UNIVERSITAT DE BARCELONA

PixelCanvas.io

Creación de un juego colaborativo masivo en línea

Trabajo de Fin de Grado

Autor: Rafael Arquero Gimeno

Directora: Dr. Maite Lopez-Sanchez

Barcelona, 27 de junio de 2018

Abstract

On April 1, 2017, on the occasion of the April Fools' Reddit (one of the most popular websites on the Internet) launches /r/place: a blank canvas that their users could edit by changing the color of a pixel every 5 minutes from a 16-colors palette.

At the end of this collaborative art experiment that lasted 72 hours, several websites immediately appeared that recreated the same dynamic, including PixelCanvas.io, which offers an eternal canvas of 2,000,000 x 2,000,000 pixels, anonymity and variable waiting times.

Of all the clones that have emerged, this is the most successful: as of today, 40,000,000 users have placed a pixel, the page has achieved at its peak in the Top 6,000 of Alexa and appear in the Top 100 of Brazil and Mexico.

This work explains the creation by me of PixelCanvas.io, the infrastructure that makes its operation possible, the technical solutions to the problems caused by success and all the source code of the project.

Resumen

El 1 de Abril de 2017, con motivo del *April Fools'* (día de los inocentes en EE.UU.) Reddit (uno de los sitios web mas populares en Internet) lanza /r/place: un lienzo en blanco que los usuarios de Reddit podían editar cambiando el color de un pixel cada 5 minutos a partir de una paleta de 16 colores.

Al terminar este experimento de arte colaborativo que duró 72 horas, inmediatamente surgieron varios webs que recreaban la misma dinámica, entre ellas PixelCanvas.io, que ofrece un lienzo eterno de 2.000.000 x 2.000.000 de pixeles, anonimato y tiempos de espera variables.

De entre todos los clones surgidos, este es el que más éxito tiene: a fecha de hoy 40.000.000 de usuarios han colocado algún pixel, la pagina ha logrado en su apogeo colocarse en el Top 6.000 de Alexa y aparecer en el Top 100 de Brasil y Mexico.

Este trabajo relata la creación por mi de PixelCanvas.io, la infraestructura que hace posible su funcionamiento, las soluciones técnicas a los problemas ocasionadas por éxito y todo el código fuente del proyecto.

Resum

L'1 d'abril de 2017, amb motiu de l'April Fools' (Dia dels innocents als EUA) Reddit (un dels llocs web mes visitats d'Internet) llança **/r/place**: un llenç en blanc que els usuaris de Reddit podien editar canviant el color d'un píxel cada 5 minuts a partir d'una paleta de 16 colors.

En acabar aquest experiment d'art col·laboratiu que va durar 72 hores, immediatament van sorgir diversos webs que recreaven la mateixa dinàmica, entre elles PixelCanvas.io, que ofereix un llenç etern de 2.000.000 x 2.000.000 de píxels, anonimat i temps d'espera variables.

D'entre tots els clons sorgits, aquest és el que més èxit té: a data d'avui 40.000.000 d'usuaris han col·locat algun píxel, la pagina ha aconseguit en el seu apogeu col·locar-se en el Top 6.000 d'Alexa i aparèixer en el Top 100 de Brasil i Mèxic.

Aquest treball relata la creació per mi de PixelCanvas.io, la infraestructura que fa possible el seu funcionament, les solucions tècniques als problemes ocasionades per èxit i tot el codi font del projecte.

Muchas gracias a Helena y mi tutora Maite.

Índice

1. Introducción

Este proyecto fue creado motivado por el experimento temporal de **/r/place** de **Reddit,** un lienzo en blanco de 1000 x 1000 pixeles que los usuarios de Reddit podían editar cada 5 minutos seleccionando un color de una paleta de 16 colores.

Al finalizar el experimento, algunos usuarios con los conocimientos técnicos re-crearon la plataforma y surgieron multitud de **clones**.

Yo fui usuario del /r/place, así como de otros clones como colorthis.space o pxls.space. Pero todos parecían reproducir los mismos problemas que tenia el original así que decide crear el mío propio para promover la **co-creacion artística**.

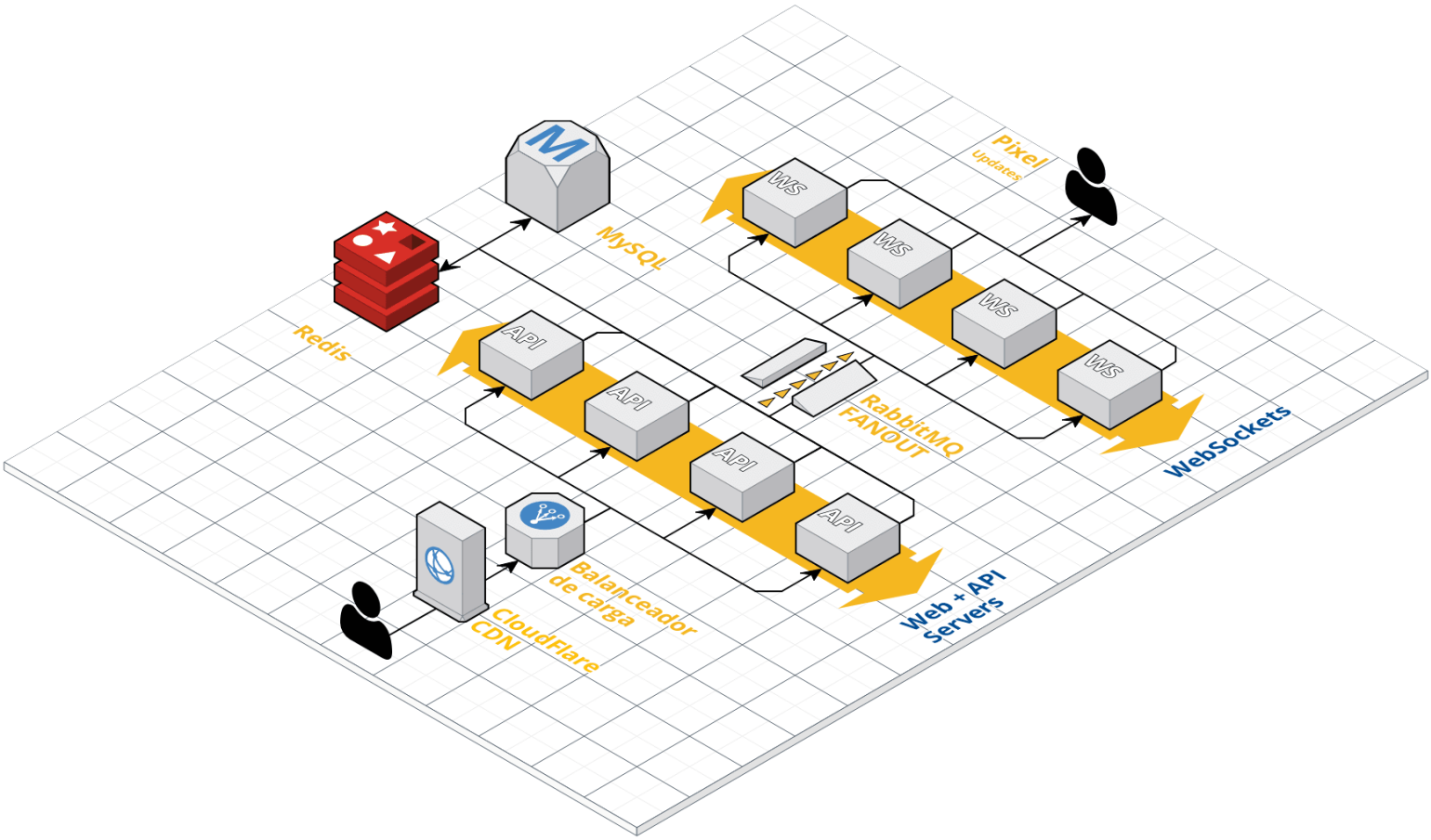
Pensé que si lo dotaba de un **espacio infinito**, todos los creadores de arte podrían encontrar su lugar, y así evitar la lucha por el espacio que se daba entre las distintas facciones en los otros sitios.

También decidí aplicar un **tiempo de espera variable** en función de la distancia del pixel al centro, siendo 0 segundos en el centro y aumentando de forma logarítmica a medida que se aleja de este. Para diferenciarme del resto de sitios que aplicaban la mismo tiempo de espera.

Para mantenerlo **extremadamente simple**, que para jugar bastase con un clic, decidí hacerlo **anónimo**, basándome en la IP, a diferencia del resto de sitios que aplicaban un sistema de usuarios.

Como el frontend debe ser hecho en **JavaScript**, decidí programar el backend también en JavaScript por simplicidad y subirlo a DigitalOcean también por simplicidad frente a *Amazon Web Services*.

2. Diseño



Arquitectura del Backend de PixelCanvas.io en DigitalOcean

El objetivo de PixelCanvas es construir arte colaborativamente a un nivel **masivo**. Es por eso que en la arquitectura cliente-servidor de esta web la complejidad recae en el backend mientras que el frontend es bastante más simple.

El Frontend es una web app hecha en **React**. Consta de una etiqueta ***<canvas>*** que se expande ocupando toda la pantalla del usuario y que muestra los pixeles del lienzo. También hay una interfaz minimalista para que el usuario se pueda desplazar libremente por el inmenso mapa.

Toda interacción entre el cliente y el backend pasa por una **API REST**. Las actualizaciones del mapa se van recibiendo en tiempo real a través de una conexión **WebSocket**.

El Backend de PixelCanvas emplea la infraestructura que se puede ver el esquema. Dado que se pretende dar servicio a una cantidad masiva se usuarios (en el pico de actividad, 16.000 usuarios conectados en el mismo momento y mas de 1.000.000 a lo largo del día) se descartó un diseño monolítico y se ha optado por crear un sistema distribuido **resilente** y **escalable**.

De cara al público, se delega en **CloudFlare** para recibir y responder las peticiones HTTP con el objetivo de minimizar las peticiones/segundos que se van a recibir en nuestra infraestrucura. CloudFlare, cuando no save resolver una petición, llama a nuestro balanceador de carga.

El **balanceador de carga**, controla en todo momento el estado de nuestros servidores web y selecciona por Round-Robin quien resolverá la petición que ha recibido de CloudFlare.

Los **servidores web** sirven los contenidos estáticos del cliente y resuelven las peticiones a la API. Su numero es variable y se pueden crear o destruir fácilmente según la demanda que se prevea, i.e., es fácilmente escalable.

Para coordinar todos estos servidores web, hace falta un espacio en común para compartir datos. Esta función la desempeña el servidor **Redis**, que alberga el estado común de los servidores web: como la información de los pixeles del lienzo, tiempos de espera de cada usuario, *locks* a recursos críticos, control del *throttle*, etc.

Como el estado del lienzo es altamente cambiante en tiempo real, hace falta mantener actualizado al cliente. Como la API REST no es **full-duplex**, ya que funciona sobre HTTP, se descartó enviar las actualizaciones a través de este canal y se vio necesario hacer uso de **WebSockets**.

Otra vez, un solo servidor WebSocket seria insuficiente para abastecer a todos los clientes, por lo que es necesario contar con un numero variable de estos.

Finalmente, la pieza que falta por detallar es **RabbitMQ**. RabbitMQ actúa como **abstracción** de este enjambre de servidores WebSocket. Cuando alguien quiera hacer un **broadcast** a todos nuestros clientes conectados, simplemente tendrá que publicar un mensaje en la cola de RabbitMQ y esta se encarga de distribuirlo a todos los servidores WebSocket para que lo envien. Así de esta manera, también se garantiza el orden en el que se reciben las actualizaciones de los pixeles.

Accesoriamente, se ha añadido un servidor **MySQL** para tareas de **administración**, como bloquear a ciertos usuarios.

3. Implementación

TODO bla bla bla

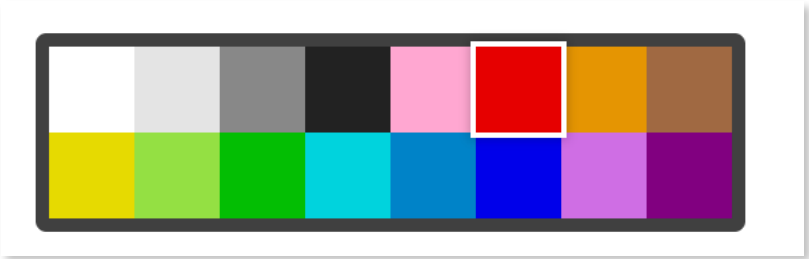
Dar detalles concretos de:

- Las tecnologías utilizadas para el desarrollo
- aspectos interesantes que haya que programar (como por ejemplo lo que explicas de cómo se almacenan los bits…)
- puedes incluir aspectos de planificación de la ejecución del proyecto: tiempo de desarrollo, coste asociado…

3.1 Estructura del Canvas

El lienzo se almacena en **Redis**[1] de una manera bastante peculiar que pasaré a relatar a continuación.

Primero de todo, es necesario **codificar** de algún modo los colores de la **paleta**. Asignamos un entero a cada color de la siguiente forma: determinamos que el blanco sea el 0 y asignamos los siguientes números de manera sucesiva de izquierda a derecha y de arriba a bajo.



Interfaz de la paleta de colores de PixelCanvas

Como hay un total de 16 colores bastan **4 bits** para representar cada color. Por ejemplo, el color rojo seleccionado en la imagen, tiene asignado el numero 5 y es representado por la secuencia binaria 0101.

Luego subdividimos la cuadricula del Canvas en cuadrados de 64x64 pixeles que llamaremos chunks de manera que al pixel situado en las coordenadas (x, y) le corresponde el chunk :

$$(\lfloor \frac{x}{64} \rfloor, \lfloor \frac{y}{64} \rfloor)$$

Cada chunk no almacena los pixeles ni en forma de matriz ni de lista, sino que tiene concatenada la representación binaria de todos los pixeles que contiene. Así pues, un chunk es una binario de 16384 bits = 2048 Bytes = 2KB.

$$64 \cdot 64 \cdot 4 = 16384$$

[1] **Redis** es un motor de base de datos en memoria, basado en el almacenamiento en hash tables (clave/valor).

La posición que le corresponde a cada pixel dentro de esta secuencia se calcula de la siguiente forma:

```
export function getOffsetOfPixel(x: number, y: number): number {
  const cx = mod(x, 64);
  const cy = mod(y, 64);
  return (cy * 64) + cx;
}
```

Cada chunk *(i, j)* se almacena en Redis bajo el key *"chunk:i:j"*. Para modificar un pixel en concreto dentro de Redis, se usa el comando **BITFIELD**[2], invocado del a siguiente forma:

```
const [i, j] = getChunkOfPixel([x, y]);
const args = [`chunk:${i}:${j}`, 'SET', 'u4', `#${getOffsetOfPixel(x, y)}`, color];
await redis.sendCommandAsync('bitfield', args);
```

Por defecto, no se crea ningún chunk, sino que antes de escribir un pixel, se comprueba que el chunk exista y en caso de no existir, se crea el chunk escribiendo una secuencia de 16384 ceros como valor, que correspondería a todos los pixeles de color blanco.

Redis a pesar de ser una base de datos en memoria, cuenta con una opción de persistencia llamada RDB[3], que básicamente vuelca cada 5 minutos en un fichero toda la base de datos, para que en caso de fallo o reinicio del sistema, solo tenga que leer en fichero para cargar toda la información. Esto causa que en caso de fallo del servicio Redis, se pierdan los pixeles colocados en los últimos 5 minutos. Considero esto más que aceptable.

---

[2] The command treats a Redis string as a array of bits, and is capable of addressing specific integer fields of varying bit widths and arbitrary non (necessary) aligned offset. https://redis.io/commands/bitfield

[3] The RDB persistence performs point-in-time snapshots of your dataset at specified intervals. https://redis.io/topics/persistence

3.2 API REST

La API REST consta de los siguientes endpoints:

POST /pixel

Es el endpoint mas importante de PixelCanvas. Se llama cuando el usuario quiere emplazar un pixel.

Se envían las coordenadas del pixel (x, y) que se desea emplazar, junto con un el código numérico del color. Se envía también un *fingerprint* del navegador como ayuda para identificar al usuario. Además del token cuando se pide un CAPTCHA. Y por ultimo, el valor 'a' que se computa a partir del resto de datos de la petición y cuya formula se modifica según la versión, como medida de protección contra los *bots*.

Retorna el cooldown que se le ha aplicado al usuario o un mensaje de error cuando no se ha podido emplazar el pixel.

Ejemplo de petición:

```
{
  “x":-965,
  “y":-520,
  “color":5,
  “fingerprint":"78091240c9bfc2156b28bf823157149e",
  “token”:null,
  “a”:-1477
}
```

Ejemplo de respuesta:

```
{
   “success”:true,
   “waitSeconds":95.766
}
```

GET /bigchunk/:x(-?[0-9]+).:y(-?[0-9]+).bmp

Este endpoint retorna el estado del mapa inicial cuando el usuario carga la pagina por primera vez, o cuando se desplaza por este. Retorna una cuadricula de Chunks centrada el en el chunk (x,y) directamente desde *Redis*.

Tiene una extension *.bmp* y un contentType de *image/bmp* para que Cloudflare cachee la la respuesta en su *CDN,* pero no nos supone un problema ya que en el cliente simplemente leemos el resultado como un binario.

Tiene en la cabecera Content-Encoding: `gzip` para que se comprima la respuesta. El tamaño medio de respuesta oscila los 100 KB, mientras que si hacemos los cálculos de cuanto mediría si no lo comprimiéramos nos da:

(15 x 15) Chunks x (64 x 64) pixels/Chunk x 4 bits/pixel

= 3 400 640 bits

= 425 KB

Por lo que logramos reducir el tamaño de la respuesta en un 75%.

La duración de la cache es de 5 segundos.

No se adjunta ejemplo de respuesta porque es binaria.

GET /me

Retorna el cooldown del usuario, i.e., la cantidad de segundos que tiene que esperar el usuario para volver a colocar un pixel. Se llama cuando se carga la pagina. No se cachea porque contiene información única del usuario. También retorna el *id* del usuario, junto con otros datos que son forman parte ya del *legacy*.

Ejemplo de respuesta:

```
{
   “id":"ip:161.116.133.39",
   “name":"Anonymous",
   “center”:[0,2000],
   “waitSeconds":145.553
}
```

GET /ws

Devuelve la *URL* de un servidor WebSocket, al que se conectará el cliente para obtener el stream de todos los pixeles que son colocados en la pagina y mantener actualizado el estado del canvas.

Cada servidor WebSocket expone una pequeña API donde sirve el numero de clientes que tiene conectado. Así el servidor web que le toque servir este endpoin, solo tiene que hacer peticiones periódicas a cada servidor WebSocket para conocer si sigue estando disponible y guardar en una lista en memoria el numero de conexiones que tiene.

La lógica que sigue es muy sencilla: de entre los servidores disponibles se devuelve el que tenga menos conexiones, o si es imposible hacer este calculo (por ejemplo, sin o de disponen de datos suficientemente actualizados), se selecciona uno aleatoriamente. De esta manera es como se balancea la carga entre los diferentes servidores WebSocket.

Se cachea durante 30 segundos en CloudFlare.

Ejemplo de respuesta:

```
{
   "url":"ws://138.68.96.54:8080"
}
```

GET /online

Se llama periódicamente desde el cliente.

Selecciona de la base de datos MySQL la cantidad de usuarios que han emplazado pixel en los últimos 5 minutos. Devuelve este valor como la cantidad de usuarios online.

Uno podría obtener la cantidad usuarios online sin necesidad de consultar la base de datos, simplemente sumando la cantidad de conexiones a los servidores WebSocket. Pero esto ya se ha probado de hacer y devuelve una cifra demasiado alta, ya que entre las conexiones a los servidores WebSocket se encuentra muchos clientes en estado idle, de por ejemplo usuarios que han dejado el navegador abierto.

Se cachea durante 30 segundos en CloudFlare.

Ejemplo de respueta:

```
{
   “online”:40
}
```

3.3 WebSockets

WebSocket es un protocolo que permite establecer una comunicación **full-duplex**[4] sobre **TCP** y esta soportado por la mayoría de navegadores en la actualidad[5]. Su uso es excelente para enviar actualizaciones del servidor hacia el cliente, hecho que antes de la implantación de esta tecnología requería de métodos rudimentarios como el *Long Polling*[6].

En PixelCanvas se usa para enviar al cliente los pixeles que son modificados en tiempo real.

Al abrir el navegador, el cliente se conecta a un servidor WebSocket donde recibirá mensajes binarios de 6 Bytes que contendrán toda la información necesaria para actualizar un pixel. Veamos como como estos 48 bits codifican una actualización:

Los primeros 16 bits indican la coordenada x del chunk.

Los siguientes 16 bits indican la coordenada y del chunk.

Los 16 bits del final determinan el offset (posición dentro del chunk) y el color del pixel:

Los primeros 12 bits codifican el offset dentro del chunk.

Los 4 últimos codifican el color del pixel tal y como esta especificado en el apartado de Redis.

Ahora, un servidor WebSocket seria insuficiente pues el tipo de servidores usados solo son capaces de mantener unas 1.000 conexiones abiertas, insuficiente para la escala de PixelCanvas, y aunque fuesen suficientes, tendríamos un *single Point of Failure*. Debemos tener más y tenerlos sincronizados.

Los múltiples servidores WebSocket son usados como un servicio a través de los servidores de la API gracias a **RabbitMQ**[7]. De manera que cuando un pixel es colocado, para notificar a todos los servidores WebSocket solo hace falta efectuar el siguiente comando:

```
broadcast(PixelUpdate.dehydrate({ x, y, color }));
```

---

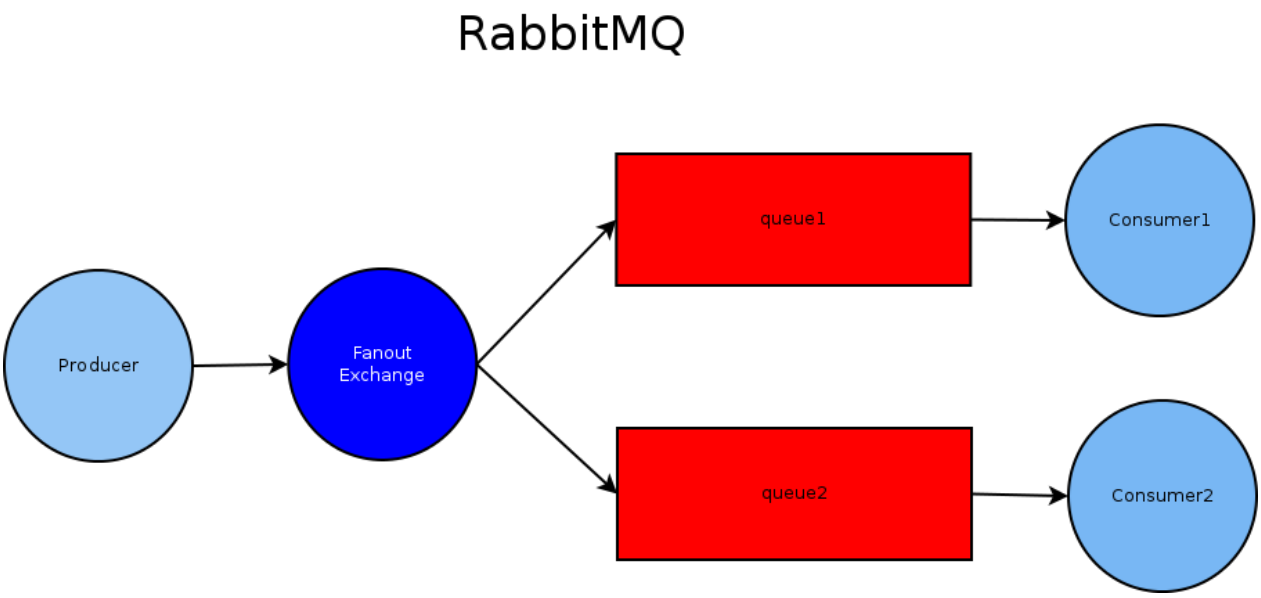
[4] Que permite el envío y recepción de datos simultáneamente.

[5] Actualmente el 94,42% de los usuarios tiene un navegador que soporta WebSocket https://caniuse.com/#search=websocket

[6] With long polling, the client requests information from the server exactly as in normal polling, but with the expectation the server may not respond immediately. If the server has no new information for the client when the poll is received, instead of sending an empty response, the server holds the request open and waits for response information to become available. Once it does have new information, the server immediately sends an HTTP/S response to the client, completing the open HTTP/S Request. Upon receipt of the server response, the client often immediately issues another server request. In this way the usual response latency (the time between when the information first becomes available and the next client request) otherwise associated with polling clients is eliminated.
https://en.wikipedia.org/wiki/Push_technology#Long_polling

[7] https://www.rabbitmq.com/

Esto se consigue creando un canal en modo **FANOUT** de manera que cuando un *publisher*[8] publica un mensaje en este canal, RabbitMQ se encarga de enviarlo a todos los los *consumer*[9].



Esquema del modo FANOUT

En este proyecto, los servidores API actúan de *publisher* y los servidores web-socket actúan de *consumers*.

Quiero hacer mención especial de resiliencia para RabbitMQ ya que durante mas del año que lleva en marcha el servidor donde esta RabbitMQ instalado no ha dado ningún problema ni jamás se ha caído.

[8] Un publisher es aquel que publica mensajes en la cola

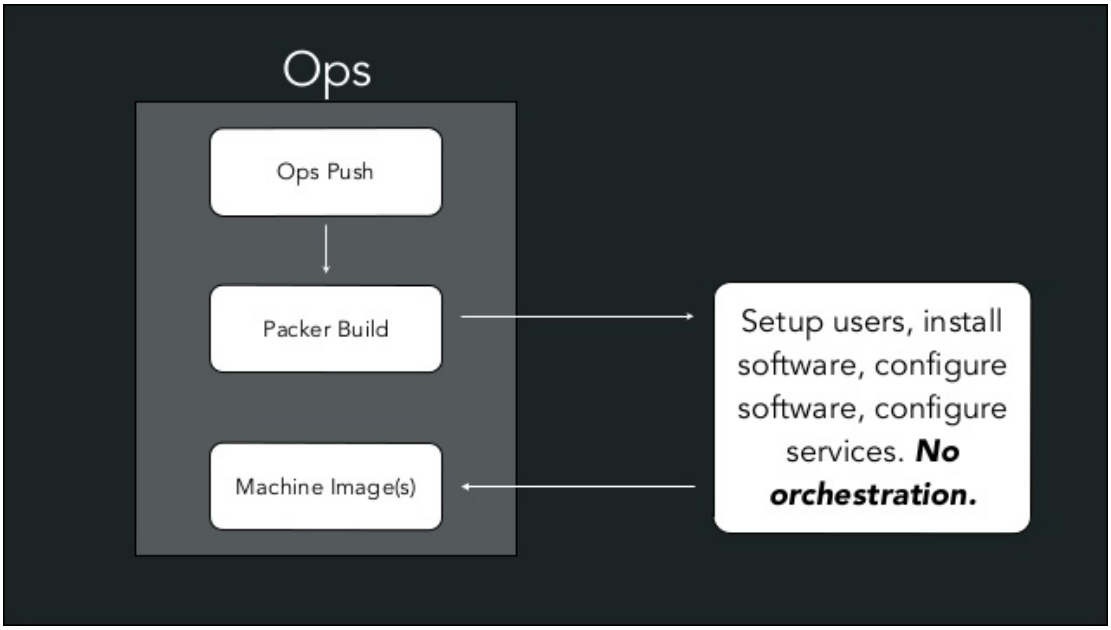
[9] Un consumer es aquel que se subscribe a los mensajes en la cola

3.4 Packer y Terraform

Packer[10] y Terraform[11] son las herramientas usadas para mantener, crear y escalar la infraestructura que soporta PixelCanvas en DigitalOcean[12].

Con Packer, se crean las imágenes ISO de los servidores web y WebSocket. Su funcionamiento es bastante básico pero muy útil. Cada vez que se ejecuta Packer, se lanza una nuevo Droplet[13] y se ejecutan los scripts bash especificados en los archivos de configuración de Packer. En estos scripts hay instrucciones que van a instalar las dependencias del proyecto, como Nginx, node, el código del proyecto optimizado para producción, etc. Una vez estos scripts finalizan, se crea una imagen con el nombre “pixelcanvas-fecha” que estará disponible en DigitalOcean y se destruye el Droplet.

De este forma, conseguimos tener controlado bajo un numero de versión todas las modificaciones que se realizan a nuestros servidores y simplificamos la creación de nuevos servidores idénticos, lo que vendrá muy bien para escalar.



Esquema del uso de Packer en el deployment

---

[10] Packer automates the creation of any type of machine image. It embraces modern configuration management by encouraging you to use automated scripts to install and configure the software within your Packer-made images. Packer brings machine images into the modern age, unlocking untapped potential and opening new opportunities.
https://www.packer.io/

[11] Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned.
https://www.terraform.io/

[12] Providing developers a reliable, easy-to-use cloud computing platform of virtual servers (Droplets).
https://www.digitalocean.com/

[13] Droplet es un servidor en DigitalOcean

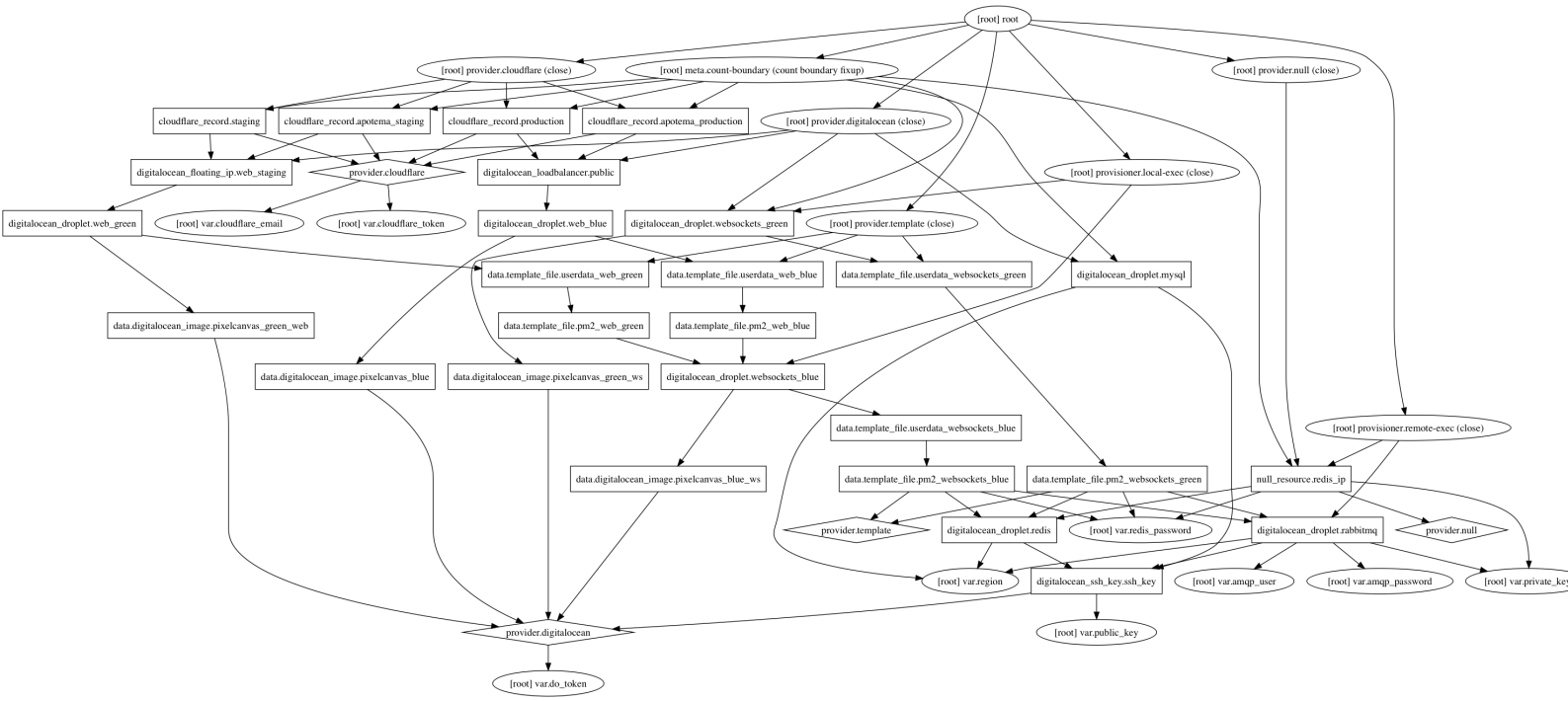
Ahora entra en juego Terraform. Terraform se encarga de ejecutar todos los pasos para transformar la infraestructura que tengas a la especificada en los archivos de configuración. De esta forma, tenemos la infraestructura del proyecto escrita junto al código y mantenida bajo el mismo sistema de versiones.

En estos archivos, especificamos como queremos crear los servidores web, WebSocket, Redis, RabbitMQ, MySQL, balanceado de carga y como configuramos los DNS a partir de estos recursos.

Además, se ha montado un esquema de deployment blue/green. Consiste en que cuando se quiere hacer un deploy de una nueva versión del proyecto, se crea una copia (grupo GREEN) de la infraestructura actual (grupo BLUE) con el nuevo código. Se apunta el DNS de staging.pixelcanvas.io a un servidor web del grupo blue, se comprueba que todo funcione bien.

Si funciona bien, se hace el switch: el DNS de staging.pixelcanvas.io pasa a apuntar al grupo GREEN y pixelcanvas.io pasa a apuntar al grupo BLUE con 0 downtime. Si después del cambio si que se detectan errores que habían pasado desapercibidos en staging, se puede volver a aplicar el switch y volver al estado original con 0 coste.

Una vez pasado un tiempo prudencial, se puede proceder a destruir el grupo de staging para ahorrar costes.



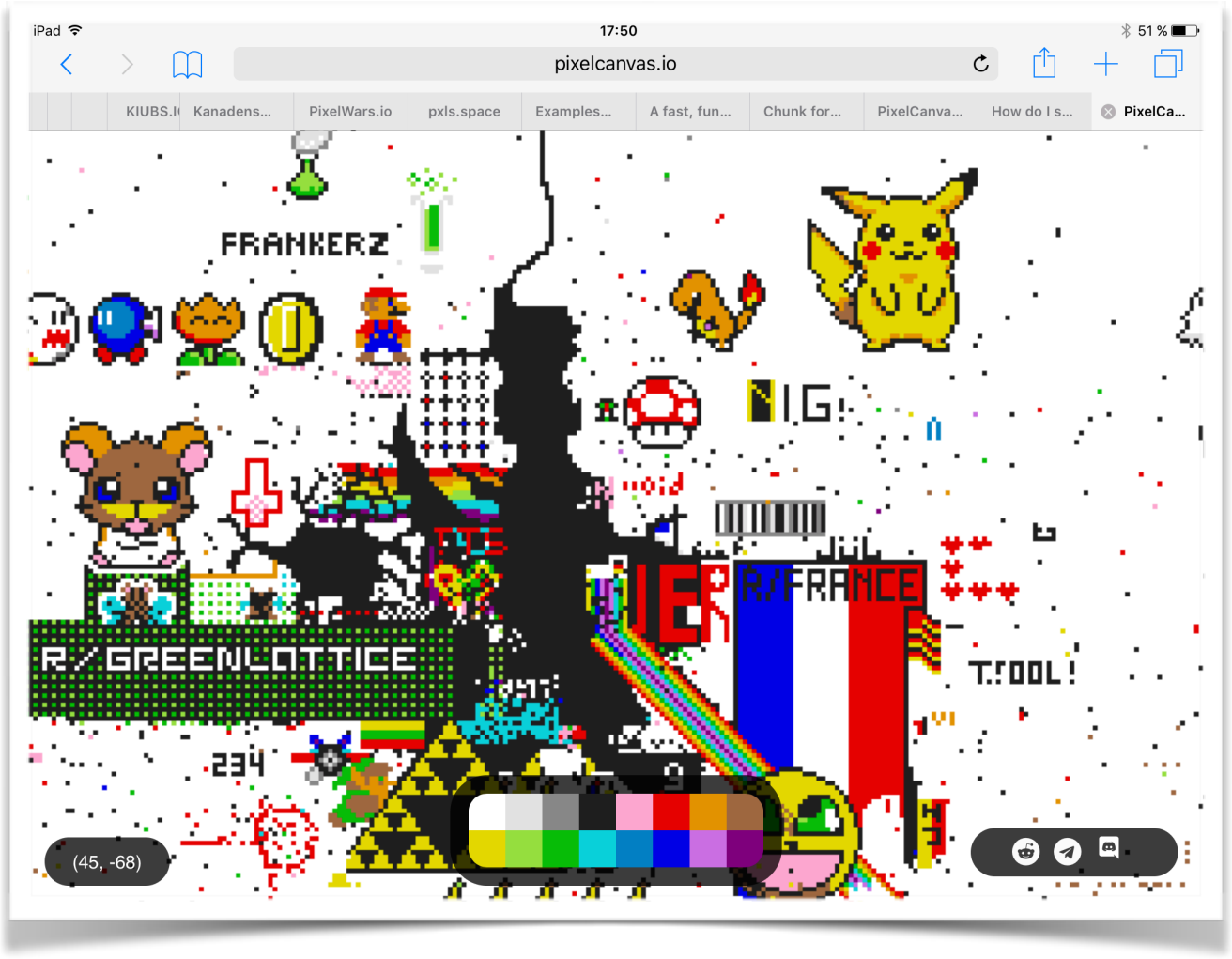
Grafo generado por terraform con $ terraform graph | dot -Tpng > graph.png

4. Resultado

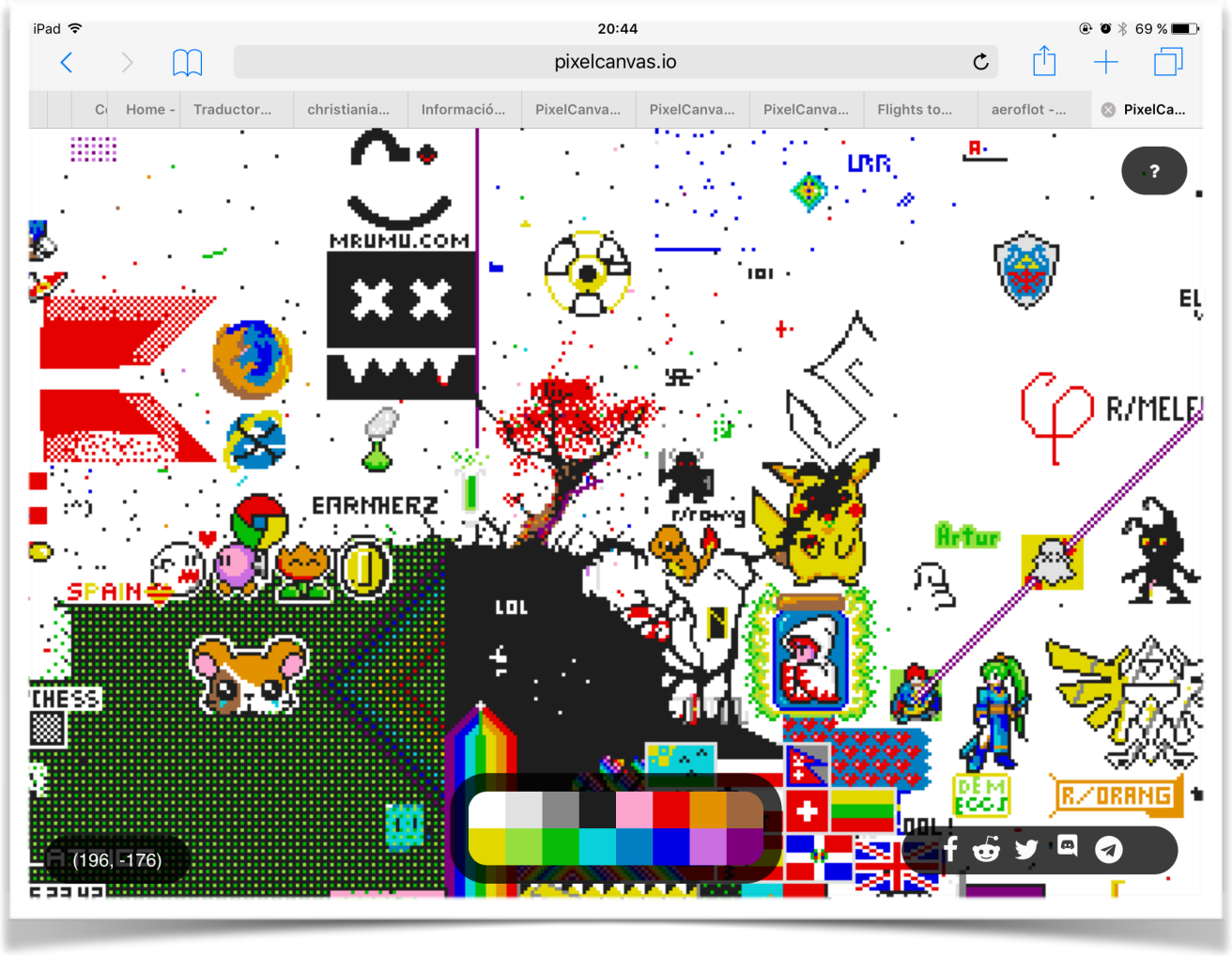
PixelCanvas.io se lanzó como aplicación web el 8 de abril de 2017.

Desde ese día y durante las próximas dos semanas se procedió a atraer usuarios activos de otras plataformas web y foros populares mediante mensajes personales a sus cuentas informándoles del nacimiento de este juego de colaboración - no sería erróneo decir que fueron mensajes de spam- o en el caso de los foros mediante posts y comentarios en posts gaming ajenos.

Inmediatamente la gente comenzó a investigar la web y se creó un colectivo de unos 60 usuarios activos diarios fieles al juego -es decir que se conectaban a diario.



Captura del centro del lienzo del día 12 de Abril de 2017 a las 17:50.



Captura del centro del lienzo del día 17 de Abril de 2017 a las 17:50.

Pronto ese número escaló a unos 100 jugadores, y surgió la necesidad de crear algún sitio de contacto entre los jugadores, puesto que les interesaba ponerse de acuerdo en que “dibujo” hacer a continuación como grupo.

Para satisfacer esa demanda, el 14 de abril se creó el canal de Discord de Pixelcanvas, conmigo como único administrador, y el 16 de abril se creó la página de Facebook de este, con un amigo que había estado desde el inicio del desarrollo de la aplicación como consejero.
La creación de estos dos canales de comunicación fue un acierto, pues no solo permitió satisfacer la demanda de los jugadores, sino que estos hicieron difusión de la web - mediante el botón de share de Facebook en publicaciones y agregando gente al canal de Discord- lo que permitió que el número de usuarios creciera casi exponencialmente esos días.

Pixelcanvas se había consolidado como juego en plataforma web, aunque comparado con juegos de éxito, su número de jugadores fieles era aún pequeño; debía de tener en torno a los 300 jugadores cuando surgió un evento que lo cambió para siempre: los usuarios del foro Forocoches se interesaron por la plataforma.

Un grupo de usuarios del famoso foro español decidió construir la bandera de Forocoches en el centro de coordenadas del Pixelcanvas, y alargarlo en el tablero tanto como quisieran.

Esa acción hirió e hizo marchar a muchos de los usuarios permanentes que tenia la web, pues para la construcción de la bandera destruyeron muchos de los pictogramas que los usuarios antiguos habían construido - hasta ahora al ser el lienzo infinito, se había respetado el trabajo de todos y nunca destruido, aunque si modificado a mejor. Aún así, y aunque la moderación en el canal de Discord fue difícil esos días, para Pixelcanvas, como plataforma, fue muy beneficioso, pues se dio a conocer a un grupo muy numeroso de gente potencialmente activa en el juego.

Captura del centro del lienzo el 1 de mayo de 2017 a las 18:11

La situación siguió así durante un periodo de tiempo prolongado, donde el numero de usuarios -aproximadamente 600- se mantuvo estable y la actividad en la web era visible.

El 12 de mayo surgió otro evento que propulsó Pixelcanvas: Legión Holk, un foro de Sudamerica, decidió entrar en la web con la máxima cantidad de usuarios activos de la que pudieran disponer y dibujar su bandera sobre la de ForoCoches - sobreescribiendo esta y los dibujos que se habían anexionado a esta. Durante una semana se creó una “guerra de guerrillas” diaria aprovechando el cambio horario de los dos países; así durante la mañana, en sentido español, ForoCoches impulsaba a sus usuarios a “reparar” su bandera, mientras que por las noches, Legión Holk, instaba a todos sus activos a mantener su bandera y terminar de “destruir” la de ForoCoches.

Durante esos días, un fluir constante de personas paso por la web, cosa que provoco que los servidores cayeran en mas de una ocasión - pues no se habían previsto para tanta cantidad de gente - aunque pronto cada error era solventado.

Fue entonces cuando se empezaron a crear timelapses de usuarios que grababan el conflicto interno y a colgarlos en youtube; por lo que el 10 de junio Pixelcanvas creó una cuenta en YouTube usada expresamente para añadir timelapses suyos y de usuarios que se los mandaban con permiso de publicación.

Al final Legión Holk gano la “batalla” y se situó en el centro de coordenadas, mientras que ForoCoches buscó una situación un poco inferior para su bandera. Así termino el conflicto y la cantidad de usuarios bajo, aunque se mantuvo muy superior a la de antes del inicio del conflicto.

Los usuarios siguieron difundiendo la web de forma masiva y participando en el juego, lo que propició que llegara a oídos del foro Seguidores de la Grasa - un foro sudamericano poco conocido en España pero de gran repercusión y polémica, por alguno de sus posts, en América. Resultado de una rivalidad ya anterior con Legión Holk, estos procedieron a un “ataque” a la bandera de los segundos para sustituirla por la suya; un acto parecido a la primera “guerra de guerrillas” de Legión Holk con ForoCoches pero a gran escala, pues Seguidores de la Grasa consta con más de 500.000 usuarios activos.

Para Pixelcanvas esa entrada masiva de gente le proporciono una gran publicidad, y escaló rápidamente al juego basado en dibujar pixeles con más trafico diario, llegando a tener mas de 1.000.000 de visitantes únicos al día.

Después de este “ataque” que duró aproximadamente un mes y medio - donde los momentos mas intensos fueron durante las dos primeras semanas, cuando comparecía más gente - Pixelcanvas había llegado a un estado pico, la cantidad de gente era abrumadora y la participación y mención en foros sobrecogía.

Se necesitaron dos meses mas para que la cantidad de gente descendiera otra vez notablemente, pero la participación siguió siendo alta.

Después de eso, aunque PixelCanvas sufrió picos y caídas, se mantuvo estable -con un ligero decrecimiento - donde lo más relevante fue la iniciación de la participación de 4chan - el foro más importante de habla inglesa - en PixelCanvas y la entrada de muchos grupos de usuarios de pequeñas comunidades que deseaban dejar su marca en la web.

Probablemente esa fue la mejor época de PixelCanvas, pues los meses que se sobrevinieron resultaron ser muy caóticos.

El incremento de usuarios hizo de tema principal encontrar el modo de saltarse la limitación de tiempo interpuesta a la hora de “pintar el lienzo” que proporcionaba pixelcanvas y aprovechar al máximo el tiempo disponible para pintar. Por eso, un grupo notable de usuarios se centraron en desarrollar bots que actuaran por ellos en la creación de los pictogramas. Eso creo que usuarios fieles a la web se enfadaran y exigieran soluciones rápidas, pues veían “destruir su arte” en segundos y sin esfuerzo. Pixelcanvas trabajó en ello y redujo la cantidad notablemente.

Hubo solamente tres ocasiones que se vivieron difíciles para mantener el orden en el juego:

La creación de bots que mediante una cantidad elevada de IP’s “protegían” ciertos pictogramas. Eso implicaba que si tu tratabas de modificarlo inmediatamente fuera corregido y restaurado a la imagen principal por el bot. En eso Pixelcanvas, después de mucho trabajar en ello, cedió, y se le permitió la existencia.

4chan explotó un error del programa[14], mediante el cual permitía colocar pixeles sin tiempo de espera. Si se modificaban adecuadamente las cabeceras HTTP, se podía hacer creer al servidor web que la petición se originaba desde una IP falsa arbitraria.

[14] https://www.reddit.com/r/PixelCanvas/comments/6f6hn1/exploit_fixed/

Después de solventar estos incidentes, la cantidad de usuarios activos, aun siendo menor que el pico, se mantuvo estable y se creó una comunidad de usuarios activos y fieles.

5. Conclusiones y Trabajo futuro

PixelCanvas fue un éxito porque supo aprovechar el momento donde los usuarios se mostraron interesados en este tipo de juegos y le aportó un matiz distinto de los demás que lo hacia único (lienzo infinito…) y mas atractivo a los jugadores.

La mayor parte, fue debido a asuntos que escapaban a su control: como por ejemplo los *raids* entre facciones. El éxito recae en saber manejar la situación y saber gestionar los recursos disponibles así como la capacidad técnica, para saber aprovechar la situación.

Me ha aportado muchos conocimientos en el ámbito de sistemas distribuidos y me ha dado una experiencia manejando un entorno real de producción con una cantidad masiva de usuarios que muy poca gente tiene. Conozco de primera mano todas las facetas que hay detrás de un servicio web a gran escala.

Una de las tareas pendientes en PixelCanvas sigue siendo la lucha contra los bots y proxies. Como se ha explicado anteriormente, todos los métodos usados anteriormente son insuficientes y lo que cabe esperar es que no se hallará una solución definitiva a este problema, pues todas las webs ampliamente usadas (Twitter, Instagram, Reddit…) sufren de los bots.

Por tanto, la solución tiene que ir encaminada a mejorar el sistema de bloqueo de usuarios. Actualmente, cuando un usuario es **bloqueado** del *canvas*, todos los pixeles colocados por este usuario vuelven a ser **pixeles virgenes**, es decir, se tornan de color blanco y su *cooldown* baja al mínimo. Este comportamiento hacia los usuarios que realizan trampas elimina el incentivo que tienen para crear arte, pues saben que tarde o temprano serán detectados y su arte eliminada. Pero es insuficiente, pues aun-

que se desincentiva la creación de arte con métodos tramposos, persiste la incentivación de destrucción, dado que el autor legitimo del pixel anterior a este también ve su creación borrada. Lo ideal seria por tanto, no solo eliminar el pixel tramposo, sino **restaurar el pixel** hacia un estado donde su autoría sea legitima.

En la actual versión, el listado de pixeles emplazados por un mismo usuario se obtiene a través de una base de datos **MySQL**, donde por cada coordenada ***(x, y)*** se guardan varios metadatos, entre ellos el **autor**. Y como actualmente este es el único propósito de la base de datos MySQL en PixelCanvas, junto con que es el mayor **cuello de botella** en la infraestructura, su provee su eliminación.

Lo natural a primera vista, seria extender esta base de datos SQL para que dado unas coordenadas retorne el histórico de pixeles: **(x, y) -> [pixels]** no resulta trivial, amen de las modificaciones pertinentes en el código para lograr mantener actualizado este histórico no son pocas.

Por eso el plan es prescindir de la base de datos MySQL y crear un **Distributred Ledger**. Porque mantener continuamente en el estado de la aplicación esta estructura de lista? Y porque no, simplemente guardar la cantidad esencial mínima de datos de la que se pueda replicar todo el estado de la aplicación? Esta es básicamente, la idea de un Distibuted Ledger. Algo parecido así como el *blockchain* es al BitCoin, donde se guarda el histórico de transacciones y a partir de este de reconstruye el balance de cada cuenta. En nuestro caso, en PixelCanvas este Distributed Ledger o log centralizado consistiría simplemente en un log donde se almacenarían los pixeles emplazados, con sus metadatos (autor, fecha, etc).

El Distributed Ledger tiene una serie de propiedades que lo hacen muy interesante como **estructura de datos** y **único source of truth:** Es **immutable**, solo se le añaden entradas al final, y se consume **secuencialmente**. Esto resulta una idea muy potente a priori, ya que no solo nos permite reconstruir el histórico de pixeles dadas unas coordenadas [ver esquema], sino que nos permite hacerlo para pixeles emplazados **antes de implementar esta funcionalidad** (la restauración de pixeles)**,** algo que resultaría imposible con el *approach* del MySQL. Y abre un amplio abanico de posibilidades al desarrollo de nuevas funcionalidades:

Como la **generación** de *timelapses* en cualquier zona del mapa y cualquier franja de tiempo desde el inicio del juego.

Añadir nuevos colores a la paleta. Contando con la estructura actual del Redis, no seria una tarea fácil de llevar a cabo, ya que habría que intercalar una cantidad adecuada de 0 cada 4 bits y nos desmontaría todas las optimizaciones bit-wise que tenemos. Con el log centralizado, simplemente borraríamos el Redis (ya no es *source of truth*), y iteraríamos por las entradas del stream, poblando el canvas en redis ya con el nuevo codigo. Esto en el caso de Pixelcanvas, en general nos podemos ahorrar el engorro de realizar **migraciones**.

Restaurar el estado del canvas a un punto arbitrario del tiempo, sin necesidad de tener un backup de esa fecha.

En definitiva, quiero hacer hincapié en que a parte se simplificar mucho la estructura de datos utilizada, de desacoplar mas los diferentes *microservicios*, nos permite aumentar el valor potencial que podemos generar a partir de nuestros datos. Por estas razones, me parece que es la siguiente mejora que se debe aplicar al PixelCanvas y que voy a estudiar de aplicar en otros proyectos porque el Distributed Ledger es una idea muy potente que promete mucho.

Bibliografía

Node.js v8.11.3 Documentation [en línea] 2018 [consulta: 27 de junio de 2018].
Disponible en https://nodejs.org/dist/latest-v8.x/docs/api/

Advanced logging with NodeJs [en línea] 2018 [consulta: 27 de junio de 2018].
Disponible en http://tostring.it/2014/06/23/advanced-logging-with-nodejs/

ExpressJS API Reference [en línea] 2018 [consulta: 27 de junio de 2018].
Disponible en http://expressjs.com/es/4x/api.html

Mitchell Hashimoto, HashiCorp [en línea] 2017 [consulta: 3 de mayo de 2017].
Disponible en https://www.slideshare.net/profyclub_ru/8-mitchell-hashimoto-hashicorp

Terraform Documentation [en línea] 2017 [consulta: 10 de mayo de 2017].
Disponible en https://www.terraform.io/docs/index.html

Packer Documentation [en línea] 2017 [consulta: 10 de mayo de 2017].
Disponible en https://www.packer.io/docs/index.html

Load Balancers on DigitalOcean [en línea] 2017 [consulta: 10 de mayo de 2017].
Disponible en https://www.digitalocean.com/products/load-balancer/

Mitchell Hashimoto, HashiCorp [en línea] 2017 [consulta: 3 de mayo de 2017].
Disponible en https://www.slideshare.net/profyclub_ru/8-mitchell-hashimoto-hashicorp

Redis in Action [en línea] 2017 [consulta: 15 de junio de 2018].
Disponible en https://redislabs.com/ebook/foreword/

WebSockets - Mozilla Developer Network [en línea] 2017 [consulta: 2 de abril de 2017].
Disponible en https://developer.mozilla.org/es/docs/WebSockets-840092-dup

RabbitmMQ Documentation [en línea] 2017 [consulta: 10 de mayo de 2017].
Disponible en https://www.rabbitmq.com/documentation.html

Getting started with RabbitMQ and Node.js [en línea] 2017 [consulta: 10 de mayo de 2017].
Disponible en https://www.cloudamqp.com/blog/2015-05-19-part2-2-rabbitmq-for-beginners_example-and-sample-code-node-js.html

IP Intelligence - Proxy / VPN / Bad IP Detection [en línea] 2018 [consulta: 27 de junio de 2018].
Disponible en http://getipintel.net

reCAPTCHA: Easy on Humands, Hard on Bots - Google [en línea] 2018 [consulta: 27 de junio de 2018].
Disponible en https://www.google.com/recaptcha/intro/v3beta.html

React Documentation [en línea] 2017 [consulta: 1 de abril de 2017].
Disponible en https://reactjs.org/docs/hello-world.html

React Starter Kit [en línea] 2017 [consulta: 1 de abril de 2017].
Disponible en https://reactstarter.com

Redux Documentation [en línea] 2017 [consulta: 1 de abril de 2017].
Disponible en https://es.redux.js.org/

Amazon Kinesis: the best event queue you're not using [en línea] 2018 [consulta: 22 de junio de 2018].
Disponible en https://instrumentalapp.com/blog/aws-kinesis/

Apache Kafka® Delivers a Single Source of Truth for The New York Times [en línea] 2018 [consulta: 22 de junio de 2018].
Disponible en https://www.slideshare.net/ConfluentInc/apache-kafka-delivers-a-single-source-of-truth-for-the-new-york-times

Anexo

El canal de YouTube de PixelCanvas es https://www.youtube.com/channel/UCMBFg9BHa7Bw9b_iODudv2w y en el están publicados varios timelapses que pueden resultar de interés.

Todo el código está hospedado en https://github.com/pixelcanvasio/pixelcanvas desde que se liberó bajo licencia GPLv3 el 3 de septiembre de 2017.

Para poder ejecutar en local el proyecto, es necesario tener instaladas las dependencias:

Node: https://nodejs.org/es/download/package-manager/

Redis: https://redis.io/topics/quickstart

RabbitMQ: https://www.rabbitmq.com/download.html

MySQL o SQLite

pm2: Un process manager para node

```
$ npm install pm2 -g
```

Una vez satisfechas las dependencias, ejecutar en una carpeta vacía:

```
$ git clone https://github.com/pixelcanvasio/pixelcanvas.git .
$ npm install
$ npm run build
$ pm2 start environment.yml
```

Para abrir los logs:

```
$ pm2 logs
```