# Automatically Viewing the Most Interesting Streams on Twitch

Cullen Taylor



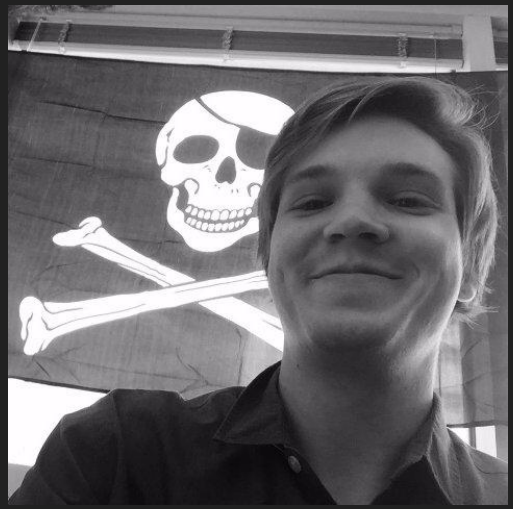
That’s a terrible title.

# Rotisserie

Cullen Taylor

# Who Am I?

- Developer Advocate at IBM.
- Focused on developing open source software around gaming and esports.
- Previously did DevOps-y stuff on various OpenStack deployments (both public and private).
- Graduated from Texas State University in 2015.
- Enjoy video games, playing guitar and drums, snowboarding, camping and road trips in my free time.



# What is Rotisserie?

- Rotisserie is a webapp for passively viewing active streams on twitch.tv.
- Borrows from NFL Redzone concept.
- Written in node.js, runs on Kubernetes and incorporates Tesseract's OCR engine.
- Still undergoing early active development with my colleagues.
- Currently focuses on a single game: PlayerUnknown's BattleGrounds.

# What’s with the chicken?

- Great question! I’ll answer with another question: What the heck is

# Or PUBG, for brevity's sake

- 100-player online Battle Royale game developed by BlueHole Studios in conjunction with Brendan Greene.
- Shares DNA with Arma III BR mod and H1Z1 KotK, but refines those concepts.
- Entered "early access" beta period via Steam on March 23rd, 2017.
- Gained traction quickly, has sold 13 million copies to date and has highest number of concurrent players on Steam.

# Basic Mechanics

- Players parachute from plane with no gear; must loot buildings for randomly-generated gear after landing.
- Can queue solo, duos, or 3/4 person squads.
- Last person standing wins (“Winner Winner Chicken Dinner”) by any means possible.
- Permadeath. Once you’re dead, you requeue. No respawns.
- Players forced to cluster together by a circle on the map which is constantly shrinking. Players outside the circle take damage until death.

# Basic Mechanics

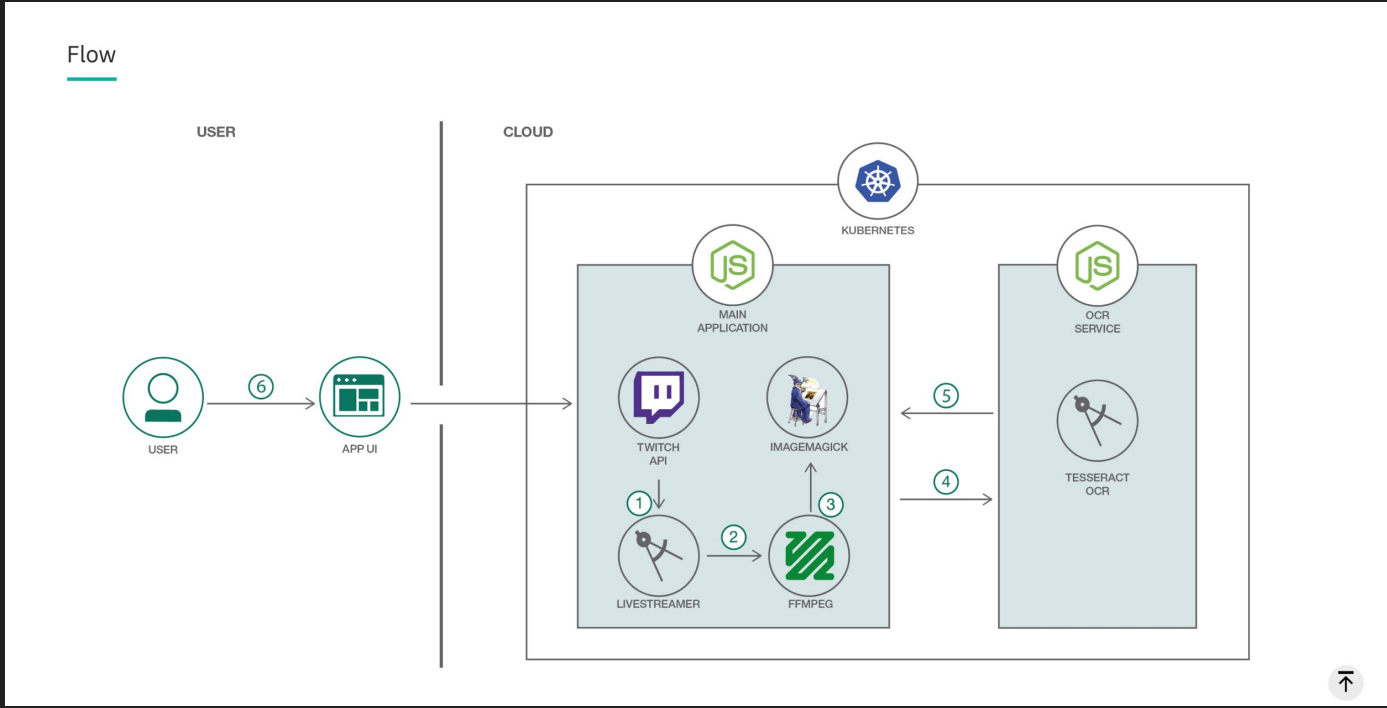
# Shrinking Circles

# The Use Case


Valve Squad Alpha
4
0:01
Round 12/30
7
Valve Squad Bravo
$11600
$22350
Viewers: 6,110
DROPPING ITEMS FOR:
SFOD-Delta
DSIX
Total items dropped this match: 14
$3400
100 Gautam
100 sockm
100 Piggles ULTRAPRO
100 Marvin
100 popdog
Vitaliy 100
ΦLynx 100
MagnarJ 100
chippy 100
[VALVE] Vitaliy
[Auto Director]
Knife
(GOTV) Mediocrity : Someone want to trade a DH box for two MP7 skullz skins?
(GOTV) shakwon : whats a dh box
(GOTV) irv : mediocrity: yes, add me
fps: 60 ping: 196 ms
loss: 0% choke: 0%
tick:128.0 sv:128.0 var: 5.7 dedicated
100
100

# The Use Case

# High Level Overview



# Twitch API Calls

- First, we grab a list of streamers currently streaming PUBG. Naturally, there is a node.js library for making Twitch api calls.
- Returns a JSON payload of streams.
- Filter by English language streams and streams which are not flagged as for mature audiences (more on that later), and just focus on the stream names from there.

```
function listStreams(twitch, callback) {
  let parameters = {"game": "PLAYERUNKNOWN\'S BATTLEGROUNDS",
    "language": "en", "length": 100};

  twitch.streams.live(parameters, function(err, body) {
    if (err) console.log(err);
    else {
      allAgesStreams = body.streams.filter(function(stream) {
        return stream.channel.mature == false;
      });
      return callback(allAgesStreams);
    }
  });
}
```


```
{
  "mature": true,
  "partner": false,
  "status": "TSM_BreaK // Full Speed 150k Follower Hype",
  "broadcaster_language": "en",
  "display_name": "BreaK",
  "game": "PLAYERUNKNOWN'S BATTLEGROUNDS",
  "language": "en",
  "_id": 37851229,
  "name": "break",
  "created_at": "2012-11-20T23:51:48Z",
  "updated_at": "2017-10-05T13:03:56Z",
  "delay": null,
  "logo": "https://static-cdn.jtvnw.net/jtv_user_pictures/break-profile_image-1c76d2b63ee3a3d7-300x300.png",
  "banner": null,
  "video_banner": "https://static-cdn.jtvnw.net/jtv_user_pictures/d8e33c2f12ebe30a-channel_offline_image-1920x1080.png",
  "background": null,
  "profile_banner": "https://static-cdn.jtvnw.net/jtv_user_pictures/d19fcd661c3af7a8-profile_banner-480.png",
  "profile_banner_background_color": "#000000",
  "url": "https://www.twitch.tv/break",
  "views": 5969370,
  "followers": 153415,
  "_links": {
    "self": "https://api.twitch.tv/kraken/channels/break",
    "follows": "https://api.twitch.tv/kraken/channels/break/follows",
    "commercial": "https://api.twitch.tv/kraken/channels/break/commercial",
    "stream_key": "https://api.twitch.tv/kraken/channels/break/stream_key",
    "chat": "https://api.twitch.tv/kraken/chat/break",
    "features": "https://api.twitch.tv/kraken/channels/break/features",
    "subscriptions": "https://api.twitch.tv/kraken/channels/break/subscriptions",
    "editors": "https://api.twitch.tv/kraken/channels/break/editors",
    "teams": "https://api.twitch.tv/kraken/channels/break/teams",
    "videos": "https://api.twitch.tv/kraken/channels/break/videos"
  }
}
```

# Recording a Stream

- Unfortunately, there is no good solution in node.js for recording raw twitch streams.
- Luckily, there is a python package called Livestreamer which makes this pretty easy.
- For each stream, we concurrently spawn a livestreamer process to record a second or two of footage, then kill it.

```
function recordStream(options) {
  return new Promise((resolve, reject) => {
    console.log("recording clip of stream: " + options.streamName);
    const child = spawn("livestreamer", ["--yes-run-as-root", "-Q", "-f",
      "--twitch-oauth-token", process.env.token,
      "twitch.tv/" + options.streamName, "720p", "-o",
      options.clipsDir + options.streamName + ".mp4"]);
    setTimeout(function() {
      child.kill("SIGINT");
      console.log("recorded stream: " + options.streamName);
      resolve(options);
    }, 4000);
  });
}
```

# Taking a Screenshot

- Now that we have some footage of a stream, we take a screenshot in order to get a single frame.
- fluent-ffmpeg , a node.js library for ffpmeg, makes this easy.

```
function takeScreenshot(options) {
  return new Promise((resolve, reject) => {
    if (fs.existsSync(options.clipsDir + options.streamName + ".mp4")) {
      console.log("taking screenshot of stream: " + options.streamName);
      new FFMpeg(options.clipsDir + options.streamName + ".mp4")
        .takeScreenshots({
          count: 1,
          folder: options.thumbnailsDir,
          filename: options.streamName + ".png",
        })
        .on("end", function() {
          resolve(options);
        });
    }
  });
}
```

# Taking a Screenshot

# Cropping a Screenshot

- Now, we can further utilize ImageMagick to crop down the screenshot to just the number of players alive.


```
function cropScreenshot(options) {
  return new Promise((resolve, reject) => {
    console.log("cropping screenshot of stream: " + options.streamName);
    if (fs.existsSync(options.thumbnailsDir + options.streamName + ".png")) {
      gm(options.thumbnailsDir + options.streamName + ".png")
        .crop(28, 20, 1190, 25)
        .write(options.cropsDir + options.streamName + ".png", function(err) {
          resolve(options);
          if (err) reject(err);
        });
      console.log("cropped screenshot of: " + options.streamName);
    } else {
      reject(new Error(options.streamName + ": input file not found"));
    }
  });
}
```

# Tesseract in a Microservice

- Finally, we can pass our cropped screenshot containing just a number to our OCR microservice.
- Tesseract is a fairly robust open source OCR engine.
- Make a POST request to the OCR microservice's endpoint. It will return a string containing the number in the box.
- Assign the return value to the stream name and throw that in an array.

```
function ocrCroppedShot(options) {
  return new Promise((resolve, reject) => {
    let formData = {
      image: fs.createReadStream(__dirname
                                 + options.cropsDir.replace(".", "")
                                 + options.streamName + ".png"),
    };

    let requestOptions = {
      url: "http://" + process.env.OCR_HOST + "/process_pubg",
      formData: formData,
    };

    request.post(requestOptions, function(err, httpResponse, body) {
      if (err) {
        console.error("upload failed");
        reject(err);
      } else {
        let parsed = JSON.parse(body);
        let object = {};
        object.name = options.streamName;
        object.alive = parsed.number;
        resolve(object);
      }
    });
  });
}
```

# Play it again, Sam!

- From here, we just repeat the process on an interval.
- Each iteration will update the array of stream names and sort by the number of players alive.
- The stream with the lowest number of players alive will be determined to be the “most interesting”, and some client-side Javascript will refresh the stream iframe.
- Users can “dim the lights” and pin a stream if they like it to stop auto-switching.

# Problems and Caveats

- Lots of file operations, so lots of busywait while I/O and promise resolutions are handled.
- Many external dependencies were taken on in getting out of POC mode, resulting in lots of exception handling.
- Streamer overlays can get in the way of the number of people alive.
- Tesseract’s OCR engine can be unreliable, resulting in aberrant stream switching.
- Adblock doesn’t catch streamer ads (but that’s not all bad!).

# Future Plans

- An IBM Developer Journey (blog post + code) to be released soon.
- Want to flesh out back to the original concept of doing visual recognition for a game with a busier UI (CS:GO, Overwatch, etc).
- Exploring the possibilities of Twitch's new extensions model once more info on them has been released. Could make it easier to integrate directly with streams rather than scraping frames.

# Check It Out!

- github.com/IBM/rotisserie
- https://pubgred.zone , soon to be at https://rotisserie.tv

# Thanks!

github.com/eggshell
@eggshellcullen
blog.eggshell.me/seagl
mctaylor@us.ibm.com