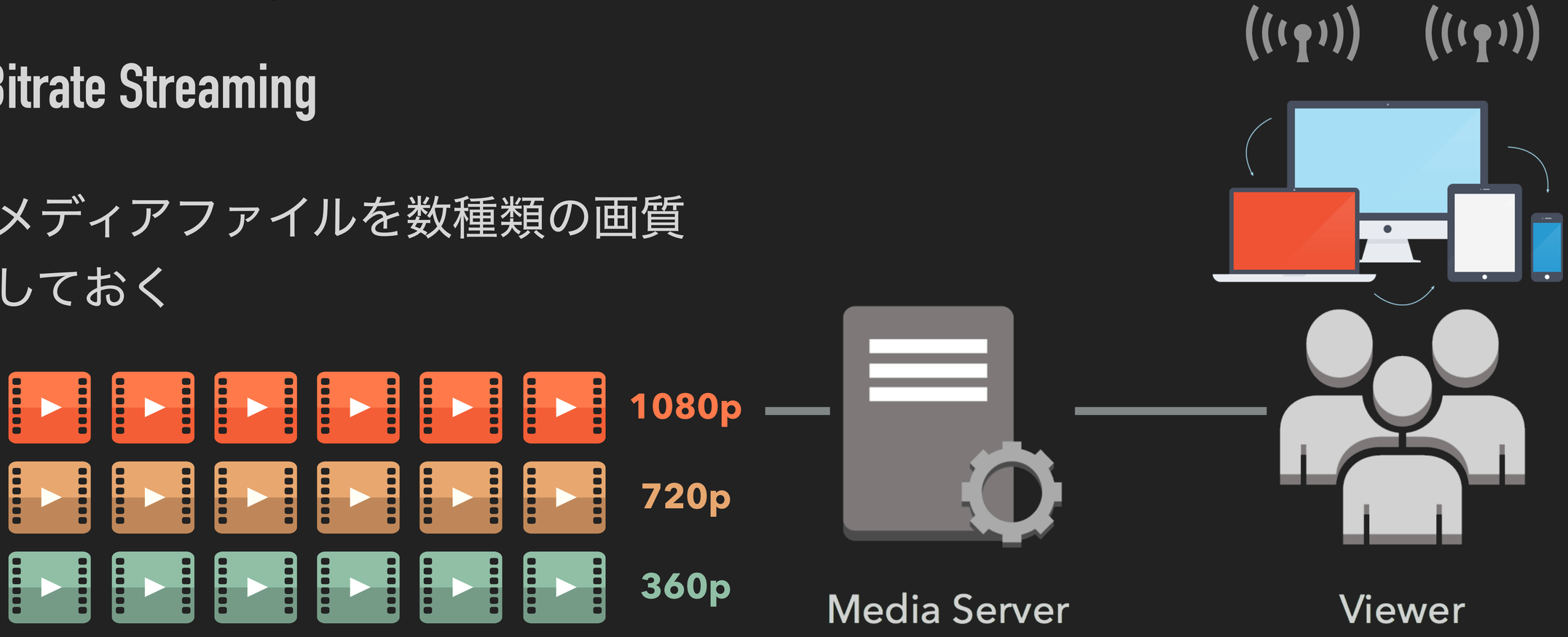
- ユーザーの回線状況によって画質を自動調整する技術
  - 帯域幅が繊細な Mobile では必須



Live Streaming: Viewer Side

Adaptive Bitrate Streaming

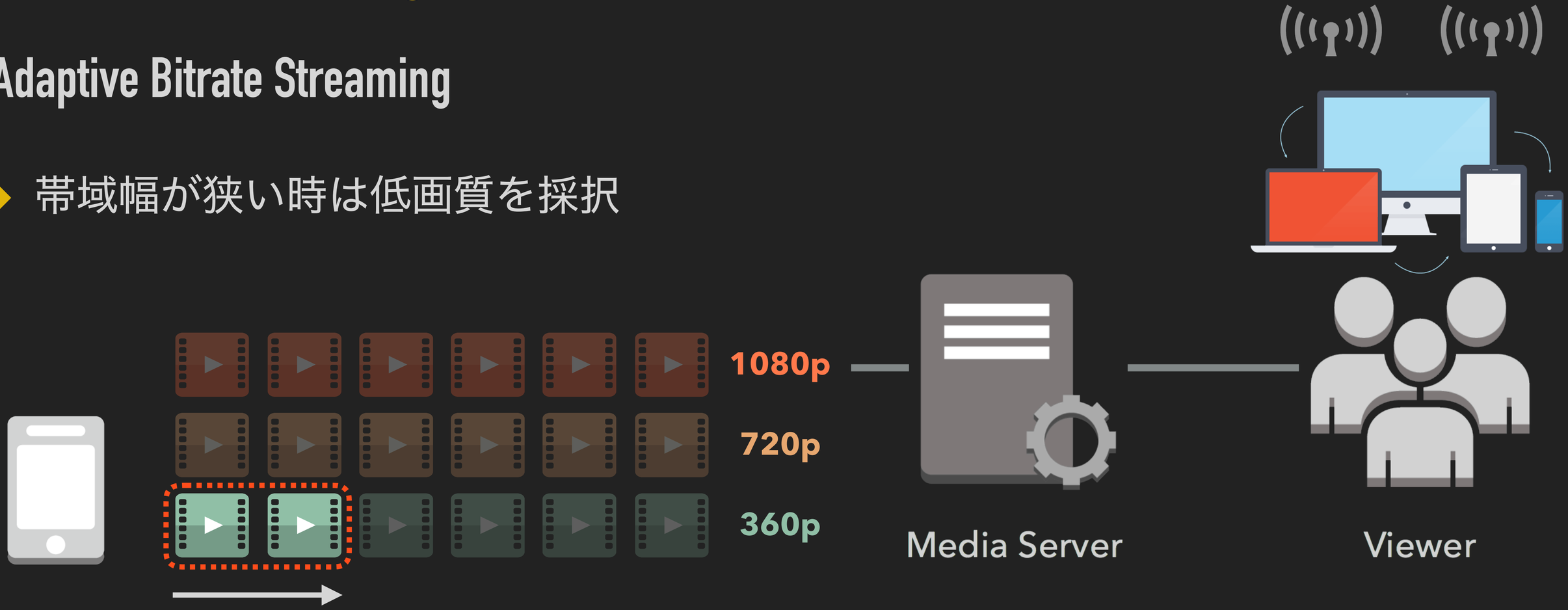
- 配信用メディアファイルを数種類の画質で用意しておく



Live Streaming: Viewer Side

Adaptive Bitrate Streaming

- 帯域幅が狭い時は低画質を採択



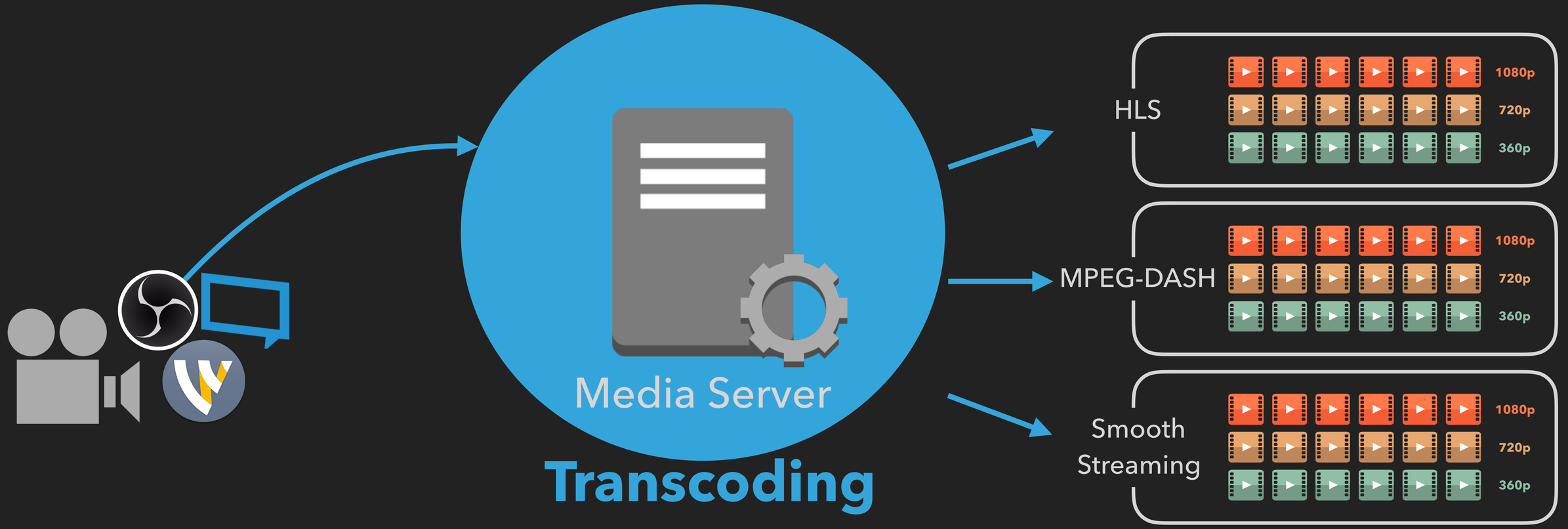
Live Streaming: Viewer Side

Adaptive Bitrate Streaming

- 帯域幅に応じて画質を切り替え

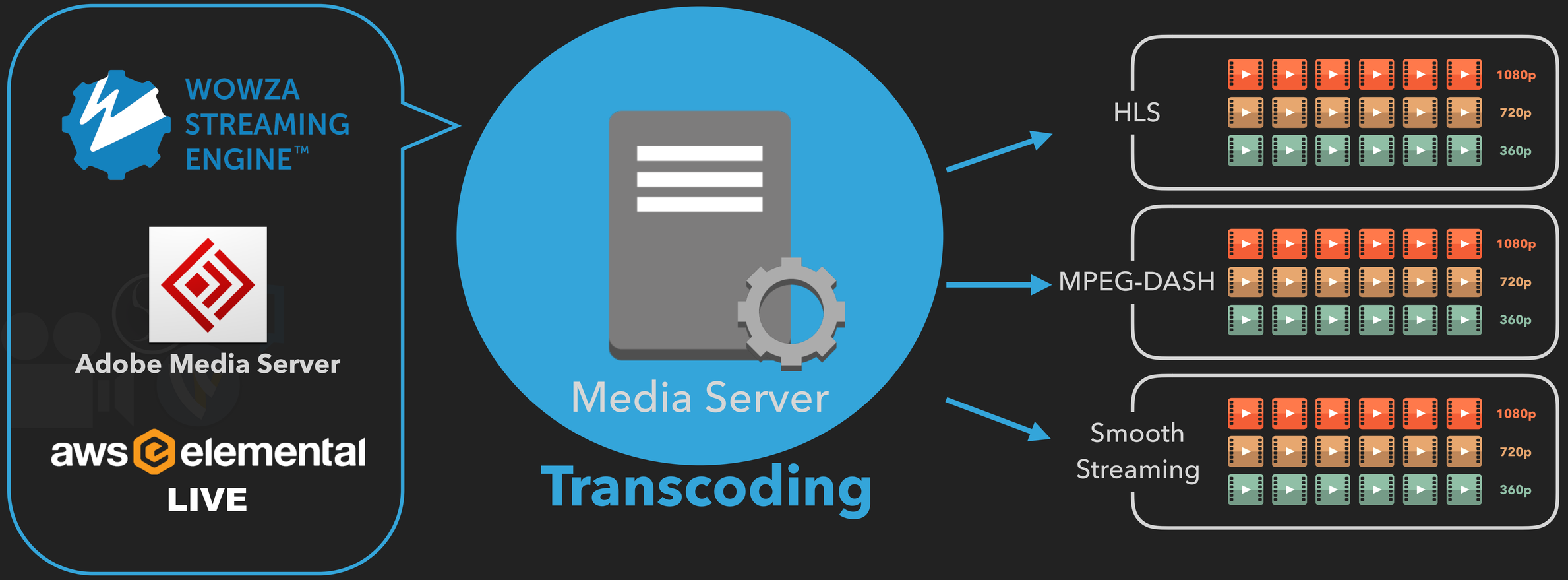
Live Streaming: Transcoding

受信したメディアデータを配信用データに変換



Live Streaming: Transcoding

Middleware for Live Transcoding



Live Streaming

一般的なライブ動画配信のアーキテクチャ

Broadcaster

Media Server
(Ingest / Transcode)

CDN

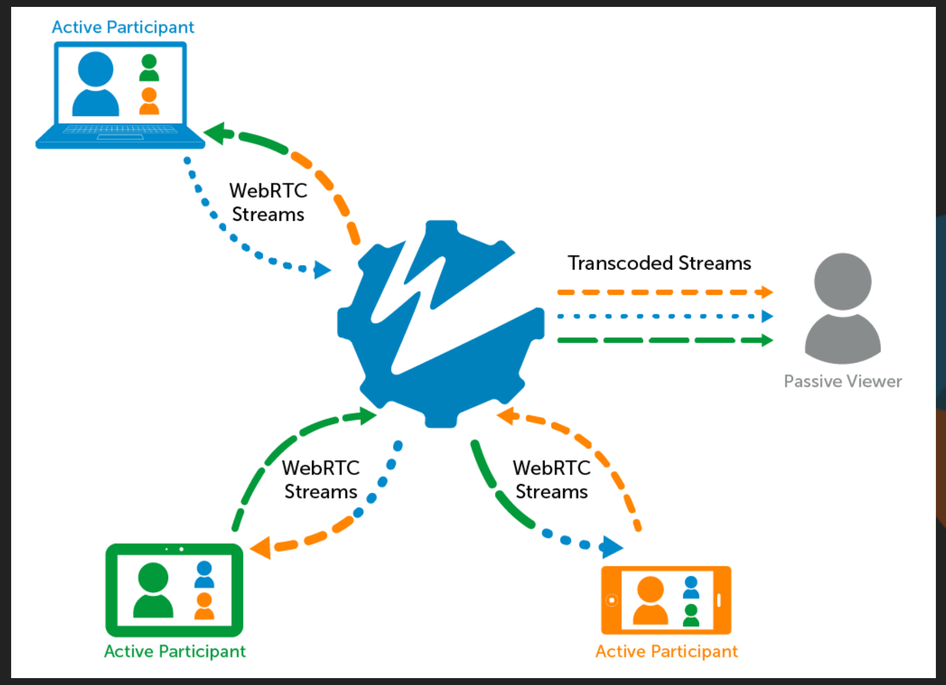
Viewer

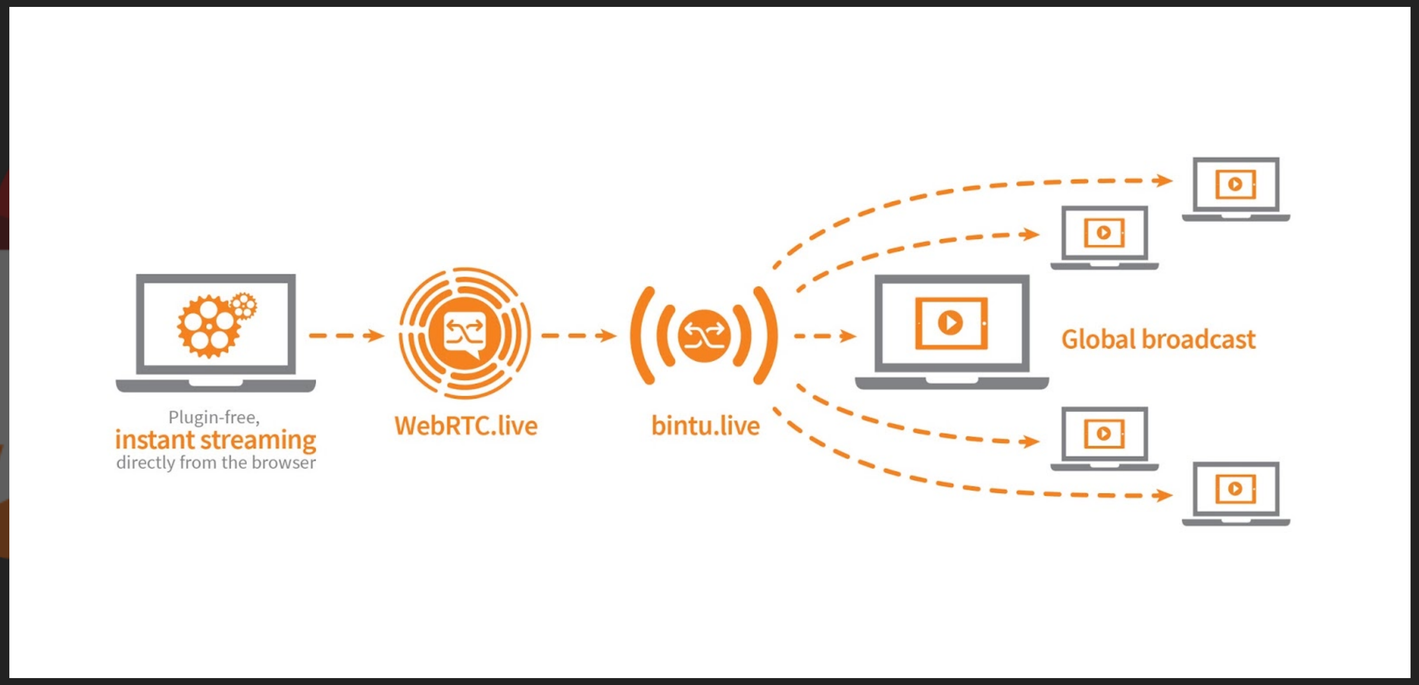
Live Streaming: New Generation

最近では WebRTC を用いた動画配信も

Live Streaming: New Generation

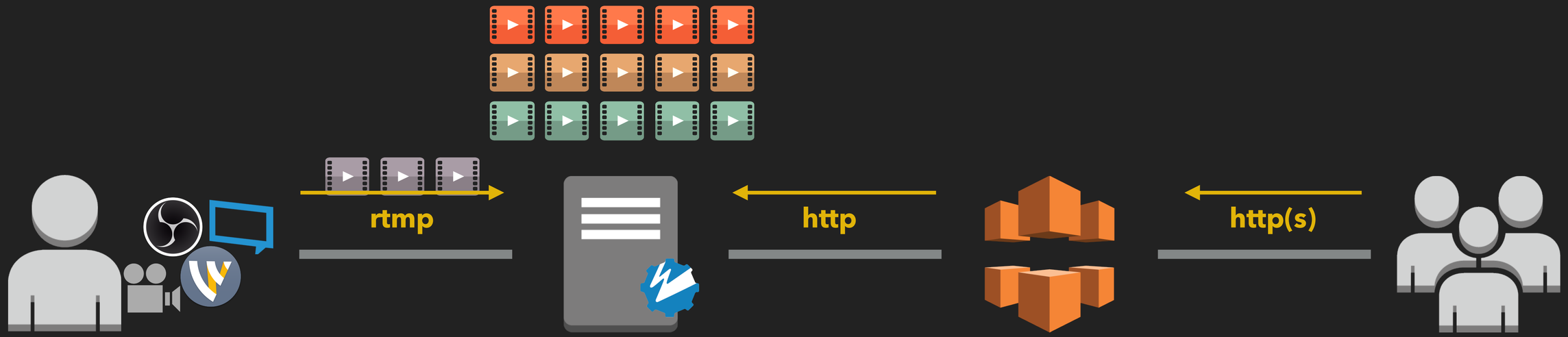
最近では WebRTC を用いた動画配信も





Time-Shifting

視聴者が **見たい時** に動画を視聴することができるようにすること



Broadcaster

Media Server
(Ingest / Transcode)

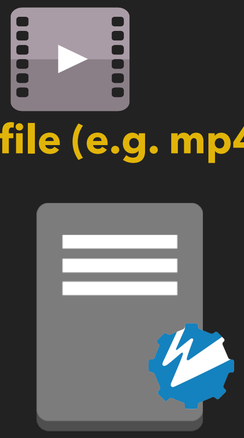
CDN

Viewer

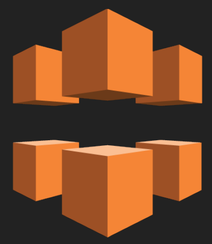
Time-Shifting

Live Streaming >> Stop

Broadcaster



Media Server
(Ingest / Transcode)

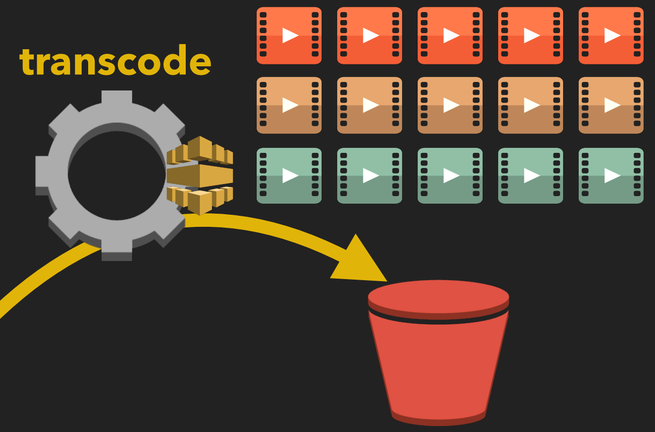


CDN

Viewer

Time-Shifting

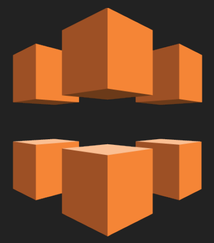
Live Streaming >> Stop >> Archiving



media file (e.g. mp4)

Broadcaster

Media Server
(Ingest / Transcode)

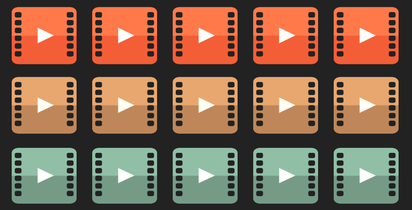


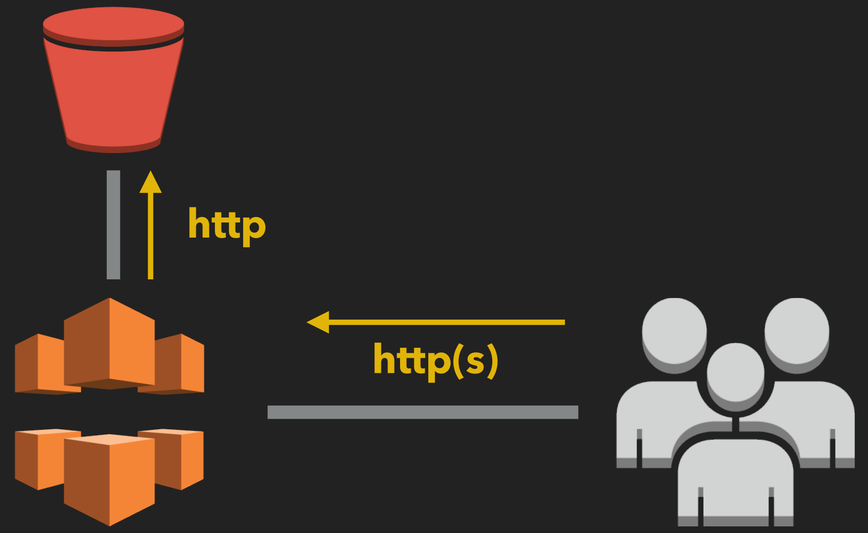
CDN

Viewer

Time-Shifting

Live Streaming >> Stop >> Archived = VOD





Broadcaster

Media Server
(Ingest / Transcode)

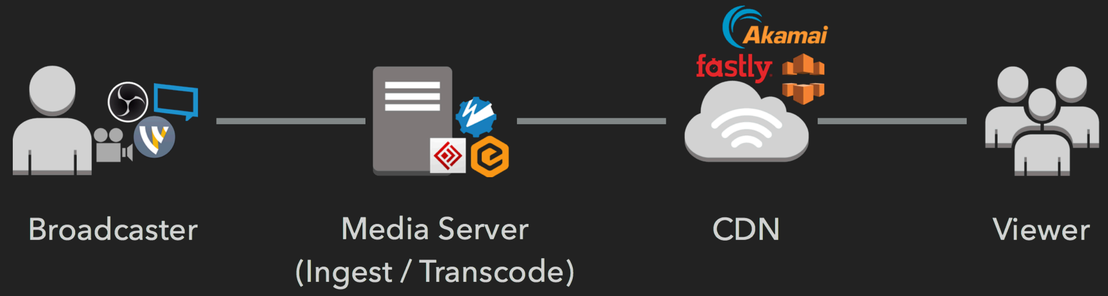
CDN

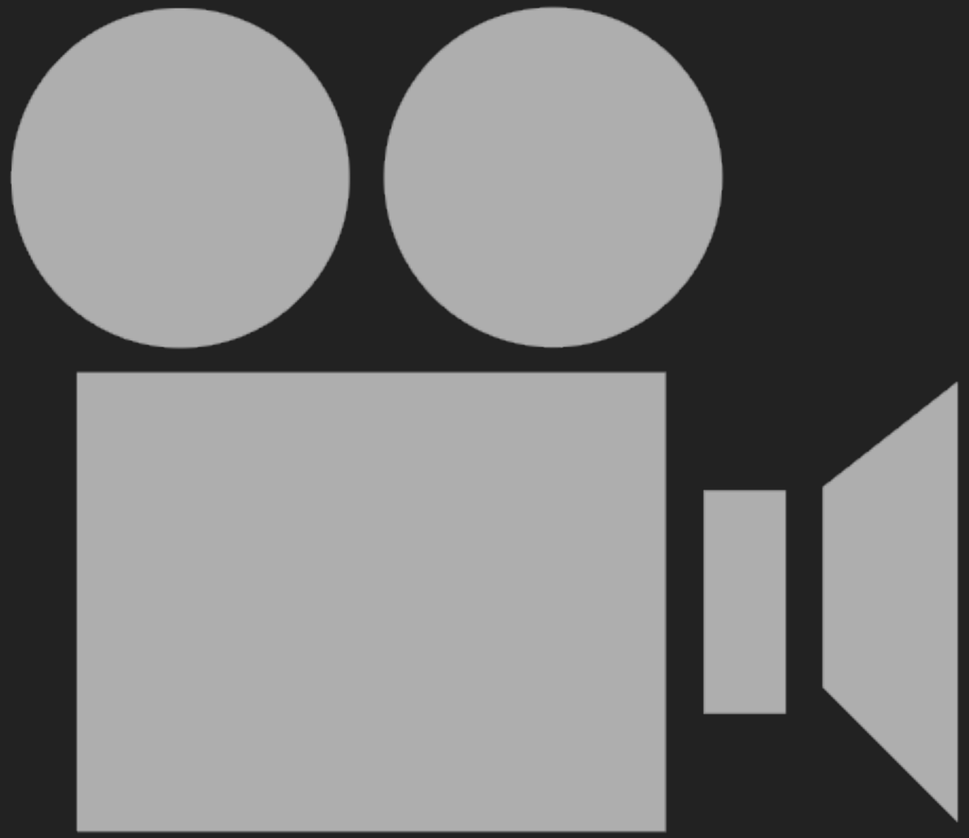
Viewer

動画配信の基本原則

「お金がかかる」

- Processing
  - トランスコーディングに要するパワフルな Processing Resource
- Memory
  - トランスコーディング時の Buffer
  - 配信 Caching
- Network
  - テキストデータと比べてサイズの大きいメディアデータ
  - 不特定多数の視聴者
- Storage
  - テキストデータと比べてサイズの大きいメディアデータの永続化





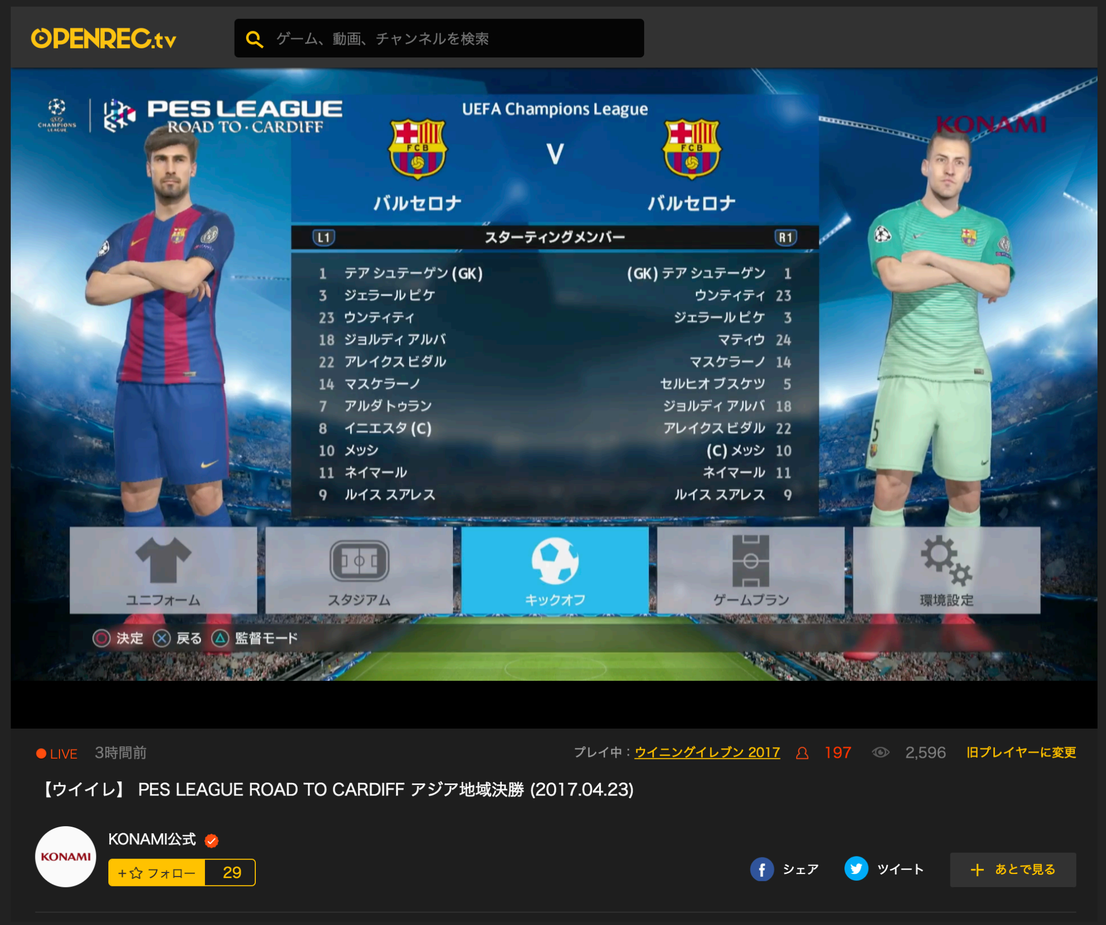
Case of

OPENREC.tv

Case: OPENREC.tv

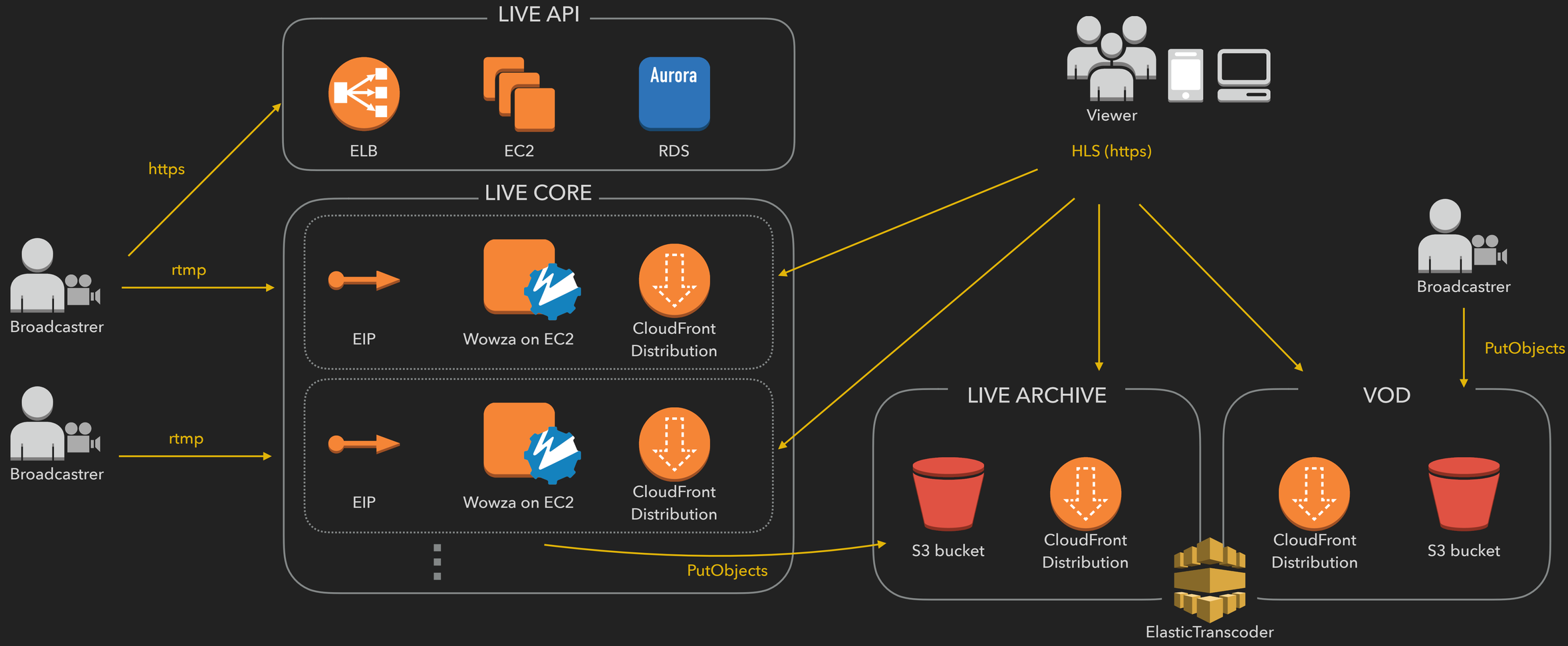
OPENREC.tv のライブ配信の要件

- **高画質・低遅延**
- **ユーザードリブンの配信**
  - 同時配信数も配信時間も配信者次第
  - 集客力も配信者次第
  - ∴ 負荷が読みづらい
- **同時視聴者数 無制限**
  - いわゆる “枠” は無い
  - 全ユーザー等しくライブ視聴できること
- **全ライブ配信 アーカイブ**
  - いわゆる “Time Shifting”, VOD化
  - アーカイブ動画を公開するかどうかは配信者の自由



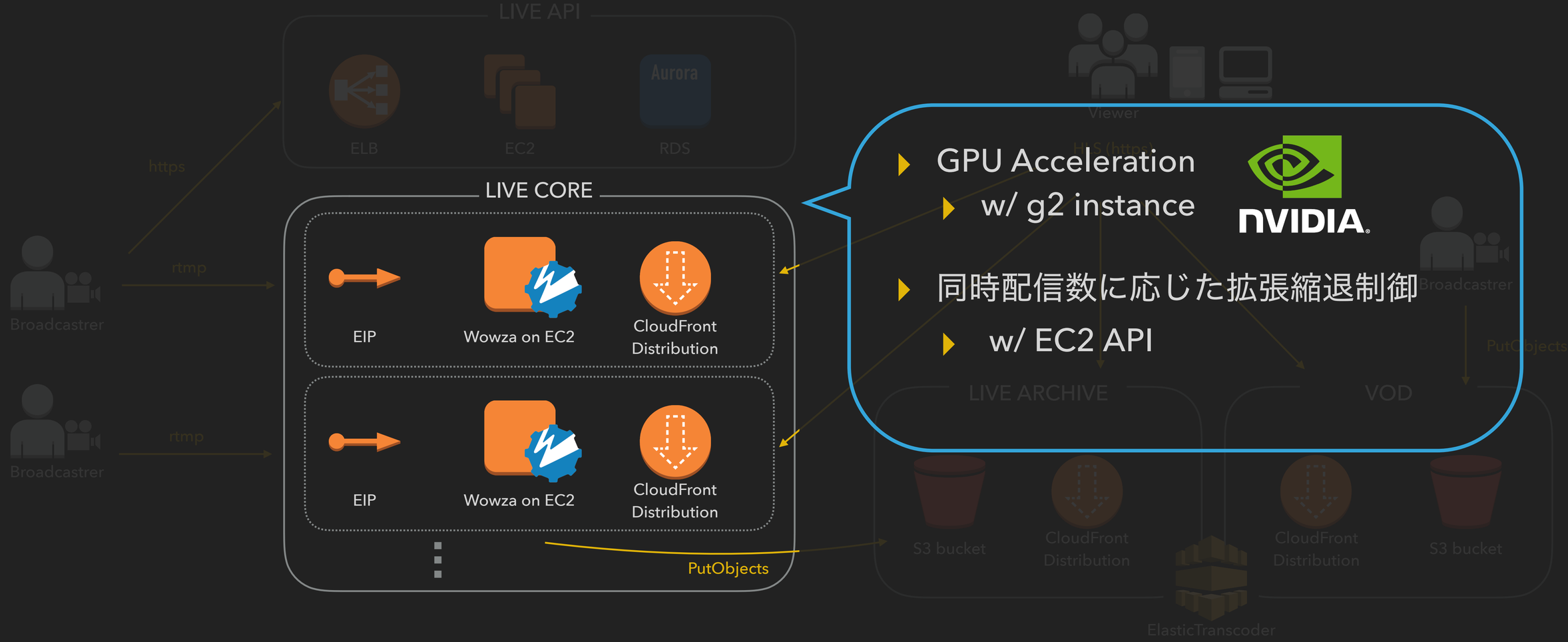
Case: OPENREC.tv

Architecture: Wowza on EC2 (c4, g2), CloudFront, S3, ElasticTranscoder



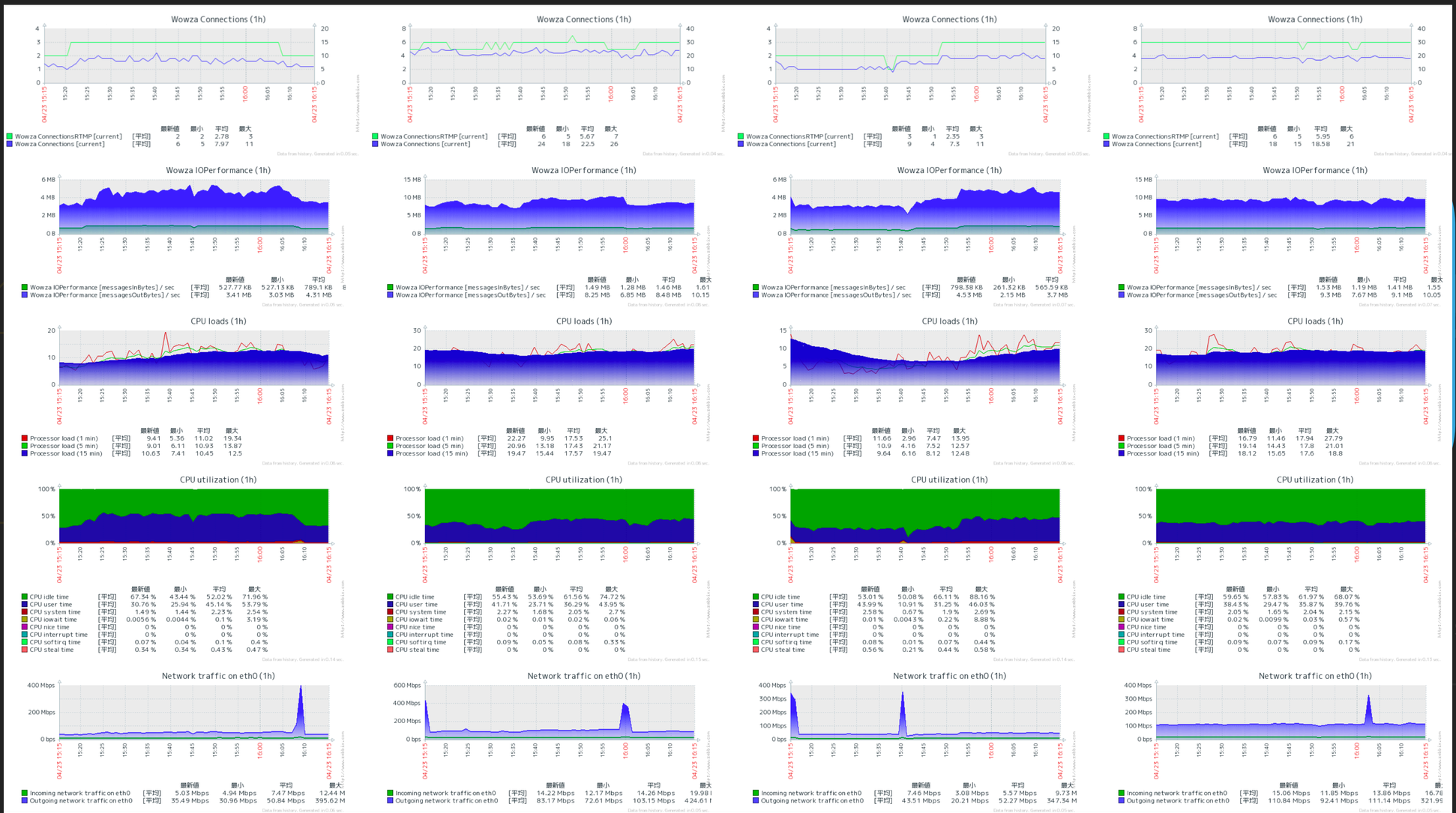
Case: OPENREC.tv

Processing Layer



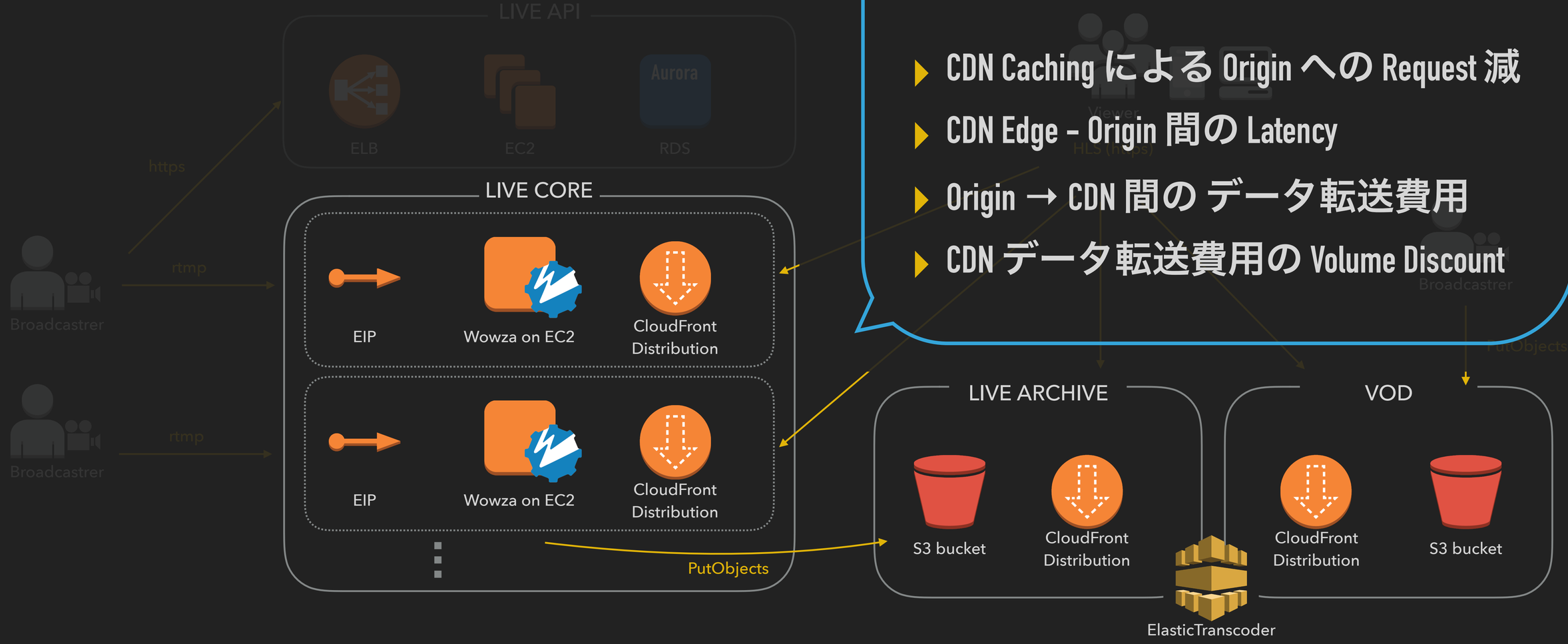
Case: OPENREC.tv

CPU 使用率が 50 〜 70% 程度に抑えられるように配信を分散



Case: OPENREC.tv

Network Layer



All transfer in AWS

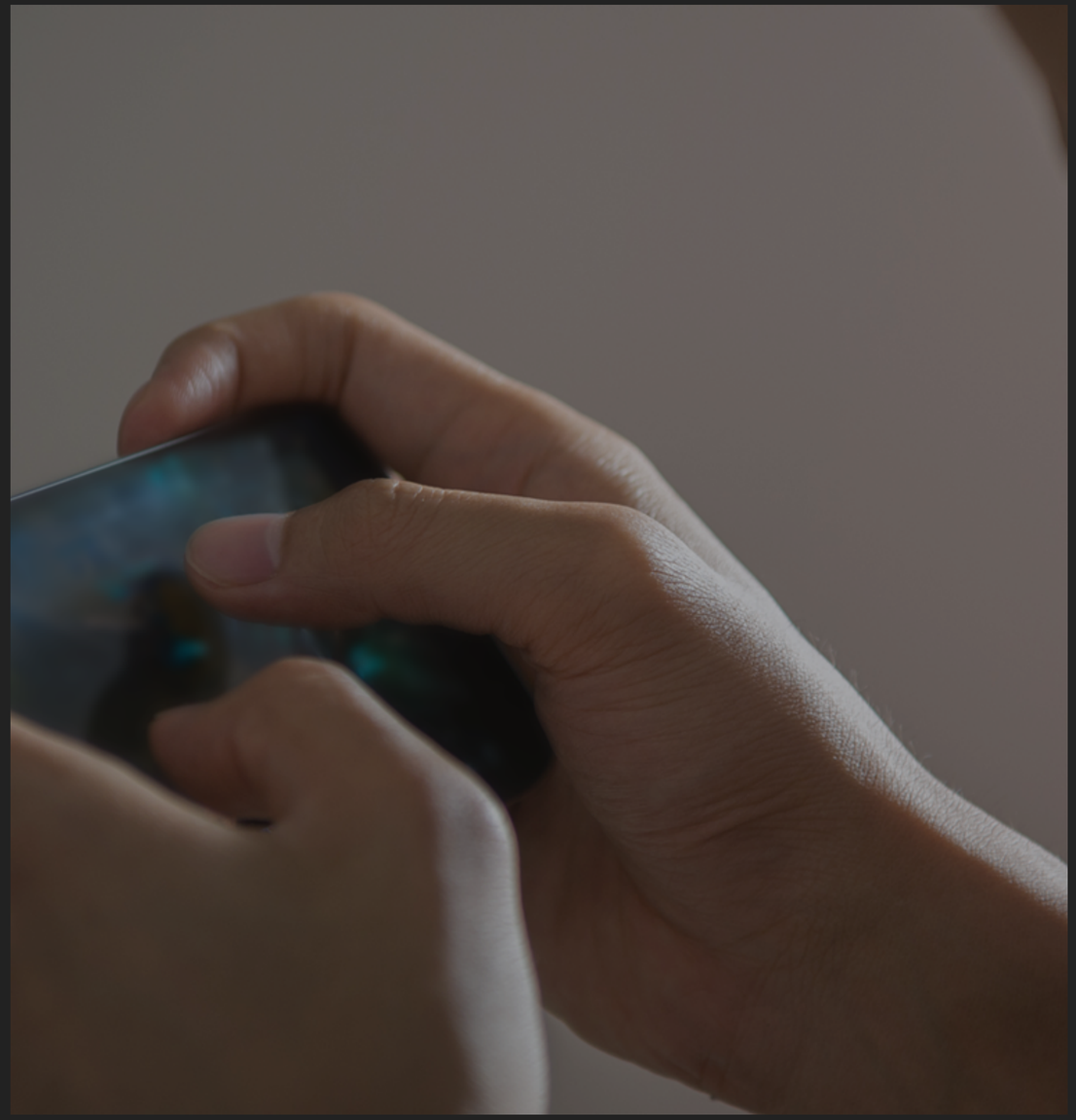
- CDN Caching による Origin への Request 減
- CDN Edge - Origin 間の Latency
- Origin → CDN 間の データ転送費用
- CDN データ転送費用の Volume Discount

Replaces

2014 - 2015

プレイ動画録画 SDK 時代

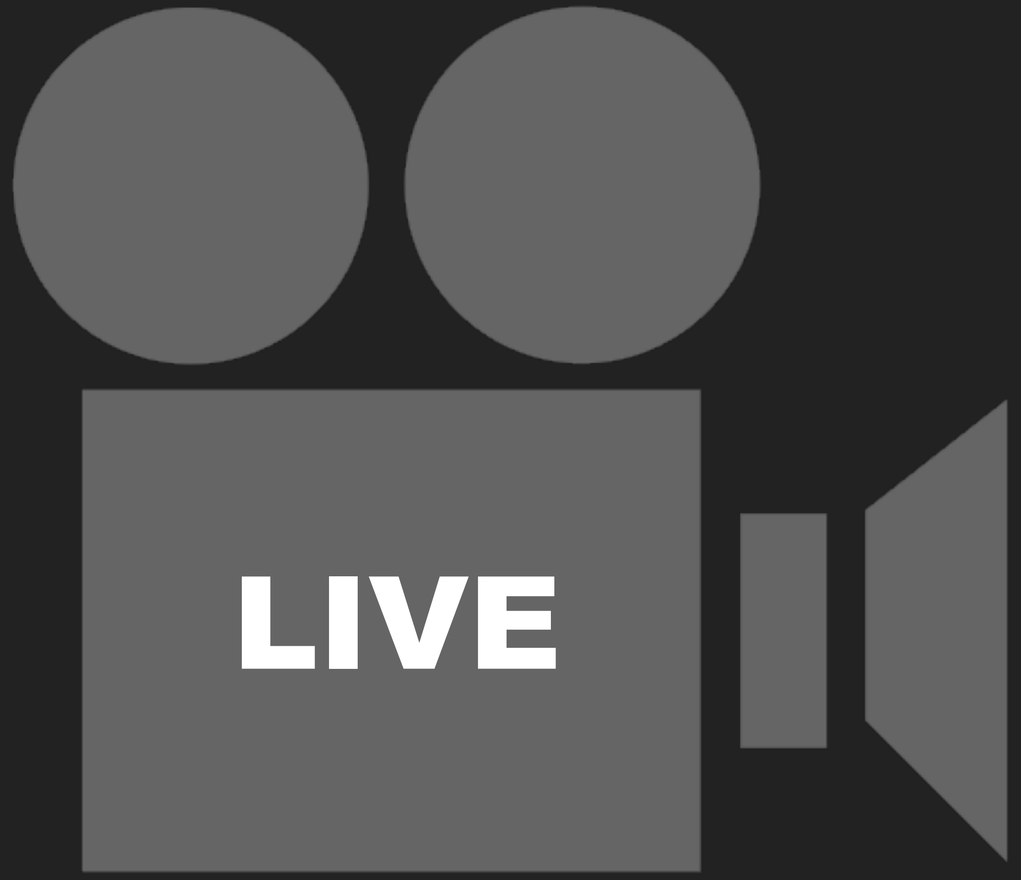
- スマホ特化のプレイ動画共有サービス
- iOS / Android から 録画 / 編集 / 投稿
- brightcove ZENCODER 採用
  - 当時の肝の SDK 開発に注力



2015 - 2016

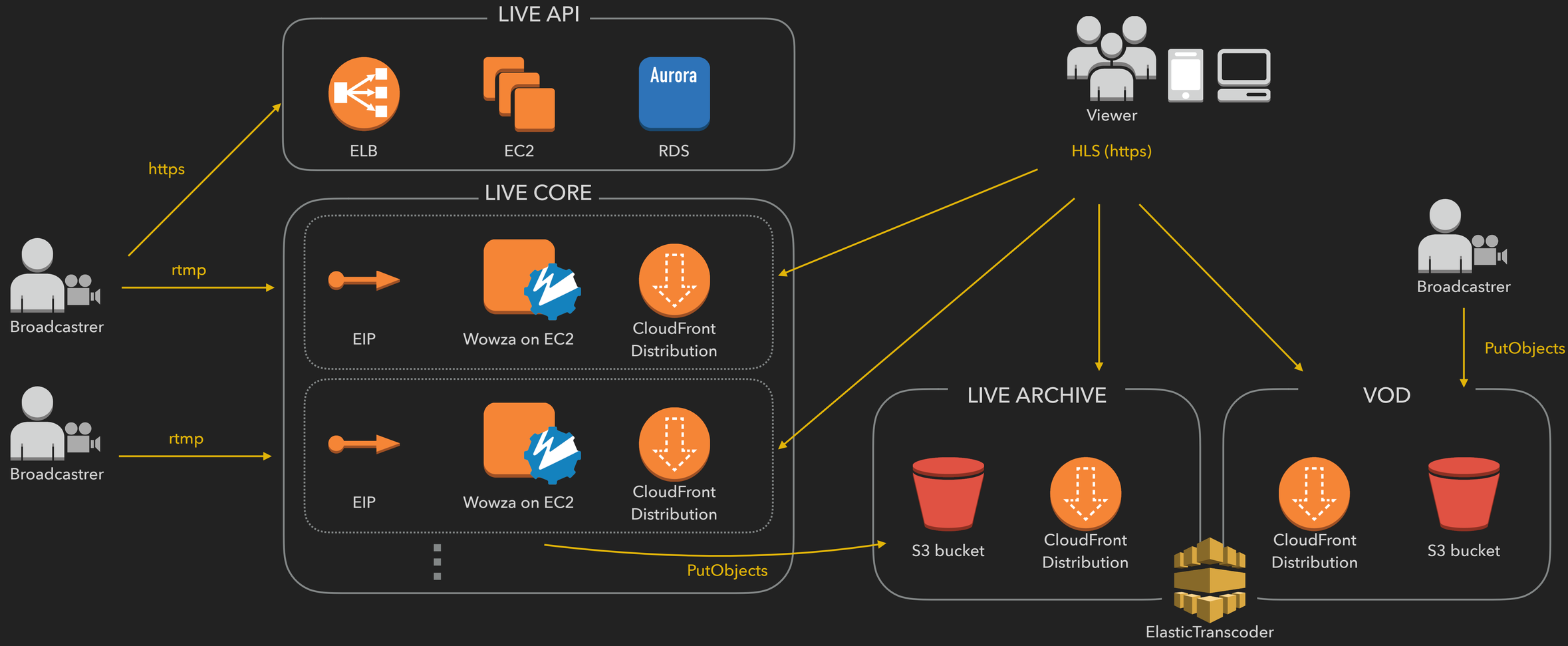
動画投稿 → LIVE 配信へ

- 小さく始める
  - リードタイム削減のため Zencoder Live を活用
- 大きく育てる
  - 2016 年に入ると LIVE に注力
  - 高画質・低遅延・低費用に向き合う



1st Replace in 2016 Mid

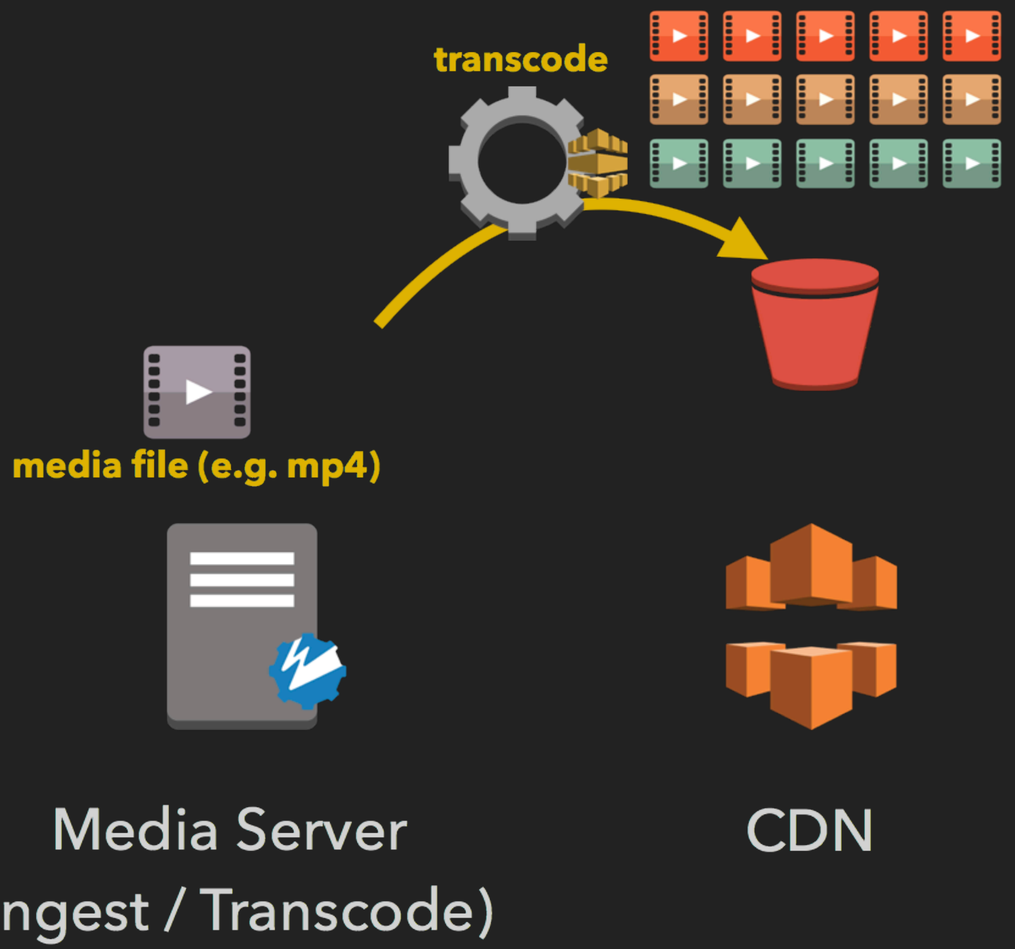
Replace Transcoder



2016 Mid - Later

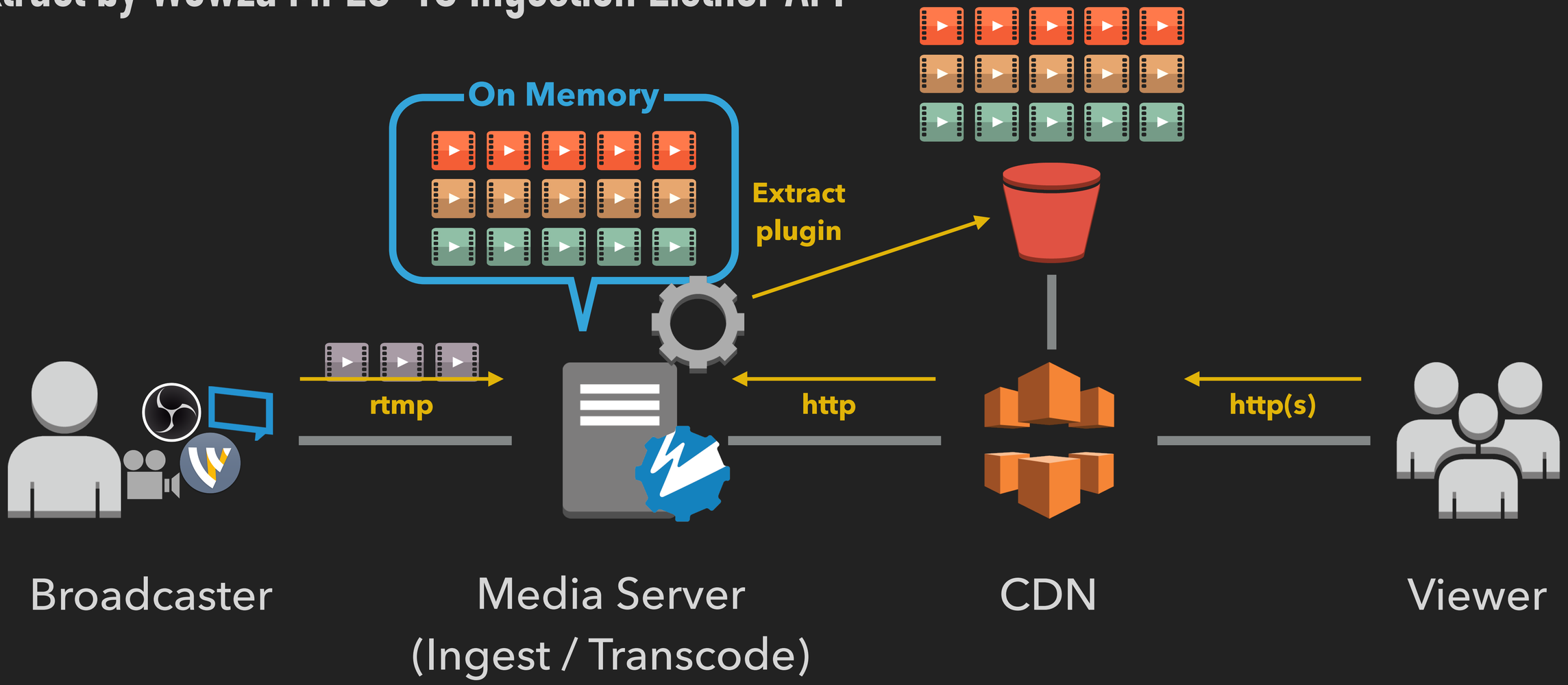
アーカイブ機能の課題感

- VOD化に要するリードタイム
  - ライブ終了後にパッケージングされた mp4 を ETS で 再 Transcode して S3 に永続化
  - 配信が24時間にもなると目も当てられない
- Elastic Transcoder の費用増
  - アーカイブ対象は 全ライブ配信
    - 公開/非公開は 配信者の権利
  - 料金は 動画数 × 配信時間 の掛け合わせ



2nd Replace in 2016 Later

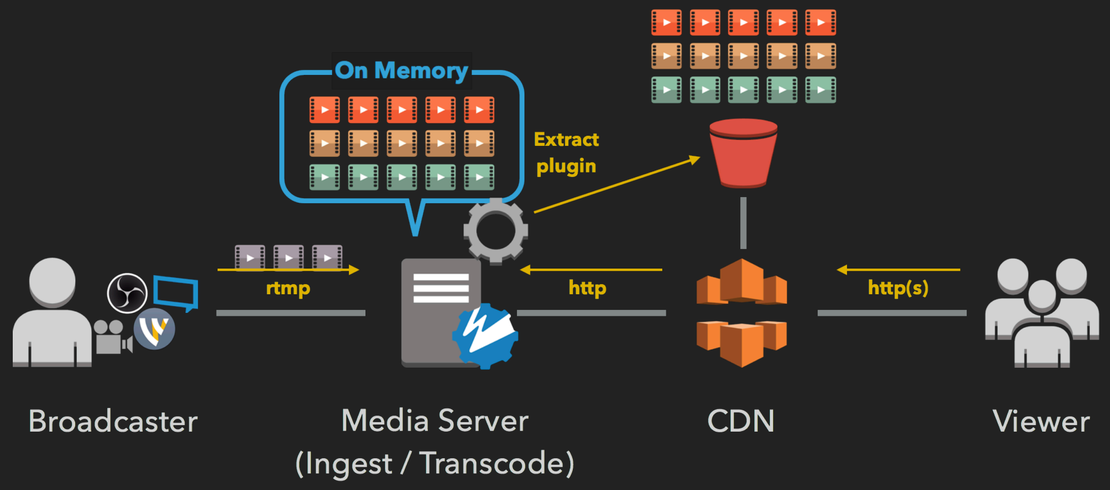
Extract by Wowza MPEG-TS Ingestion Listner API



2nd Replace in 2016 Later

Extract by Wowza MPEG-TS Ingestion Listner API

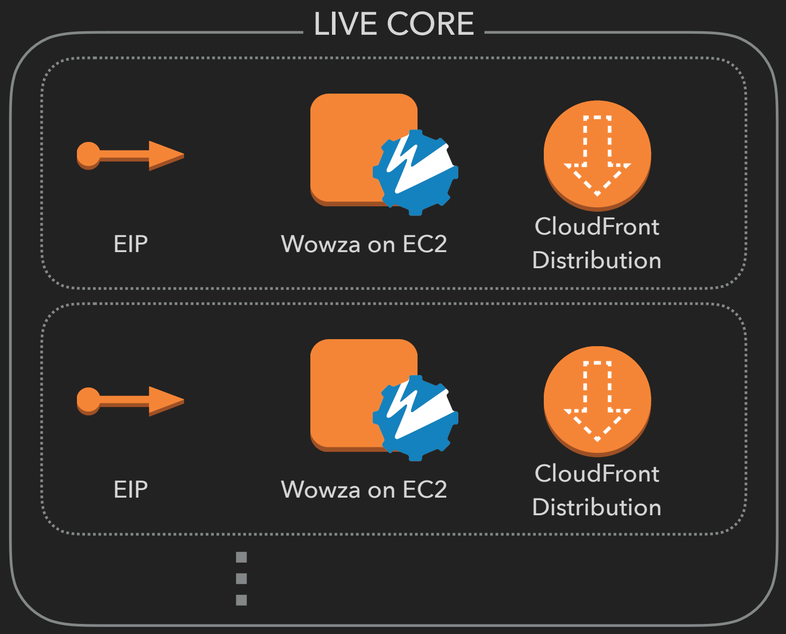
- Wowza Streaming Engine v4.5.0 or later
- ライブ配信中メモリ上にある HLS を抽出
- 即時アーカイブ化を実現
- アーカイブ化のための再 Transcoding 費用を大幅削減
  - Elastic Transcoder は投稿動画や、不安定なライブ動画の変換に引き続き利用



2017 Early

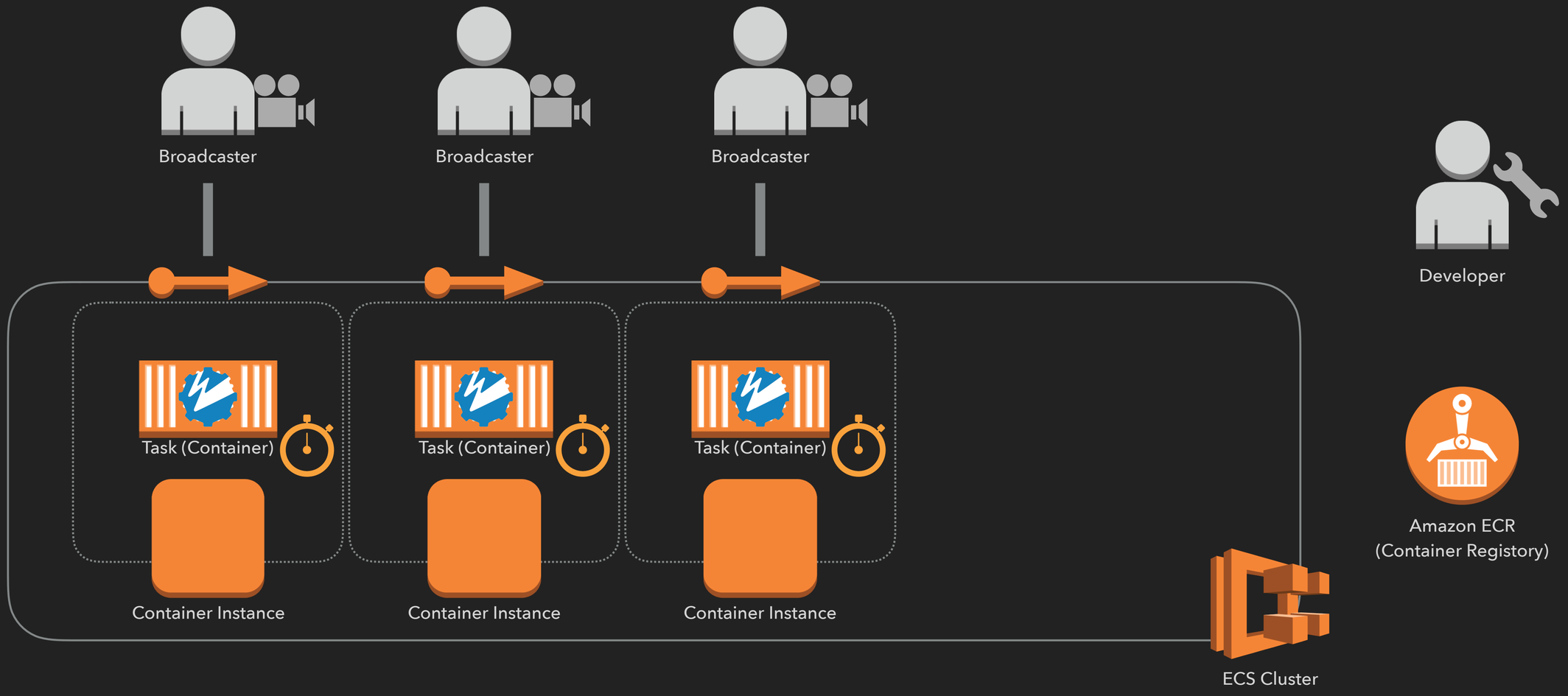
規模の拡大

- 同時ライブ配信数 増加
  - 2016 Mid: 〜 数十
  - 2017 Mid: 〜 数百
  - これに伴い サーバ数も増加
- ローリング デプロイ / メンテナンスの煩わしさ
  - ライブ配信を受け付けた Wowza サーバをライブ終了まで落とせない
    - 配信者からの接続は持続接続 RTMP
  - ユーザー主導のライブ配信は終了時間が読めない



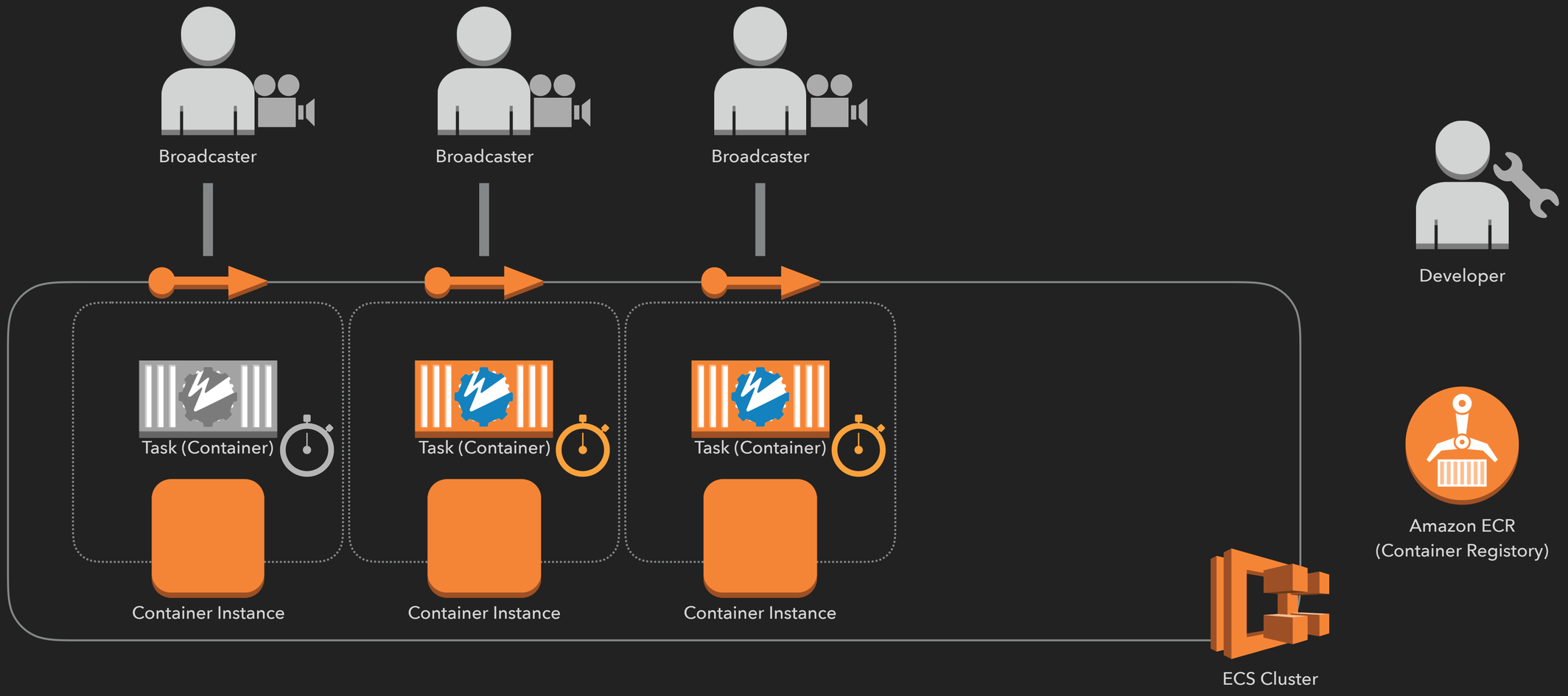
Next 3rd Replace

Wowza サーバを揮発化



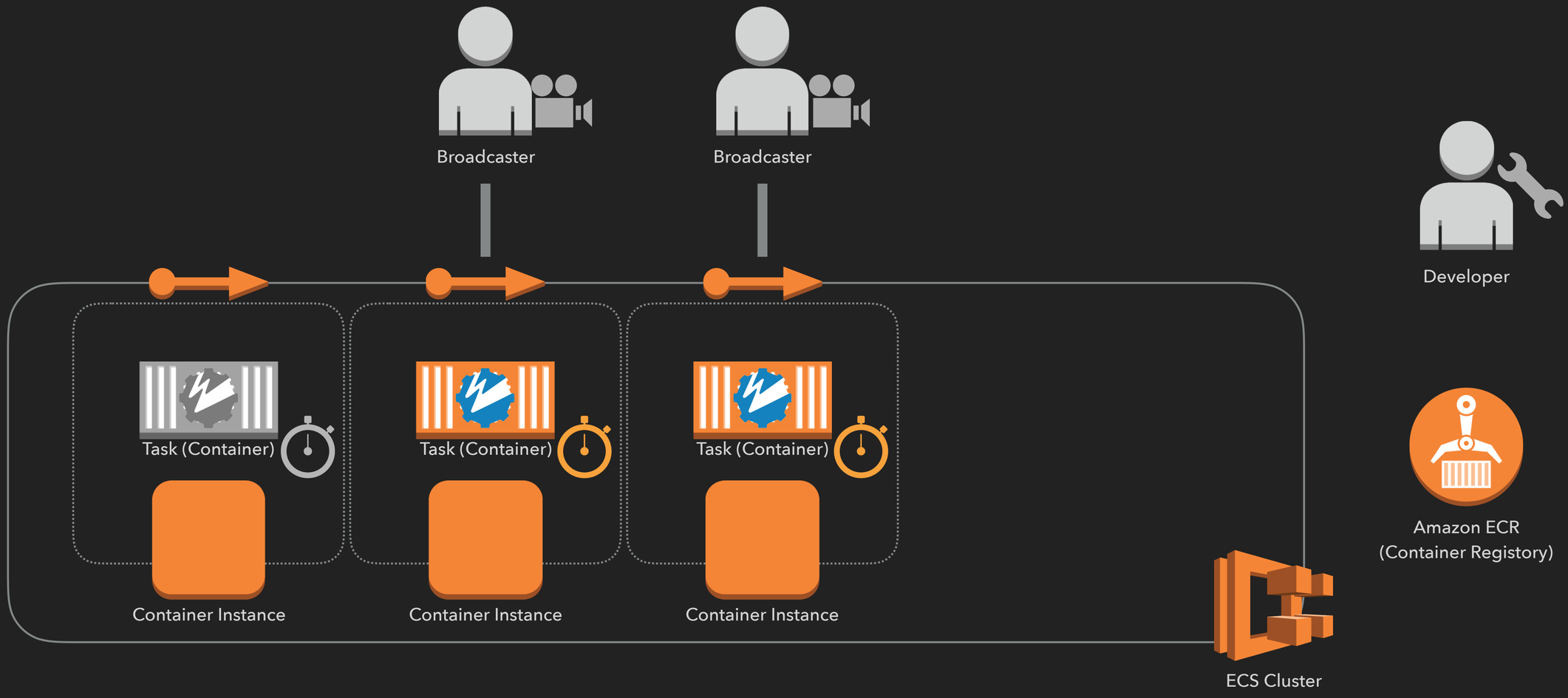
Next 3rd Replace

Expire >> Drain >> Stop Container



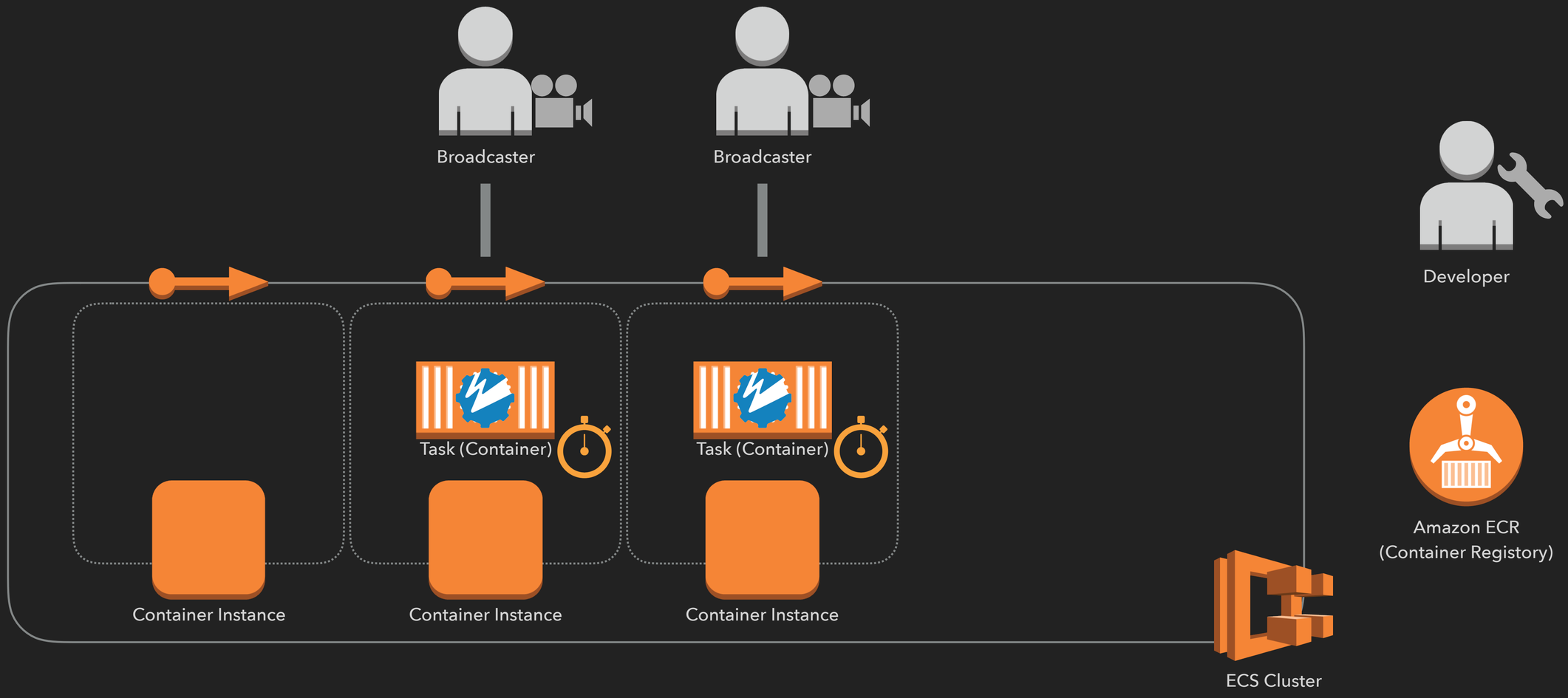
Next 3rd Replace

Expire >> Drain >> Stop Container



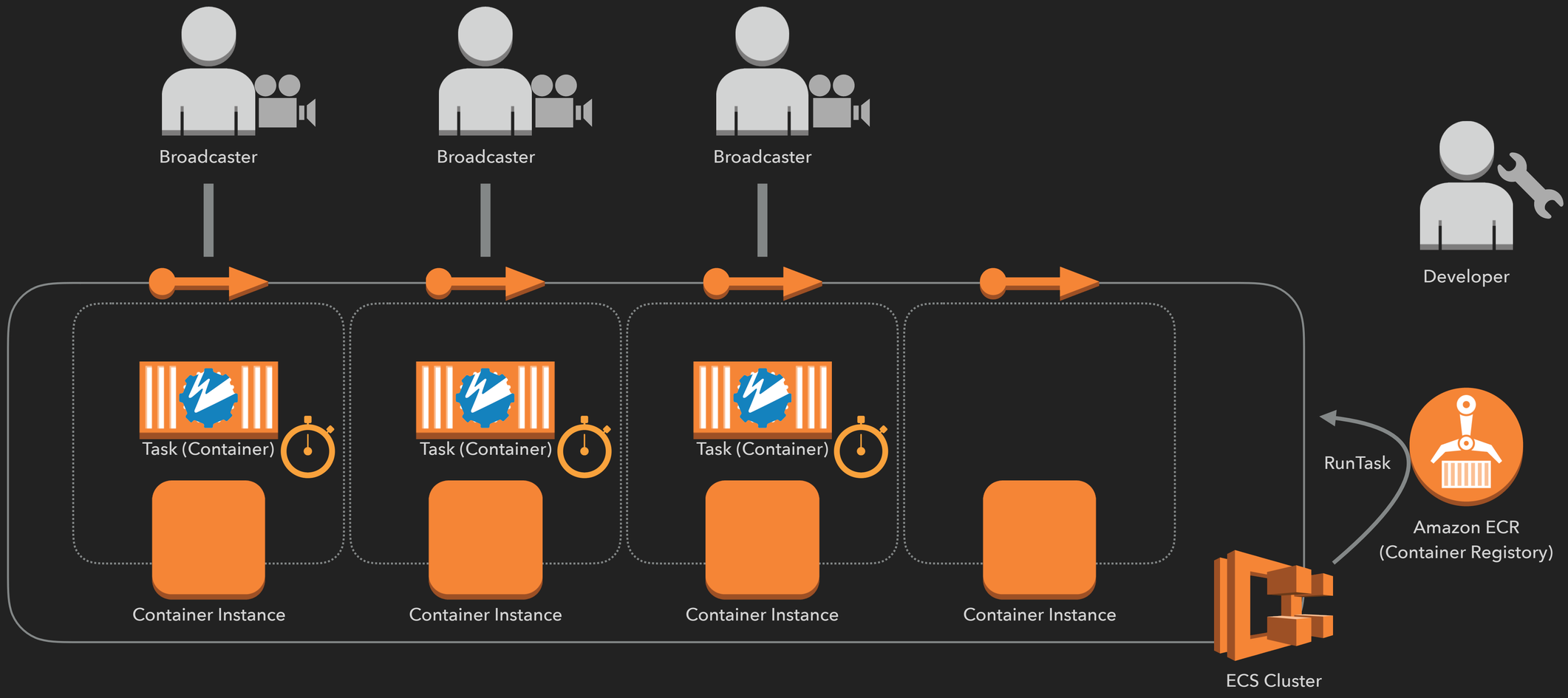
Next 3rd Replace

Expire >> Drain >> Stop Container



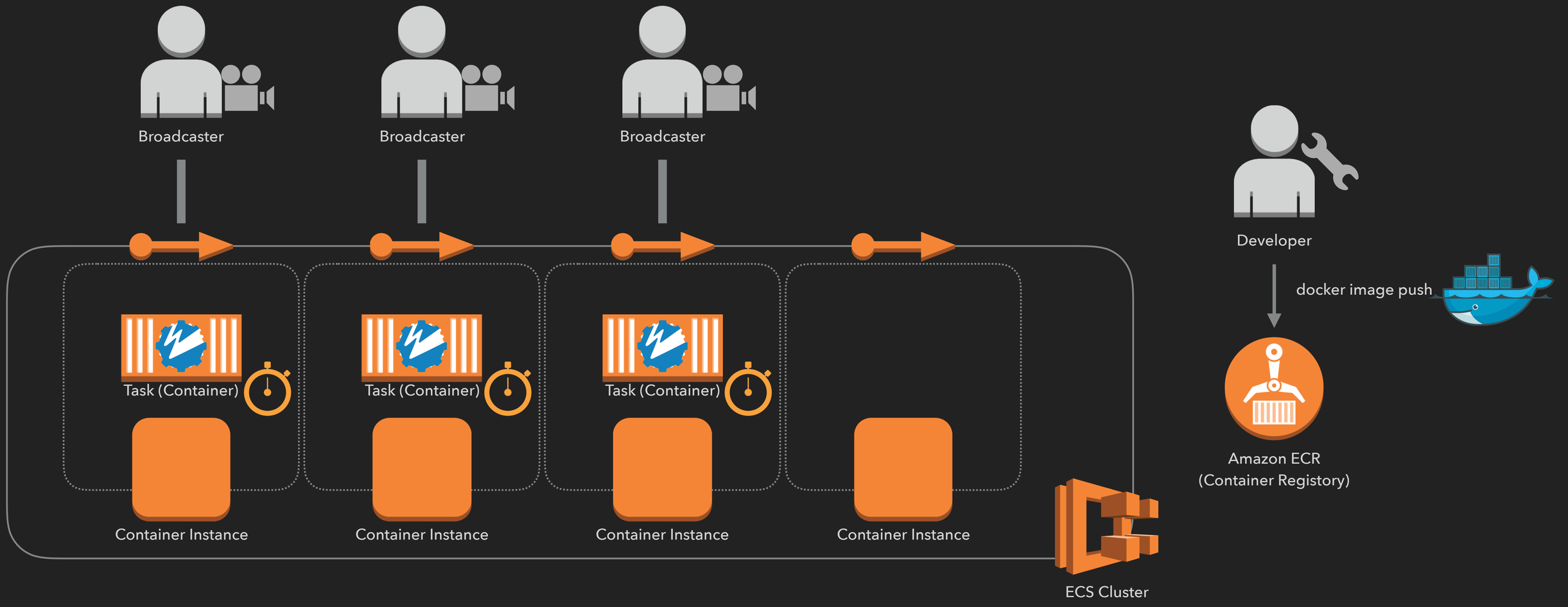
Next 3rd Replace

配信数に応じて Container から Instance から EIP/DNS Record まで AutoScale



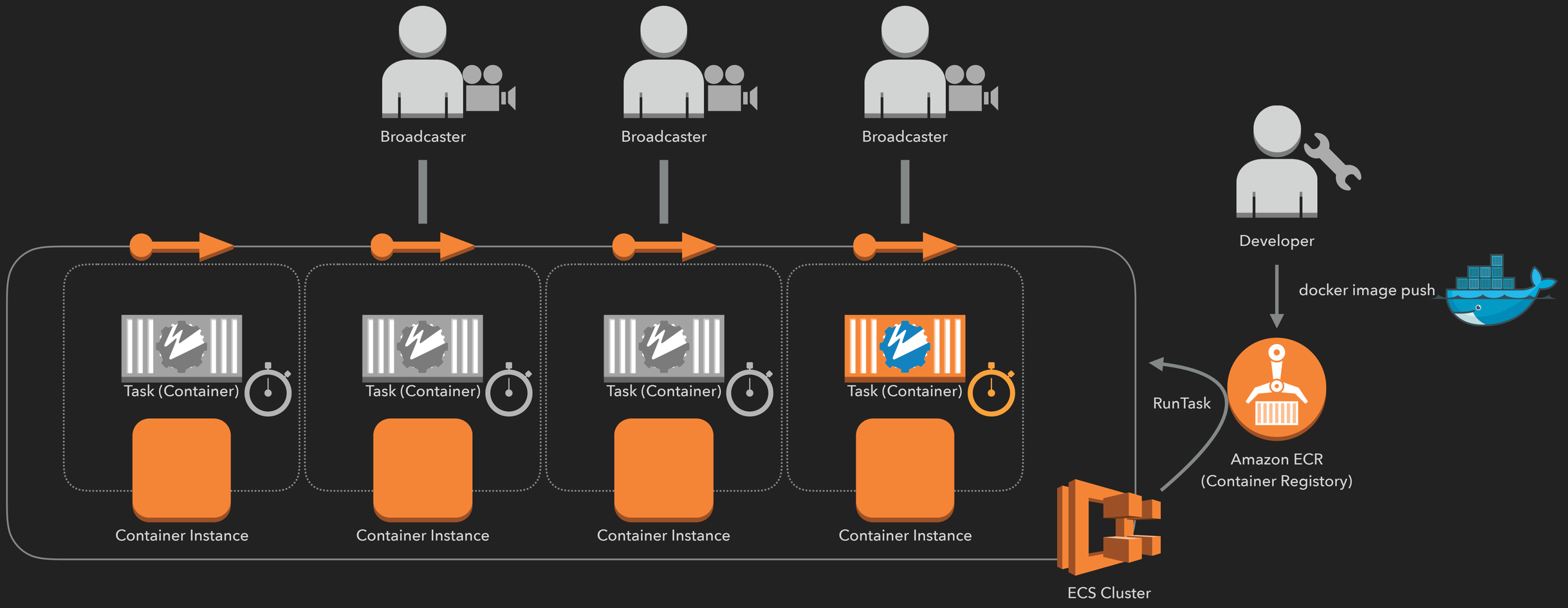
Next 3rd Replace

リリース作業は`docker image push`のみ



Next 3rd Replace

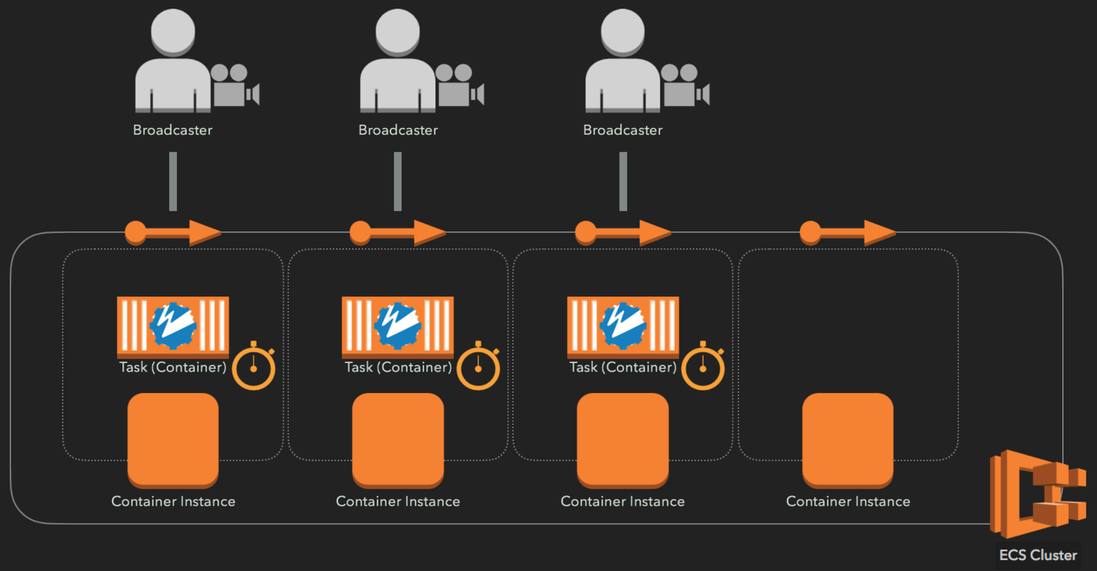
あとは新イメージが勝手に浸透するのを待つだけ



Next 3rd Replace

Container Scheduler

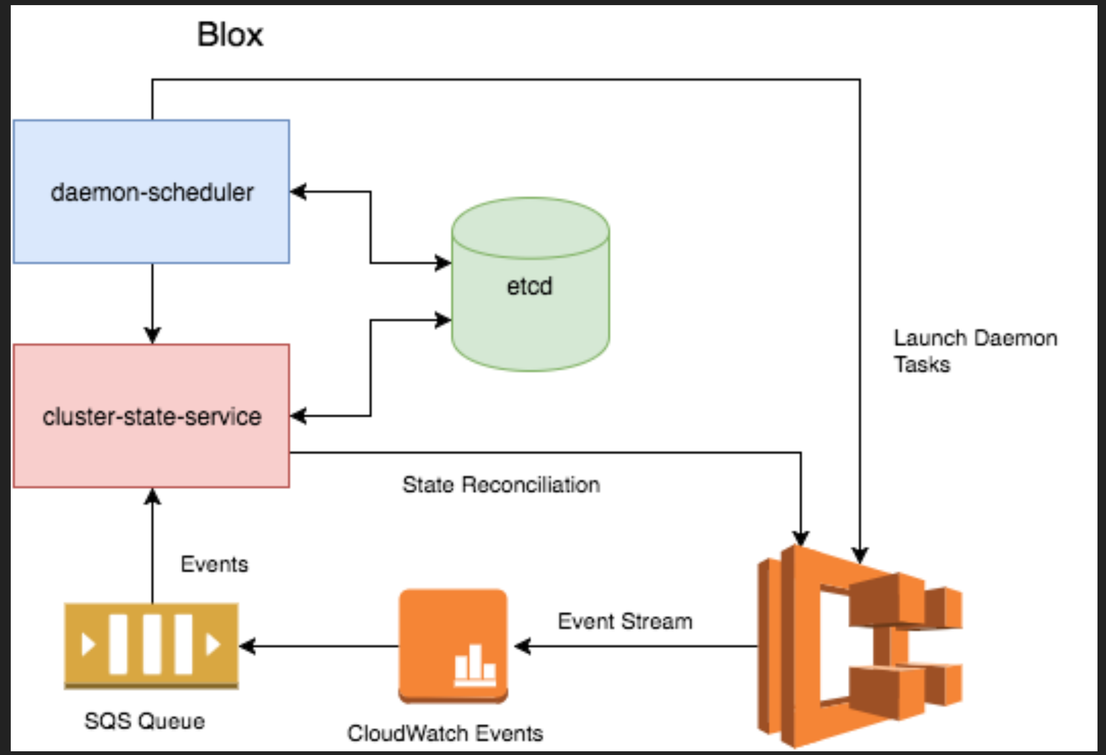
- EC2/ECS Auto Scaling は相性が悪い
  - RTMP = 常時接続
  - 負荷が低くても配信していれば縮退できない
- “配信状況” という独自指標に基づいて
  スケールさせるスケジューラーが必要



Container Scheduler

1. Blox

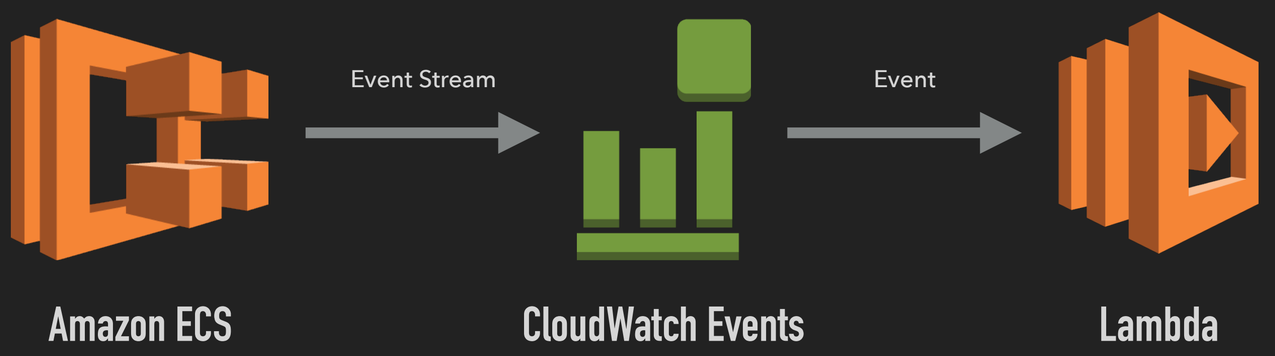
- Container Scheduler for Amazon ECS
  - re:Invent 2016 で公開された golang 製 OSS
- ECS Cluster 用のカスタムスケジューラを実装可能
  - ECS Cluster のイベント検知
  - ECS Cluster の状態追跡
  - カスタムスケジューラーの実行
  - REST API の提供



Container Scheduler

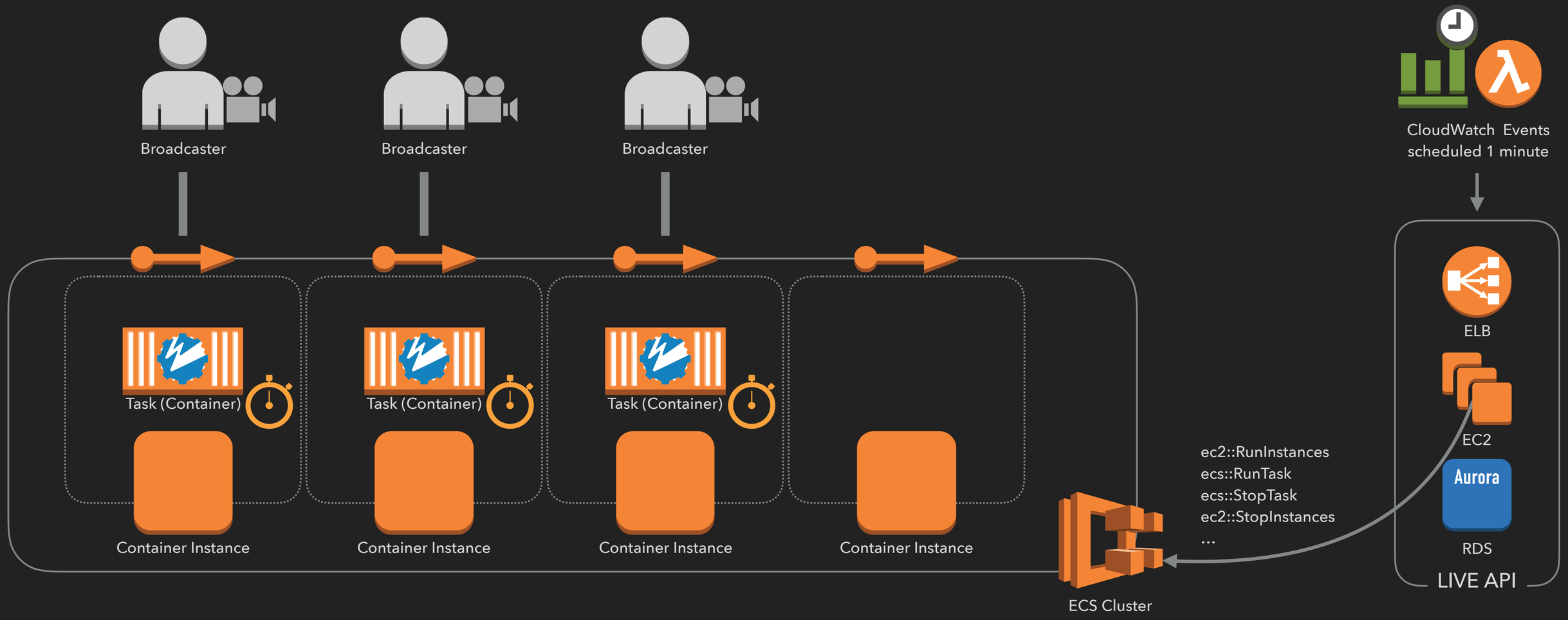
2. Amazon ECS Event Stream + Lambda

- Amazon ECS のタスクとコンテナインスタンスの状態更新を hook に Lambda function を実行



Next 3rd Replace

3. CloudWatch Events (scheduled/1min) + Lambda + API



Next 3rd Replace

3. CloudWatch Events (scheduled/1min) + Lambda + API

Coming soon . . .

Chat

Messaging

OPENREC.tv における Chat

Offline - Online 間の空間共有

- 会場の臨場感を Online で疑似体験
  - e.g. e-sports イベント **RAGE**

- Communication
  - 配信者 - 視聴者 間
  - 視聴者 - 視聴者 間

Case: Minecraft

シーラカンスqp： 平然と入ったな

シオン： 景色いいじゃん

ホームズ： 爆発物があるw

へぴいちごqp： 躊躇なく

(╬ﾟдﾟ)： テラスを照らすw

しぎちゃん： こんばんは！今来ました！

シオン： スイートルームかな？

yuukimegan： 躊躇なく

ponilyokorikei1215： さらっと

カゲさん： お風呂の水が漏れない不思議システム

(╬ﾟдﾟ)： ふたばさんは何にも作らないんですか？

シーラカンスqp： 温泉欲しいねー

Masaki 09391： 今北産業

海ホタル： 今来ました～こんばんは～

あずみqp： 28さんもこういうの作ろー

ぷよくっくqp： おー

LL TE： 食堂のおばちゃんかな？

はやじ： ん？なに？

シオン： 28さんも何か作って下さーーい

へぴいちごqp： バリアブロックって何の為にあるの？

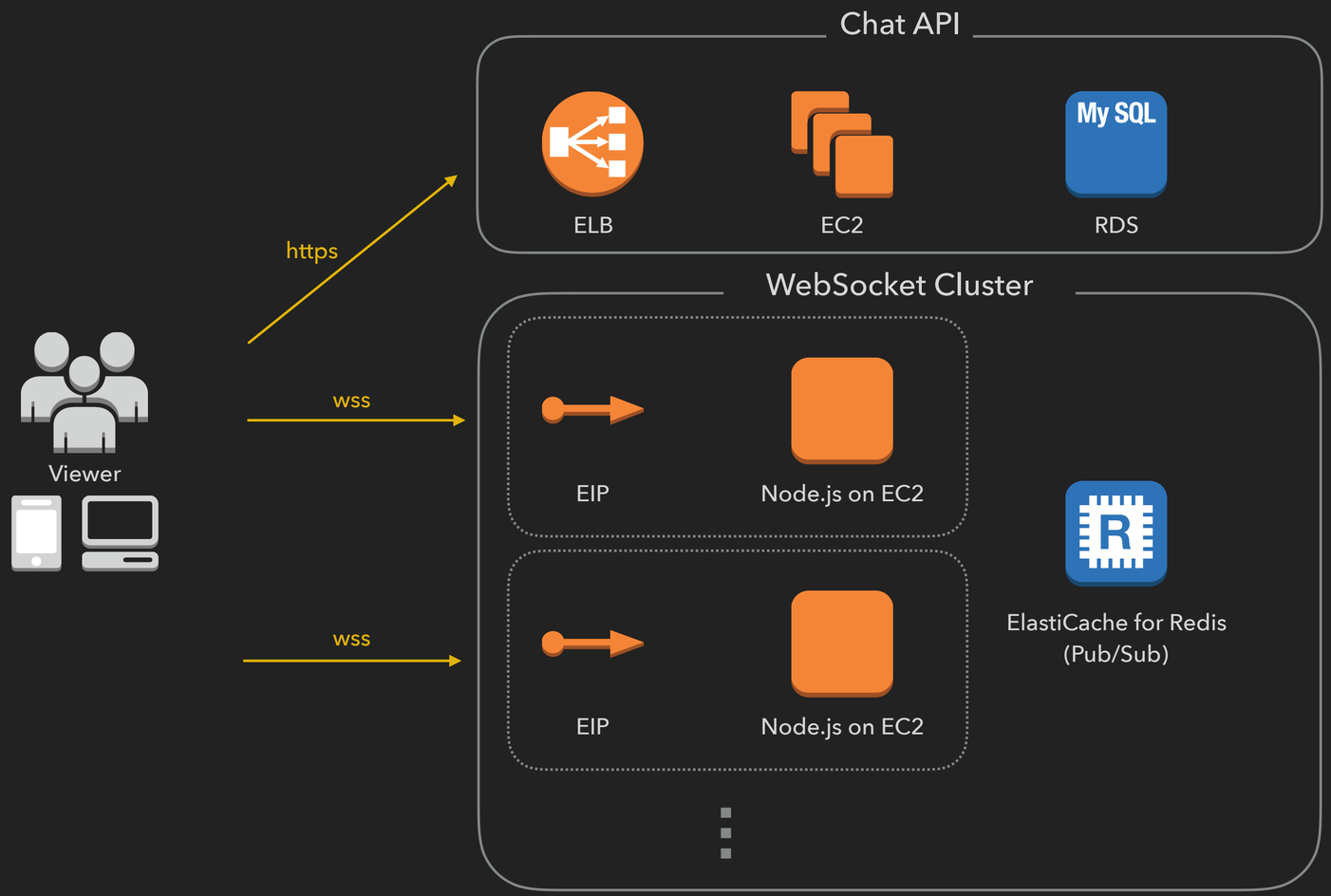
OPENREC.tv のチャットの要件

リアルタイムメッセージ配信基盤

- リアルタイム性
  - Online - Offline 間の臨場感の共有
  - 配信者 - 視聴者, 視聴者 - 視聴者 間の重要な対話手段
- 高拡張性
  - 同時視聴者数増に追従できるスケーラビリティ
  - 夜間に集中するライブ配信

OPENREC's Chat in 2015

1st Architecture: Node.js (WebSocket) + Redis (Pub/Sub) + API (ELB / EC2)

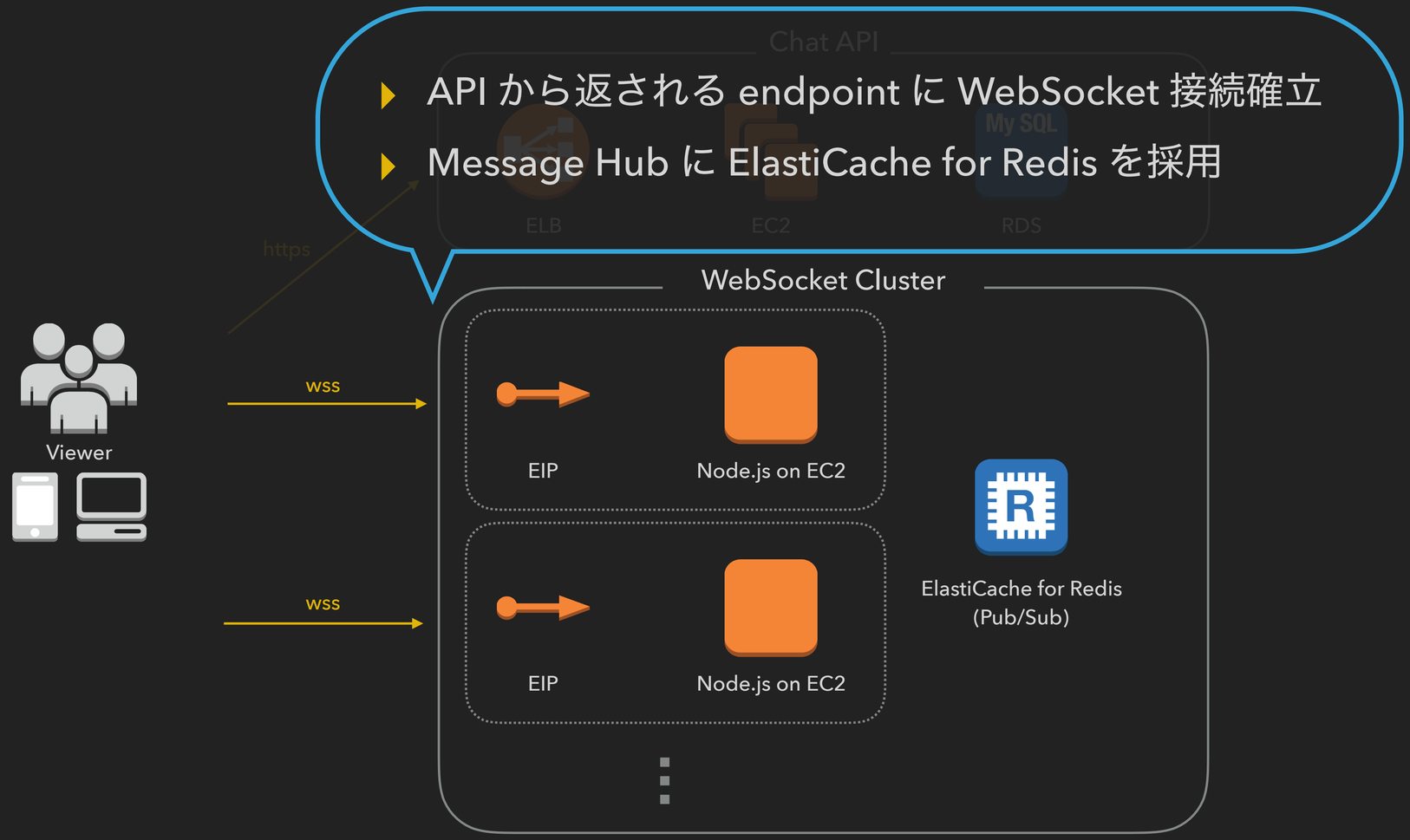


OPENREC's Chat in 2015

1st Architecture: Node.js (WebSocket) + Redis (Pub/Sub) + API (ELB / EC2)

OPENREC's Chat in 2015

1st Architecture: Node.js (WebSocket) + Redis (Pub/Sub) + API (ELB / EC2)



OPENREC's Chat

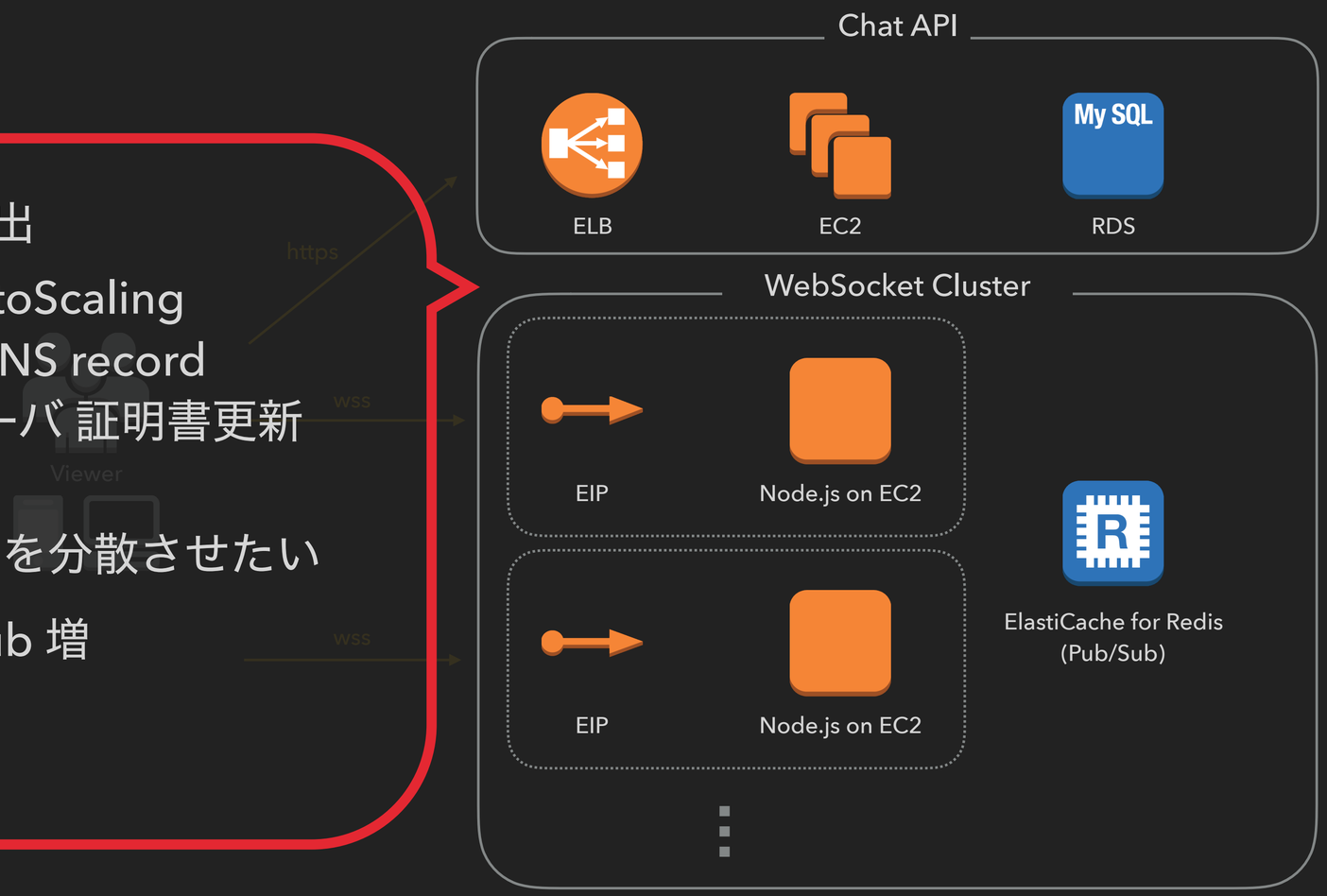
After 1 year . . .

煩わしさが噴出

- Not AutoScaling
- EIP + DNS record
- SSL サーバ 証明書更新

Redis Cluster を分散させたい

- Pub/Sub 増



【新発表】AWS アプリケーションロードバランサー

by AWS Japan Staff | on 12 AUG 2016 | in Amazon Elastic Load Balancing | Permalink

私たちはElastic Load Balancing (ELB)を2009年にAWSで発表しました(New Features for Amazon EC2: Elastic Load Balancing, Auto Scaling, and Amazon CloudWatchを見るとこの時からAWSがどれだけ変化してきたかがわかると思います)。Elastic Load BalancingはAWS上で動くアプリケーションのキーコンポーネントとなりました。Auto Scalingとの連携によって、Elastic Load Balancingは高い可用性を保ちながらアプリケーションをスケールアップ・ダウンする仕事を非常に簡単にしてくれました。

レベル

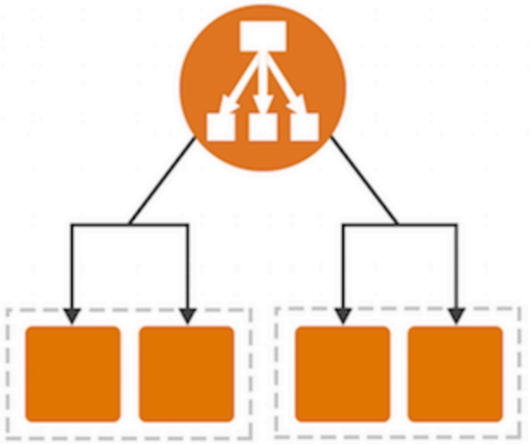
よく知られているOSIモデルでは、ロードバランサーは一般的にレイヤー4 (ネットワーク)かレイヤー7 (アプリケーション)で実行されます。

レイヤー4のロードバランサーはネットワークプロトコルのレベルで動作しネットワークパケットの中身を見ないので、HTTPやHTTPSの機能は無視します。別の言い方をすれば、これは全てを知る必要なく負荷を分散する仕組みになります。

レイヤー7のロードバランサーはより洗練されていて強力です。パケットの中を見て、HTTPとHTTPSのヘッダにアクセスし、(他の情報も合わせて)より賢い方法で負荷をターゲットに分散させることができます。

AWS上のアプリケーションロードバランサー

本日、私たちは新しいアプリケーションロードバランサーをELBの１つとして発表します。これはレイヤー7で実行され、幾つかの先進的な機能をサポートしています。元々のロードバランサー(これからはクラシックロードバランサーと呼ばれます)は引き続き利用可能で、レイヤー4とレイヤー7の両方の機能を提供します。



アプリケーションロードバランサーはコンテントベースのルーティングと、コンテナで実行されるアプリケーションをサポートします。業界標準の２つのプロトコル(WebSocketとHTTP/2)をサポートし、またターゲットのインスタンスやコンテナのヘルス状態の可視化も追加されています。コンテナやEC2インスタンスで動いているウェブサイトやモバイルアプリケーションは、アプリケーションロードバランサーを使うことで利益を得ることができるでしょう。

それでは、これらの機能を一つ一つ見ていって、最後に新しいアプリケーションロードバランサーを自身で作成してみましょう！

【新発表】AWS アプリケーションロードバランサー

by AWS Japan Staff | on 12 AUG 2016 | in Amazon Elastic Load Balancing | Permalink

私たちはElastic Load Balancing (ELB)を2009年にAWSで発表しました(New Features for Amazon EC2: Elastic Load Balancing, Auto Scaling, and Amazon CloudWatchを見るとこの時からAWSがどれだけ変化してきたかがわかると思います)。Elastic Load BalancingはAWS上で動くアプリケーションのキーコンポーネントとなりました。Auto Scalingとの連携によって、Elastic Load Balancingは高い可用性を保ちながらアプリケーションをスケールアップ・ダウンする仕事を非常に簡単にしてくれました。

レベル

よく知られているOSIモデルではロードバランサーは一般的にレイヤー4 (ネットワーク)かレイヤー7 (アプリケーション)で実行されます。

レイヤー4のロードバランサーはネットワークプロトコルのレベルで動作しネットワークパケットの中身を見ないので、HTTPやHTTPSの機能は無視します。別の言い方をすれば、これは全てを知る必要なく負荷を分散する仕組みになります。

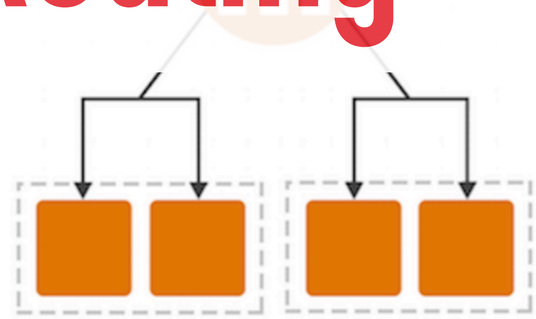
レイヤー7のロードバランサーはより洗練されていて強力です。パケットの中を見て、HTTPとHTTPSのヘッダにアクセスし、(他の情報も合わせて)より賢い方法で負荷をターゲットに分散させることができます。

AWS上のアプリケーションロードバランサー

本日、私たちは新しいアプリケーションロードバランサーをELBの1つとして発表します。これはレイヤー7ロードバランサーに幾つかの先進的な機能をサポートします。もちろんクラシックロードバランサー(以前はELBと呼ばれていました)は引き続き利用可能で、レイヤー4とレイヤー7の両方の機能を提供します。

アプリケーションロードバランサーはコンテントベースのルーティングと、コンテナで実行されるアプリケーションをサポートします。業界標準の２つのプロトコル(WebSocketとHTTP/2)をサポートし、またターゲットのインスタンスやコンテナのヘルス状態の可視化も追加されています。コンテナやEC2インスタンスで動いているウェブサイトやモバイルアプリケーションは、アプリケーションロードバランサーを使うことで利益を得ることができるでしょう。

それでは、これらの機能を一つ一つ見ていって、最後に新しいアプリケーションロードバランサーを自身で作成してみましょう！

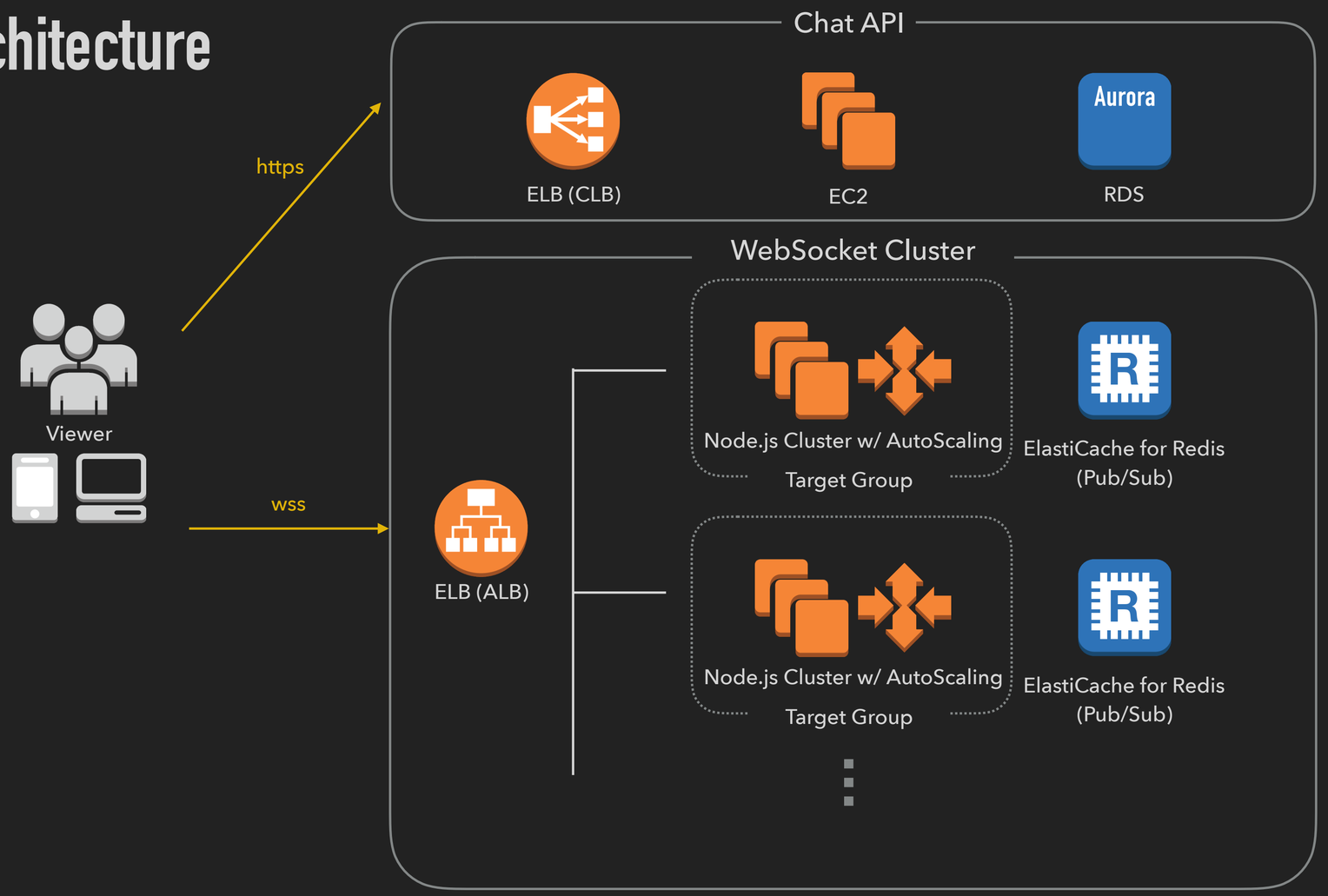


WebSocket

Content Based Routing

OPENREC's Chat

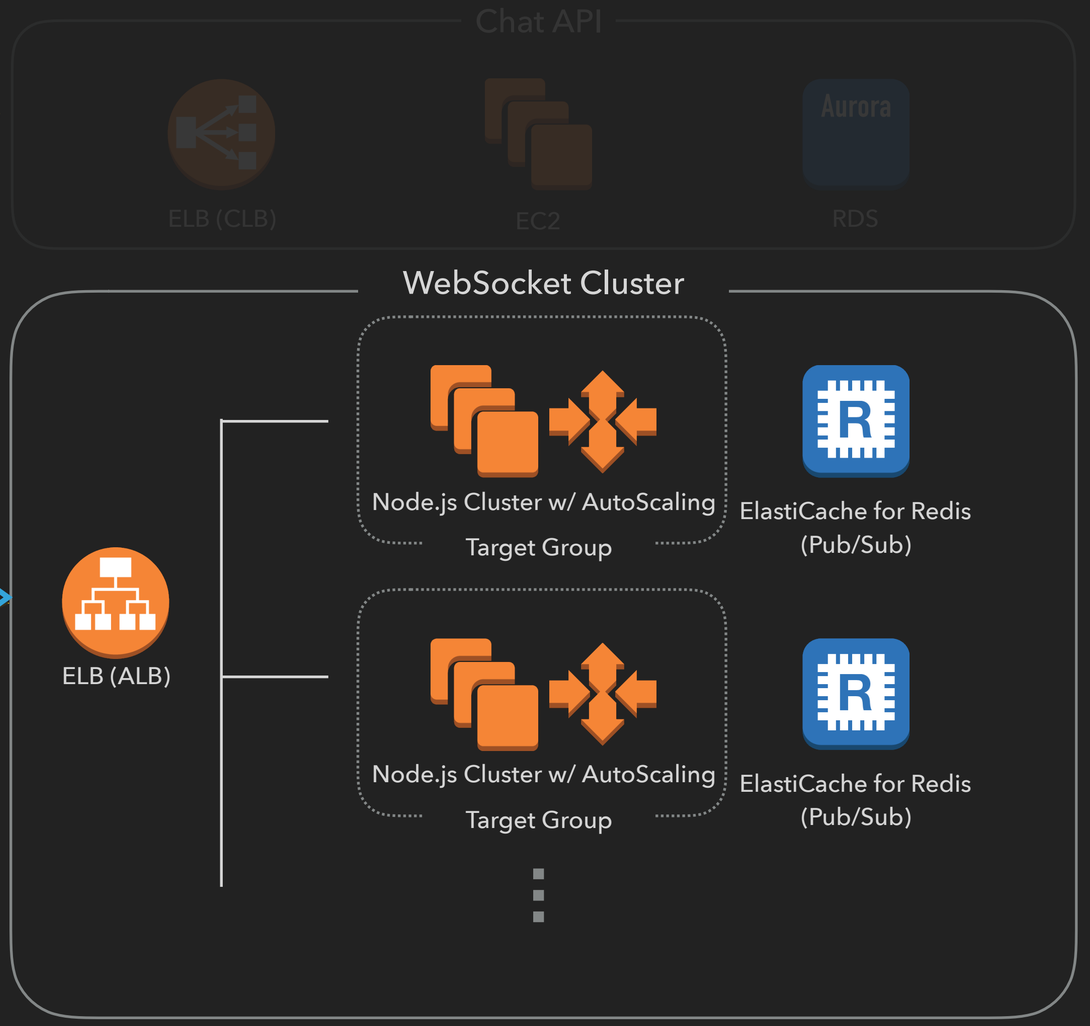
Replaced Architecture



OPENREC's Chat

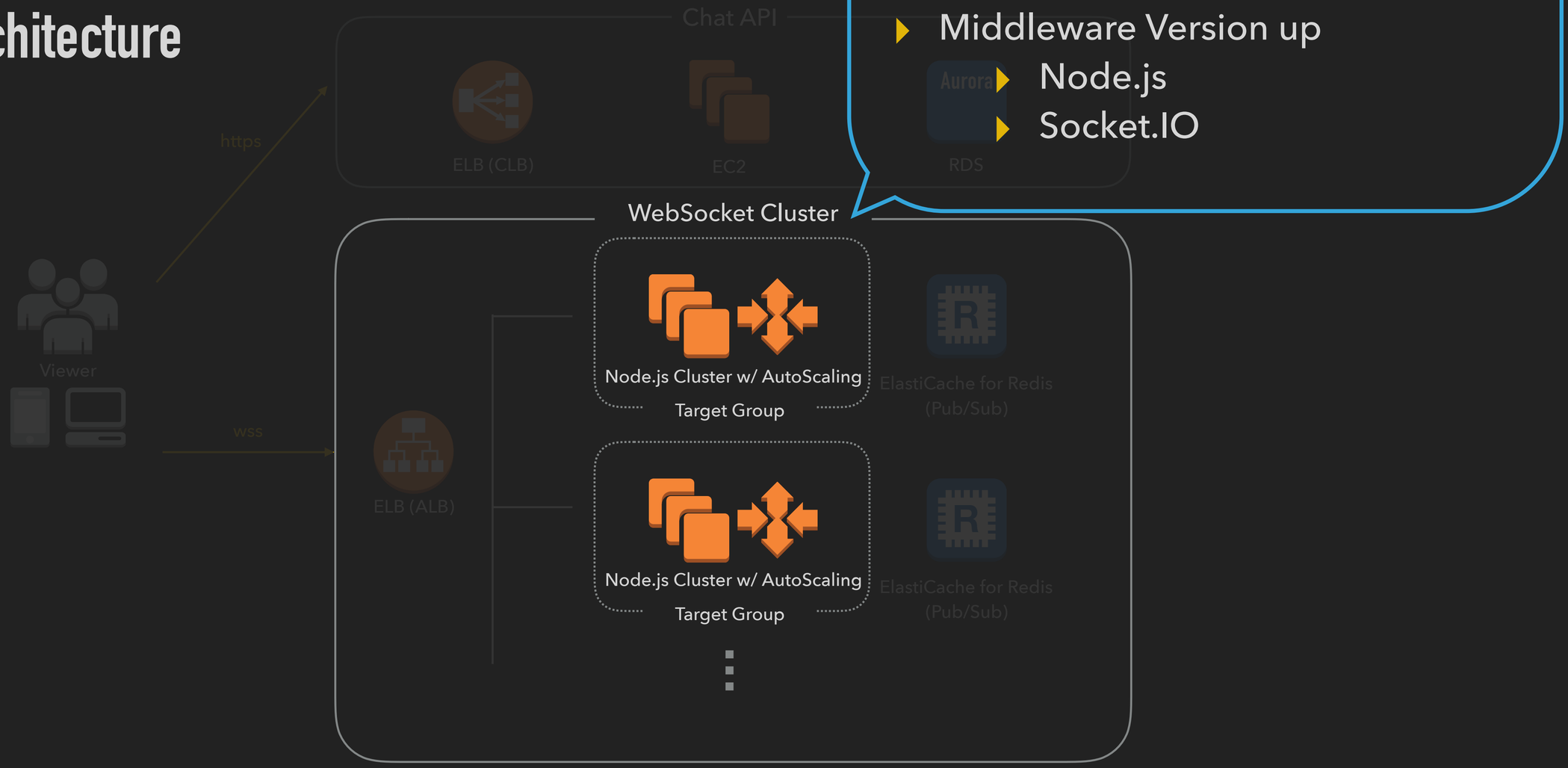
Replaced Architecture

- Contents Based Routing を活用した Sharding
- wss 接続終端による運用負荷減



OPENREC's Chat

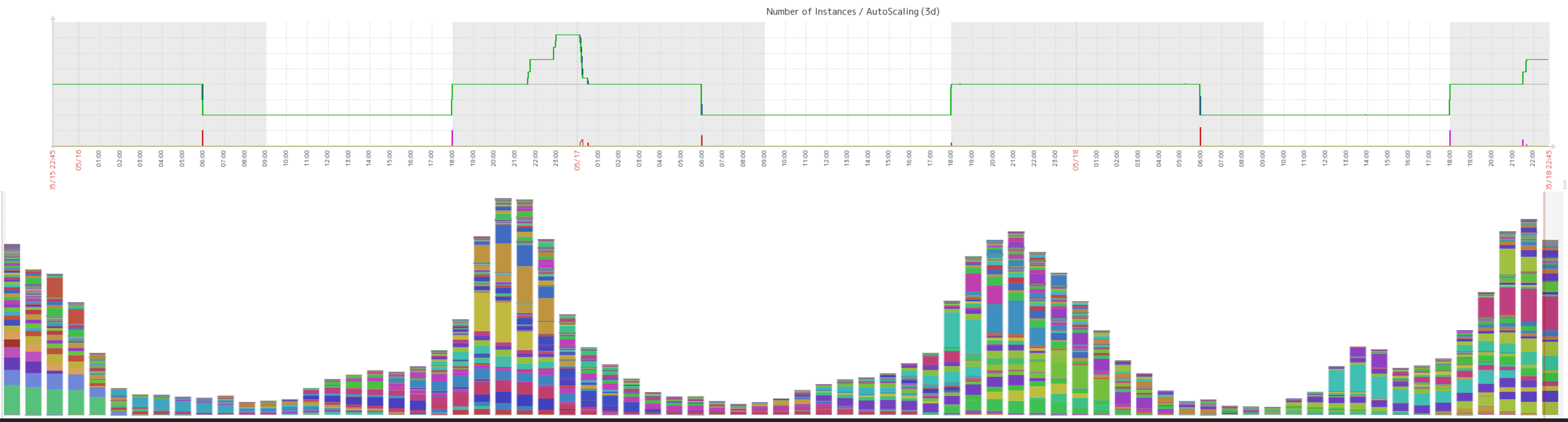
Replaced Architecture



- AutoScaling 活用による運用負荷減
- Middleware Version up
  - Node.js
  - Socket.IO

Results

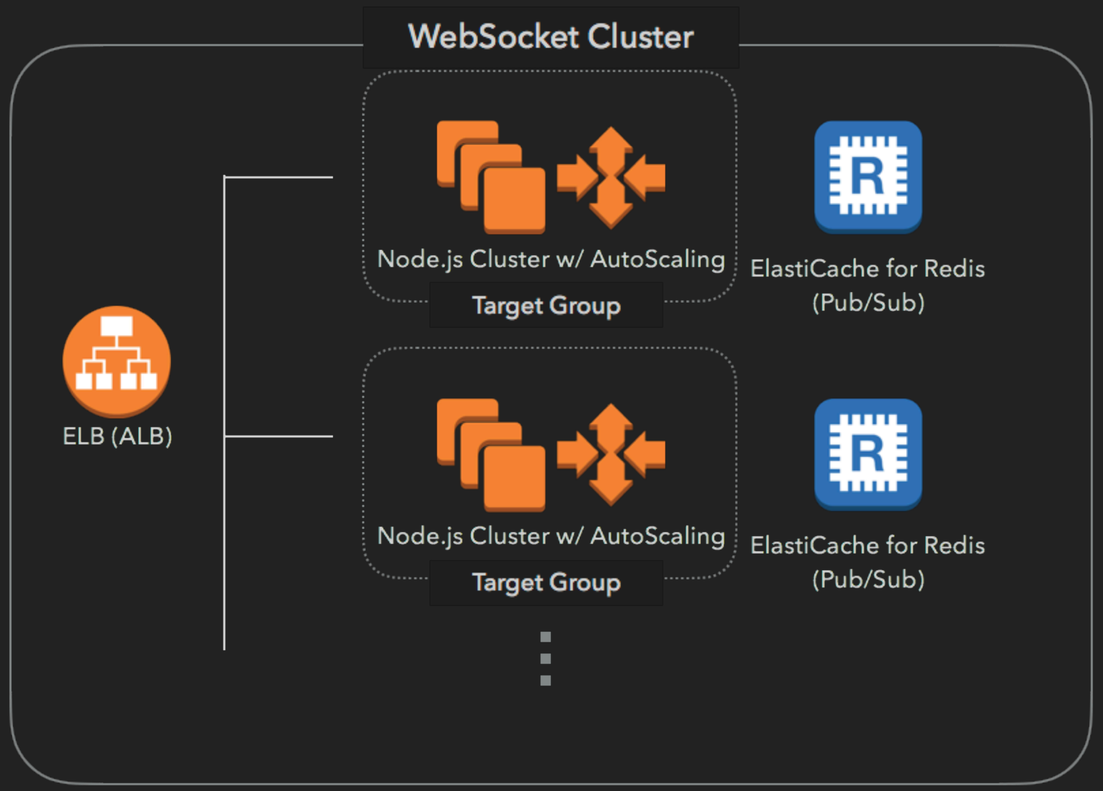
サービス負荷に対して柔軟なScaling を実現



Point of WebSocket Application w/ AutoScaling

Scaling において幾つかの注意が必要

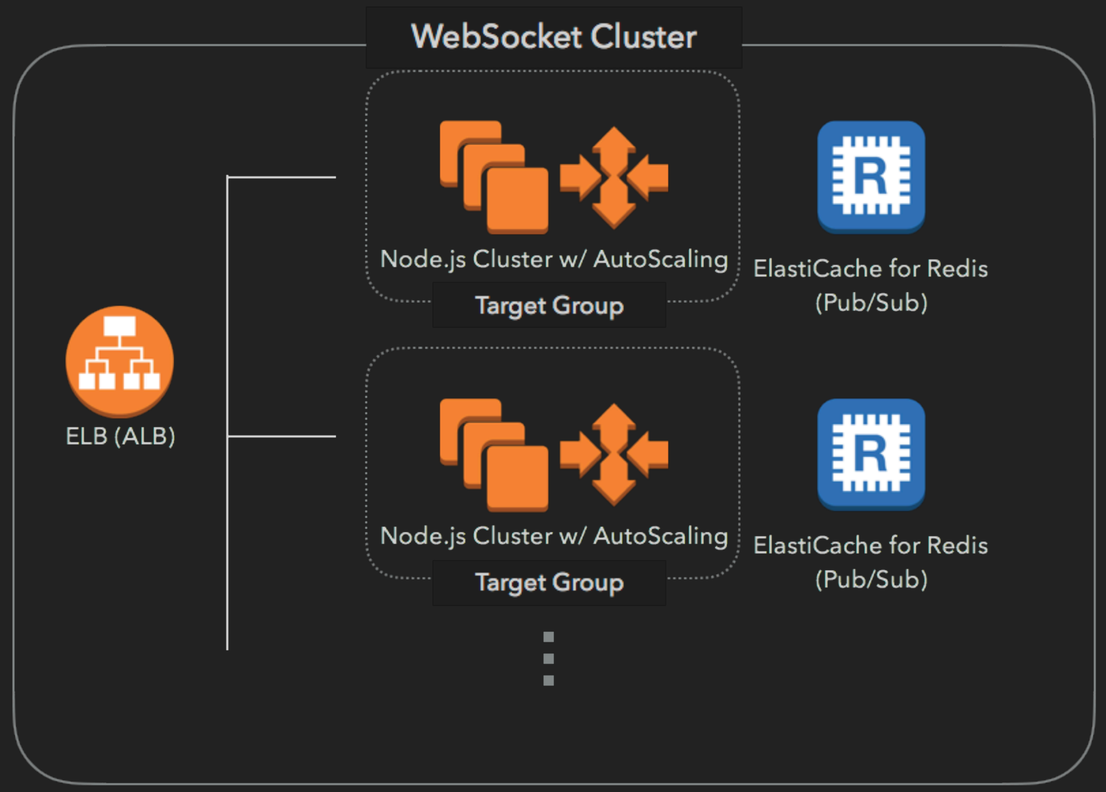
- 早めにスケールアウトさせる
  - WebSocket = 持続接続型プロトコル
  - 負荷が偏りすぎないように早めにスケールアウトさせると良い
- 重要な指標
  - 同時接続数
  - メッセージ流量（Pub/Sub 頻度）
  - メッセージサイズ



Point of WebSocket Application w/ AutoScaling

Scaling Policy

- シンプルに CPU Utilization がおすすめ
- 閾値を厳しめに 早めに Scale out
  - Increase は敏感に Decrease は緩く
- ピーク帯やイベントが読める場合は Scheduled Policy が有効



Caution of ALB w/ WebSocket

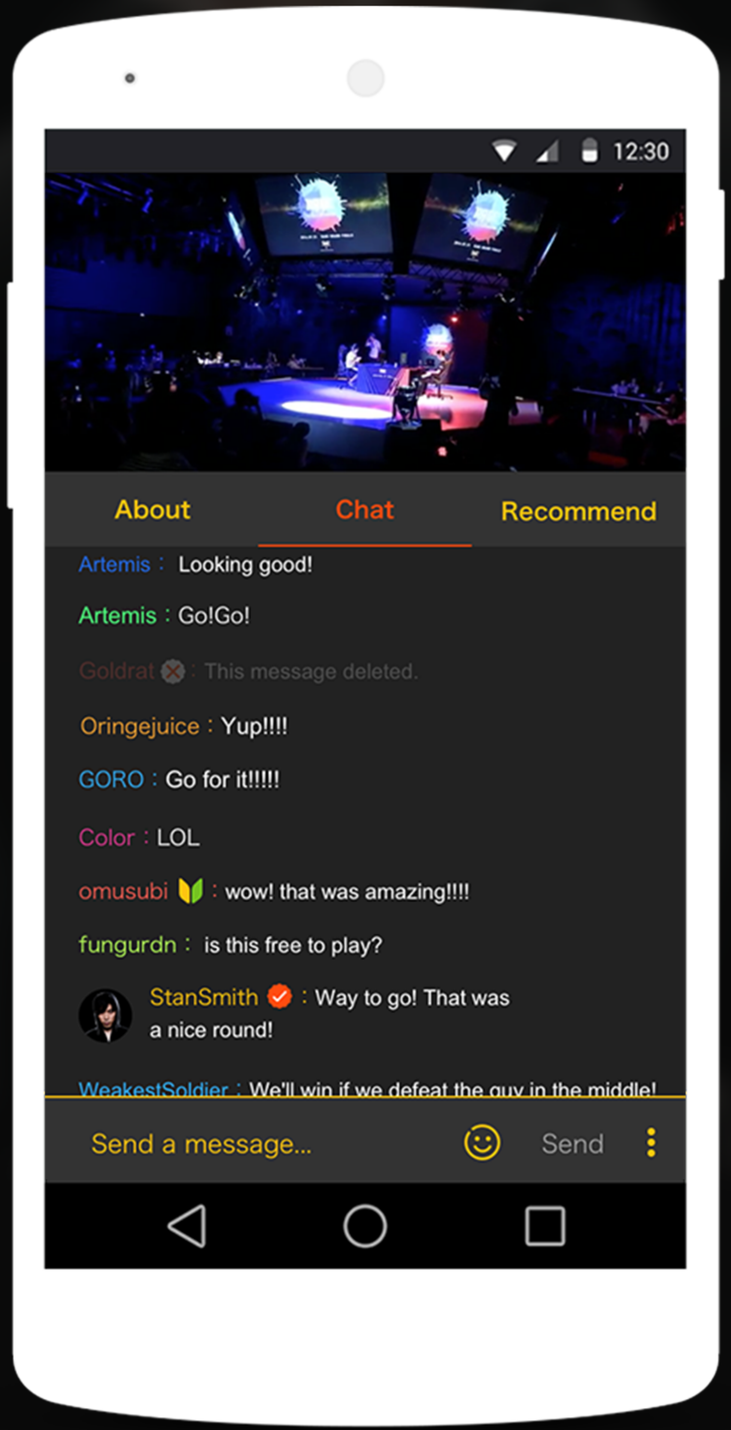
- 予期せぬ再接続に備える
  - ALB 内部ノードが Scale up / out するタイミングで一斉切断と再接続が走る
  - onConnect, onDisconnect のコールバック処理は流量制御しておくと良い

- Stickness 有効化 (Socket.IO の場合)
  - Handshake 処理 〜 Upgrade までの xhr-polling 通信時に必要
  - client によっては Upgrade 時に Cookie が渡らず Stickness が効かない事がある
    - socket.io-client-swift, socket.io-client-java で確認
  - 基本 Stickness を有効化し、ライブラリに応じて forceWebSocket して回避する

Requests to ALB

- Sticky Session の Custom Cookie 指定
  - CLB では可能
- ALB 内部のスケーリングの動作をシュミレーションしたい
  - 内部ノードの Scale up / out / in が発生した場合の、アプリケーション層に与える影響確認
- Leastconn Balancing Rule
  - 最も接続数の少ないバックエンドに優先的に Balancing したい
  - 特に WebSocket のような常時接続型通信の負荷を均等分散したい

Summary



OPENREC.tv

Grow larger, Start small

RUNNING LEAN

スタートアップにおける3つのステージ

集中：検証による学習
実験：ピボット

集中：成長
実験：最適化

RUNNING LEAN

学習に必要な 最も必要なことをやる

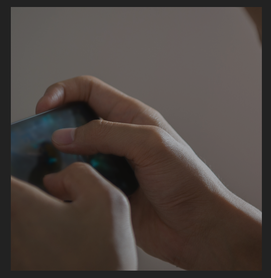
課題 / 解決
フィット

製品 / 市場
フィット

拡大

集中：検証による学習
実験：ピボット

集中：成長
実験：最適化

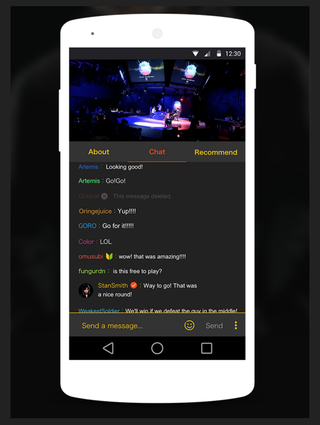


- ゲームプロモーションの問題と効果向上という命題
- プレイ動画共有という検証
- **スマホ特化 / SaaS 活用 / SDK 集中**

RUNNING LEAN

学習に必要な 最も必要なことをやる

課題 / 解決 フィット → **製品 / 市場 フィット** → 拡大



集中：検証による学習

実験：ピボット

- LIVE 配信にピボット
- **SaaS 活用，2 week Chat実装, UI/UX集中**

集中：成長

実験：最適化

RUNNING LEAN

Grow larger, Small Start

課題 / 解決
フィット

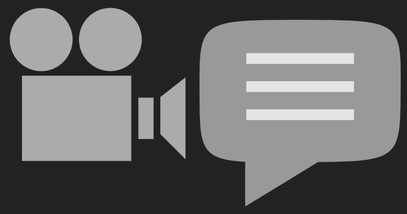
製品 / 市場
フィット

拡大

集中：検証による学習
実験：ピボット

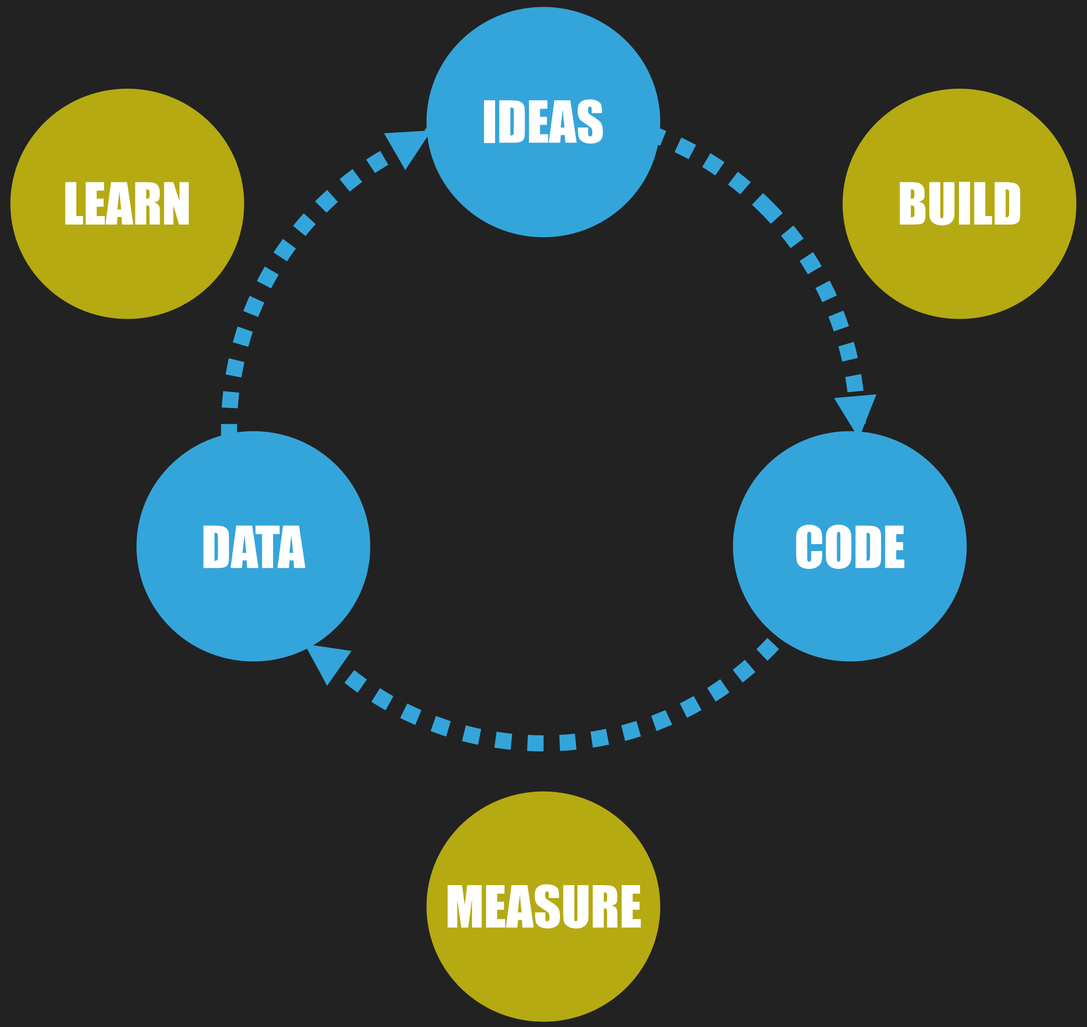
集中：成長
実験：最適化

- ユーザー急増
- **LIVE & CHAT 基盤の最適化**



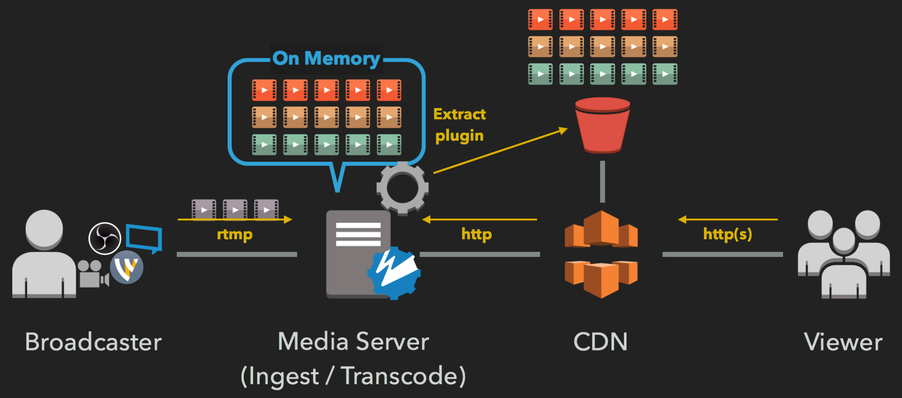
LEAN STARTUP

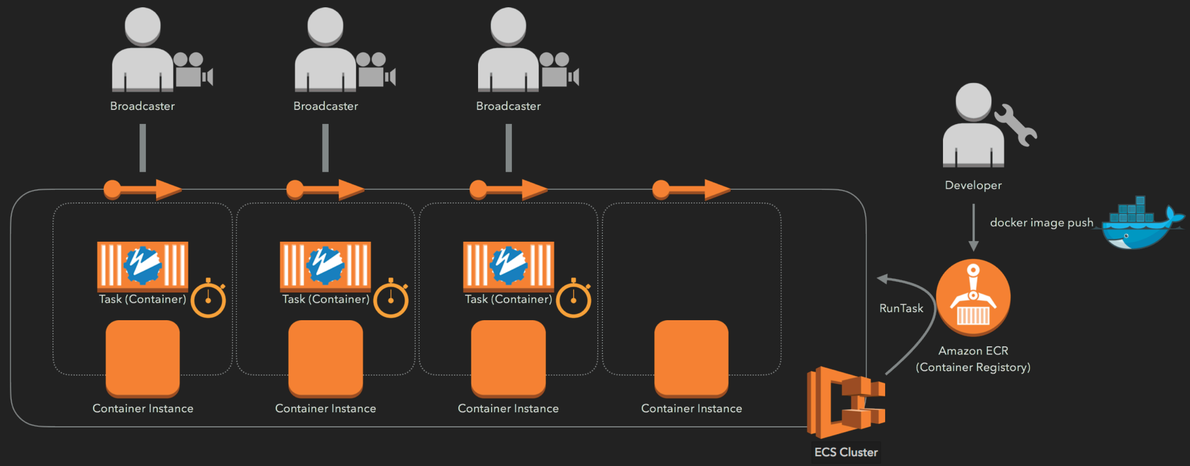
開発 - 計測 - 学習 のサイクルを如何に速く回せるか

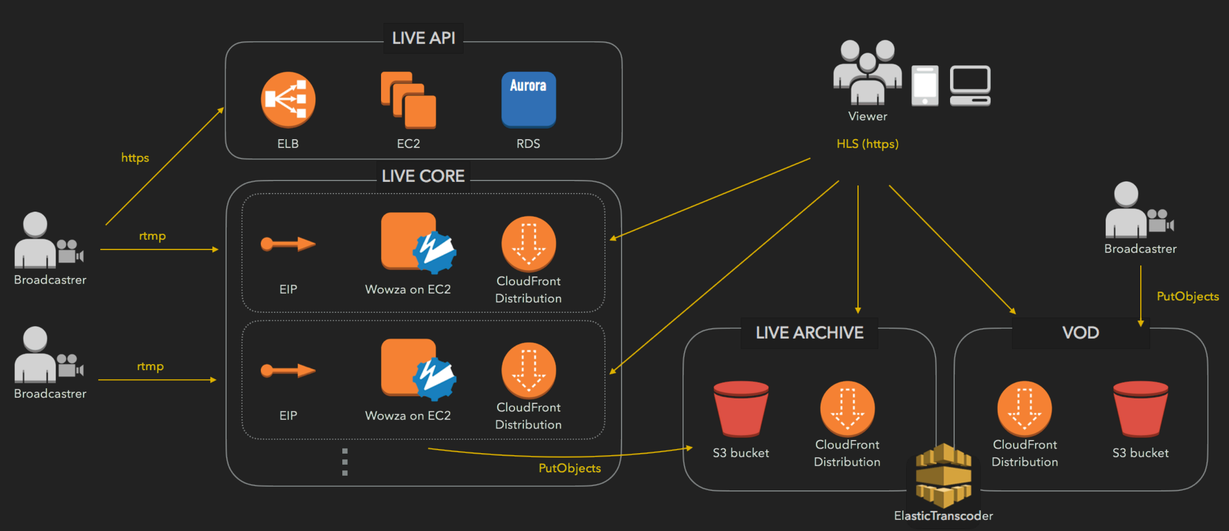


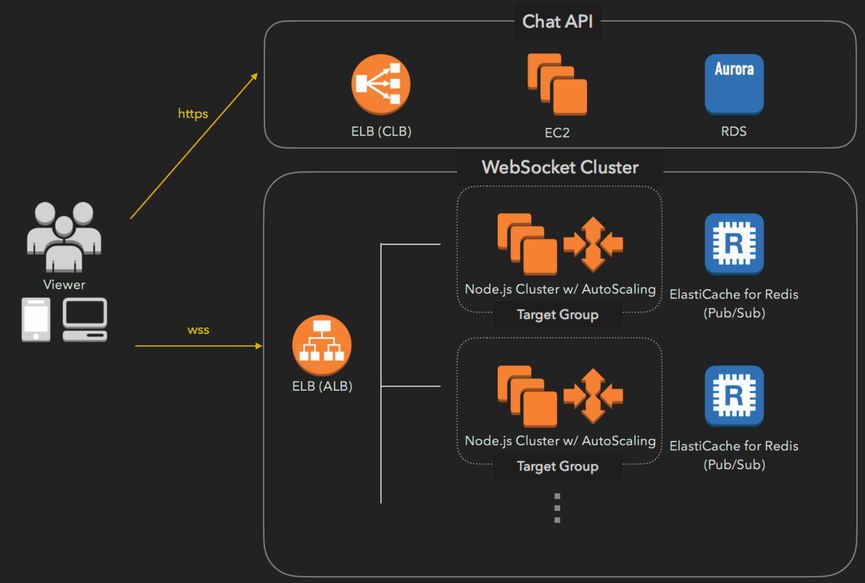
Why rebuild 4 times?

“The Perfect is often the enemy of the good”





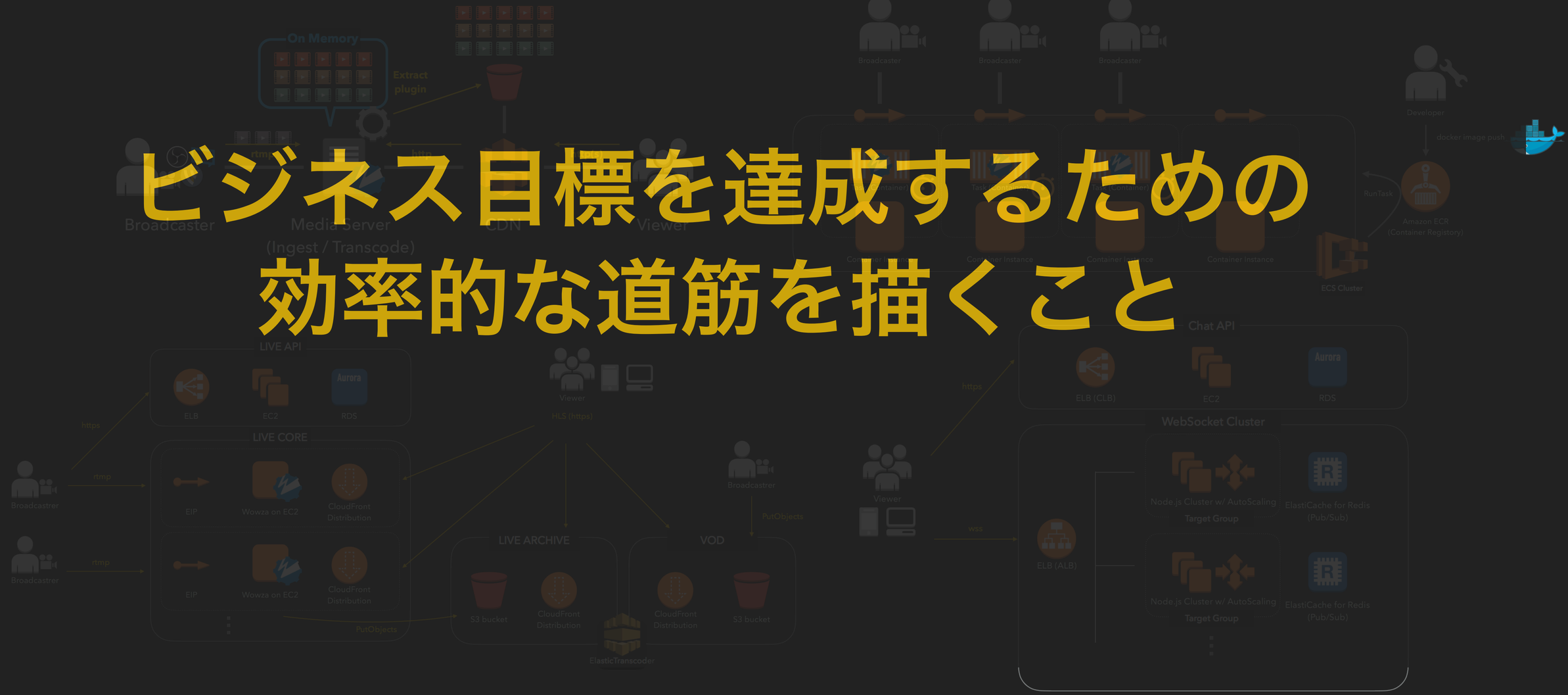




Why rebuild 4 times?

“The Perfect is often the enemy of the good”

ビジネス目標を達成するための
効率的な道筋を描くこと




OPENREC.tv