Informe de Trabajo Profesional

Análisis de ecosistemas para la implementación de plataformas como servicio para despliegue de aplicaciones comunitarias, distribuidas y descentralizadas

Integrantes

Lucas Nahuel Sotelo Guerreño
102730
lsotelo@fi.uba.ar

Sebastián Bento Inneo Veiga
100998
sinneo@fi.uba.ar

Joaquín Matías Velazquez
105980
jvelazquez@fi.uba.ar

Joaquín Prada
105978
jprada@fi.uba.ar

Tutor
Ariel Scarpinelli
ascarpinelli@fi.uba.ar

Índice

1. Resumen

En el siguiente trabajo se analizan distintos ecosistemas y tecnologías que se pueden utilizar para el despliegue de aplicaciones web comunitarias de manera distribuida y descentralizada.

Mediante tres casos de uso que ilustran diferentes características: un sitio web informativo, un repositorio de conocimiento y un mensajero en tiempo real, se comparan ventajas y desventajas del despliegue de cada uno de ellos en IPFS, blockchain y Hyphanet/Freenet, así como también se documenta el proceso del mismo.

2. Palabras Clave

Distribuido. Sistema. Comunitario. Descentralizado. Aplicación.

3. Abstract

The following work analyzes different ecosystems and technologies that can be used to deploy community web applications in a distributed and decentralized manner.

Using three use cases that illustrate different features: an informational website, a knowledge repository and a real-time messenger, the advantages and disadvantages of deploying each of them on IPFS, blockchain and Hyphanet/Freenet are compared, as well as the process to do so is documented.

4. Keywords

Distributed. System. Community. Decentralized. Application.

5. Introducción

Hoy en día, al querer desplegar una aplicación o sitio web comunitario, lo más común es hacerlo a través de un servicio de alojamiento (AWS, Azure, Google Cloud, entre otros) por la comodidad y facilidad que estas ofrecen, alquilando sus servidores para guardar y procesar datos.

Esto puede llegar a traer problemas para este tipo de aplicaciones. Uno de estos problemas puede ser monetario, ya que muchas veces estas aplicaciones dependen de donaciones o voluntarios para sustentarse, como es el caso de Wikipedia. Como también puede suceder que se encuentre en una zona de censura, lo cual facilita su bloqueo al ser servicios centralizados; entre otros problemas más.

Sin embargo, existen otros ecosistemas alternativos que se asemejan mucho más a la filosofía de estas aplicaciones, y que ayudan a combatir estos problemas. En donde las aplicaciones pueden estar alojadas por sus propios usuarios, donando su computo o espacio, y así logrando una descentralización.

En el siguiente documento presentamos un análisis sobre la infraestructura existente, donde es posible la implementación de plataformas para el despliegue de este tipo de aplicaciones, recabando las bondades y desventajas que cada una tiene.

Para esto se crearon diferentes casos de uso que representan posibles aplicaciones sobre esta metodología alternativa analizando su viabilidad. Entre ellos, se encuentran un sitio web estático, una enciclopedia colaborativa y una aplicación de comunicación en tiempo real.

6. Estado del Arte

En esta sección describiremos en qué se diferencian las aplicaciones descentralizadas de aquellas centralizadas, cuáles son las ventajas (y desventajas) del modelo de aplicación distribuido, y qué tecnologías existen actualmente para asistir en la creación de dichas aplicaciones.

6.1. Introducción a arquitecturas de red

Comunicar distintas computadoras es un trabajo que requiere coordinación por parte de todas las partes, protocolos para estandarizar la información que se transmite, e infraestructura para poder envíar cada bit de origen a destino. Entonces, se debe diseñar una red coordinada para poder ofrecer los distintos servicios a través de Internet. Para ello, existen dos arquitecturas principales.

Cliente-Servidor

Presente en la gran mayoría las aplicaciones de Internet, el modelo *Cliente-Servidor* consiste en mantener un nodo central (servidor), quién se encarga de manejar la interacción entre los demás nodos (clientes), y entre clientes y el mismo servidor. Este modelo se clasifica como **centralizado**, debido a que la subred de sistemas depende del nodo servidor, y los clientes no tienen manera de comunicarse sin él ante una eventual caída del servidor.

Entre los servicios de Internet más utilizados que utilizan esta arquitectura se encuentra la World Wide Web, el servicio de e-mail (SMTP), el servicio de DNS, entre otros.

Peer-to-Peer

El modelo **peer-to-peer(P2P)** consiste en una red **descentralizada** que tiene distintos nodos (pares) capaces de comunicarse sin necesidad de un nodo central, por lo que se puede considerar que cada nodo cumple la función tanto de servidor como de cliente a la vez.

BitTorrent El servicio más utilizado que implementa este modelo es la red de BitTorrent, que implementa el protocolo del mismo nombre para compartir archivos entre pares. Esta red logra que el mismo nodo que descarga un contenido de la red sea a la vez el servidor para otro nodo que quiera acceder a ese contenido.

6.2. Diferencias y ventajas de cada arquitectura

Ambos modelos tienen ventajas y desventajas, y por lo tanto distintos casos de uso. El modelo cliente-servidor actualmente es la arquitectura mas utilizada,

Resiliencia Cuando un servidor se encuentra fuera de servicio, toda la red que depende de él no funcionará en tanto no se restaure el servidor. Para contrarrestar esta vulnerabilidad del modelo, se desarrollaron métodos a lo largo de los años. Una manera de evitar que la red se vuelva inoperativa es la de alojar diferentes instancias del servidor en diferentes zonas geográficas. Además, se pueden implementar medidas para evitar la caída del servidor, como los técnicas de balanceo de cargas y fuentes de energía alternativas para evitar eventuales cortes de electricidad.

No obstante, una red peer-to-peer puede ser incluso más robusta. Si hay suficientes pares en la misma y están lo suficientemente dispersos geográficamente, la desconexión de uno de ellos no desactiva toda la red. Esto permite que el modelo peer-to-peer pueda ser resistente a cortes de energía masivos y desastres naturales, lo que lo hace un modelo ideal para servicios prioritarios.

Cabe destacar que en una red de una cantidad limitada de pares, en donde no hay redundancia del contenido que se distribuye, es posible que al desconectar uno de los pares parte del contenido

se vuelva no disponible. Por lo tanto, si bien la red seguirá activa, no tendrá toda la funcionalidad que si puede ofrecer un servidor mientras siga en línea.

Escalabilidad La escalabilidad de una red peer-to-peer aumenta con la cantidad de nodos disponibles, dado que hay más recursos y, en una red bien diseñada, la carga se distribuye equitativamente. En un modelo cliente-servidor, garantizar escalabilidad se puede tornar costoso. Un servidor con mayor capacidad para comunicaciones entrantes y volumen de información tiene hardware de un costo mayor. En casos de aplicaciones de uso masivo puede ser necesario multiplicar la cantidad de nodos servidores para satisfacer la demanda de clientes.

Control del contenido Un servidor, al ser la pieza central de la red a la cuál pertenece, debe soportar múltiples conexiones en simultáneo. Para aplicaciones de alto tráfico, esto requiere una infraestructura que los usuarios suelen no poseer. Una solución es tercerizar el alojamiento de la aplicación servidor en plataformas de Cloud Hosting como pueden ser AWS, Azure, Google Cloud, entre otras. Estos servicios mantienen los servidores de numerosas aplicaciones de Internet, y por lo tanto, tienen la capacidad de modificar, censurar, o remover cualquiera de ellas si así lo desean.

Una red P2P no sufre de estos problemas, ya que por naturaleza los usuarios son quienes la alojan. Por lo tanto, remover contenido de ella resulta mucho mas complejo. Esto evita la censura en zonas donde el acceso a Internet es controlado y/o limitado, pero también puede incentivar a la distribución de contenido ilegal.

Seguridad La seguridad en las aplicaciones cliente-servidor se ha investigado por mucho más tiempo debido a la popularidad de este modelo. Además, al ser centralizado, el propietario del servidor puede bloquear conexiones y eliminar contenido malicioso de su plataforma de forma transparente para los usuarios.

El modelo descentralizado, en cambio, no cuenta con el desarrollo en términos de seguridad. En este caso, el cliente es el responsable de conectarse a redes de confianza, o de hacerlo mediante VPNs (Virtual Private Networks) para mantener el anonimato mientras se integre una red P2P. Sin embargo, cualquiera sea el modelo utilizado por una aplicación, la mayor parte de la seguridad dependerá de que tan segura sea la aplicación.

Persistencia Como varias otras propiedades del modelo cliente-servidor, depende de la integridad del servidor. Si el almacenamiento físico de este se ve afectado, los datos pueden perderse definitivamente. Por esta razón, es común tener un respaldo de los datos de la aplicación en otro disco u otro nodo para evitar la pérdida total de datos.

En una red descentralizada, la persistencia depende de la aplicación utilizada. En el caso de BitTorrent, cada nodo que se conecte y descargue un archivo, podrá compartir ese archivo con otros nodos, y por lo tanto ese archivo contará con una redundancia adicional, la cuál crece a medida que más personas descargan ese archivo. A pesar de esto, en casos donde el archivo es poco compartido, puede volverse inaccesible si los nodos que lo contienen se desconectan de la red.

Latencia Dada una conexión a Internet promedio, las velocidades manejadas por las aplicaciones cliente-servidor suelen ser aceptables. Sin embargo, en zonas en donde la conexión es escasa, o en casos en donde el servidor está lejos del cliente, la velocidad de transferencia de la aplicación puede verse afectada. Además, no es infrecuente encontrar cortes en videollamadas, juegos, y demás aplicaciones de tiempo real que siguen esta arquitectura. Como en la mayoría de defectos del modelo cliente-servidor, se puede solucionar agregando múltiples instancias del servidor. Por ejemplo, es común almacenar películas, videos y demás contenido de aplicaciones de streaming en distintos servidores de CDN (Content Delivery Network). Estas redes minimizan la distancia entre el usuario y el servidor, agilizando así la transferencia del contenido.

A pesar de los avances en la optimización del modelo cliente-servidor, las redes descentralizadas, cuando son eficientes y están bien pobladas, suelen ofrecer incluso mejores resultados. Esto se debe

a que la fuente de un contenido puede estar presente en múltiples nodos, lo que aumenta la probabilidad de que un nodo cercano tenga el contenido solicitado. La velocidad de transferencia que puede proporcionar un vecino con el contenido que requerimos generalmente superará la ofrecida por un servidor.

Costos Los costos de alojar una aplicación peer-to-peer suele ser nulo, ya que los mismos usuarios de ella son los encargados de proporcionar la infraestructura de la red.

Al contrario, alojar la aplicación en un servidor conlleva tener un servidor disponible, o bien contratar un servicio de web hosting, cuya tarifa suele aumentar a medida que la aplicación escala. En la mayoría de los casos, el costo termina siendo significativo, por lo que una aplicación descentralizada es una opción viable en escenarios en los que no se desee invertir mucho dinero.

6.3. Ambientes y herramientas

Existen varios ecosistemas que apuntan a proveer un marco con el cuál desarrollar una aplicación descentralizada. A su vez, cada uno de ellos cuenta con herramientas especializadas para los diferentes tipos de aplicaciones.

6.3.1. IPFS

Un suite modular de protocolos para organizar y transferir datos, diseñado con los principios de 'content addressing' (recuperación de archivos en base a contenido y no en base al nombre o id) y una red peer-to-peer. Su principal caso de uso es para publicar datos como archivos, directorios y páginas web descentralizadas.[11]

Fleek Plataforma centralizada para alojar servicios. También es utilizada para almacenar datos, y hacer accesible contenido para el resto de la web mediante gateways de IPFS.[4]

OrbitDB Base de datos peer-to-peer descentralizada. Cuenta con diferentes modelos de datos, incluyendo documentos, eventos, y diccionarios clave-valor. Utiliza IPFS para guardar y sincronizar automáticamente los datos. [18]

libp2p Colección de protocolos y utilidades para facilitar la conexión y comunicación entre pares en IPFS. Entre sus herramientas, se encuentran diferentes mecanismos de seguridad, de transporte, y para descubrimiento de pares. [17]

6.3.2. Blockchain

Tecnología basada en una cadena de bloques de operaciones descentralizada y pública. Esta tecnología genera una base de datos compartida a la que tienen acceso sus participantes, los cuáles pueden rastrear cada transacción que hayan realizado.[8]

6.3.3. Alternativas

Freenet/Hyphanet Programa de código abierto para compartir datos peer-to-peer con enfoque en la protección de la privacidad. Opera en una red descentralizada, promocionando la libertad de expresión facilitando la anonimidad de los datos compartidos y eludiendo la censura.[5][9]

7. Problema detectado y/o faltante

Los servicios actuales que proveen de infraestructura tienden a ser muy costosos para las pequeñas comunidades que necesitan tener un servicio con alta disponibilidad, accesible para todos y barato de escalar. Actualmente tampoco existe un estándar para aplicaciones cuyo stack sea completamente distribuido usando peer-to-peer.

7.1. Costos

La inversión para el mantenimiento de servidores puede ser un obstáculo a la hora de proveer una red a sus usuarios cuando se trata de una aplicación pequeña o startup. En estos casos, es común sacrificar la escalabilidad en pos de mantener los costos del alojamiento de la aplicación bajos.

En el otro extremo del espectro, se encuentran las aplicaciones en declive. Debido a la falta de incentivos financieros para justificar el mantenimiento, muchas veces la solución es desconectar los servidores definitivamente. Esto es especialmente frecuente en videojuegos multijugador que no cuentan con la capacidad de crear servidores dedicados. En tales situaciones, la empresa propietaria puede desactivar los servidores, lo que resulta en que el videojuego se vuelva parcialmente o completamente inutilizable. [1]

7.2. Interrupciones del servicio

Dada la naturaleza del modelo cliente-servidor, no es infrecuente que estas redes se vuelvan inaccesibles. Esto puede ocurrir deliberadamente por temas de mantenimiento, o accidentalmente debido a modificaciones en la aplicación del servidor durante el mantenimiento o actualización de la misma.

Uno de los episodios recientes de mayor revuelo ocurrió el 4 de Octubre de 2021, día en el que la familia de aplicaciones de Meta -Whatsapp, Facebook e Instagram -, estuvo fuera de servicio por seis horas. Esto ocasionó que mas de 3500 millones de usuarios se vieran afectados. En un articulo posterior, Meta reveló que la causa se debió a una falla en la herramienta de auditoría de los comandos que se envían a la red global de datacenters de Meta, la cual evitó que se pudiera detener el comando que desconectaba los servidores de la red. [15]

7.3. Zonas de censura

Es común que aplicaciones o sitios web comunitarios sean sometidos a su censura. La centralización de los servicios ayuda a que sea mucho más fácil bloquear su acceso, dado que es más fácil identificar y bloquear un único punto de acceso. En contraste, los sistemas descentralizados, suelen ser más resistentes porque no dependen de un servidor o servicio central que pueda ser intervenido fácilmente.

Uno de los casos más conocidos y vigentes es la censura de Wikipedia. Donde algunos gobiernos bloquean su acceso en un determinado lenguaje y otros en su totalidad. [2]

8. Solución implementada

Se realizó la implementación de tres casos de uso sobre los distintos ecosistemas a analizar. Presentamos un análisis cualitativo y cuantitativo de las ventajas y desventajas de cada caso.

A continuación se detalla en qué consiste cada caso de uso.

8.1. Casos de uso

8.1.1. Sitio web informativo

Este es el caso más simple, donde no se requiere ningún tipo de procesamiento. El contenido que tiene este sitio es información sobre los casos de uso implementados, como también instrucciones o referencias de sus ecosistemas de despliegue.

Requisitos funcionales

- **Landing page del proyecto**: sitio web informativo donde se presente lo que se fue haciendo en este proyecto, información sobre los casos de uso y sus ecosistemas de despliegue.

8.1.2. Repositorio de conocimiento

Con este caso estaríamos analizando la capacidad de creación y modificación de contenido existente. La idea es que sea un servicio comunitario donde agregar información de distinta índole, similar a Wikipedia. Se podrán crear usuarios con roles de moderador y editor. Los moderadores tendrán mayor poder de decisión sobre qué contenido publicar y qué no, además de poder bloquear usuarios de ser necesario. Los editores se encargarán de agregar, eliminar y modificar el contenido del sitio buscando tener la información lo más actualizada posible. Este servicio contará con un front end, back end y una base de datos distribuidas utilizando la tecnología OrbitDB.

Requisitos funcionales

- **Edición:** los artículos dentro del repositorio deben ser editables por cualquier persona que ingrese al sitio, y este cambio debe verse reflejado (eventualmente) en las demás personas que accedan a ese artículo.
- **Historial de versiones:** cada artículo debe tener una lista de versiones anteriores, junto con hipervínculos con los cuáles acceder a ellas (mientras estén disponibles en la red).
- **Búsqueda:** una persona debe poder realizar una búsqueda global de todos los artículos.

8.1.3. Mensajero en tiempo real

Este caso se enfoca en la capacidad de la infraestructura de enfrentarse a situaciones de *tiempo real* como puede ser un chat de texto o de audio. En particular nos centraremos en el caso de chats de texto para un grupo de usuarios en donde los mensajes sean públicos. Este servicio también contará con front end, back end y una base de datos donde persistir los mensajes.

Requisitos funcionales

- **Usuarios:** se deben contar con usuarios que puedan iniciar sesión con una contraseña.
- **Grupos públicos:** grupos de chat de texto, donde cualquier usuario puede ingresar y ver los mensajes del resto, así como también participar enviando sus propios mensajes.

8.2. Proceso de descubrimiento

Durante la implementación de los casos de uso para cada ecosistema, se fue desarrollando un mejor entendimiento de lo que se estaba creando. Logrando así generar distintas abstracciones que representan la infraestructura general para la implementación de los casos de uso.

Para cada ecosistema hay 2 principales infraestructuras en acción. La infraestructura de despliegue y la infraestructura de aplicación.

La **infraestructura de despliegue** es la encargada del "hosting"de una aplicación web. Con esta se distribuye y permite el acceso a lo que es el "front end"de las aplicaciones, como también el código para el funcionamiento de la aplicación, si se trata de una aplicación no estática. Cualquier aplicación que desee ser accedida por la web va a hacer uso de esta infraestructura, como es el caso del sitio web informativo.

La **infraestructura de aplicación** es la encargada de la lógica de la aplicación, como es el almacenamiento de datos. Aplicaciones que requieran de mantener un estado y permitir la modificación de parte de los usuarios van a necesitar hacer uso de esta infraestructura, como es el caso del repositorio de conocimiento y mensajero en tiempo real.

(Mover a IPFS esto, ya que no se hicieron abstracciones en blockchain) Al lograr encontrar estas abstracciones, se permite que generar nuevos casos de uso sea mucho más fácil, ya que la mayoría de la lógica sobre el el ecosistema se encuentra encapsulada dentro de ellas, logrando que el desarrollo de un nuevo caso de uso se concentre únicamente en sus requisitos y no en el ecosistema en el que se encuentra.

A continuación se va a explicar como se componen y funcionan ambas infraestructuras en cada ecosistema.

(Explicar que lo que hicimos son packages)

8.3. IPFS

8.3.1. Infraestructura de despliegue

En IPFS, es posible desplegar una aplicación web subiendo un directorio con todos los archivos estáticos necesarios para el funcionamiento en un navegador, incluyendo el código necesario a nivel aplicación. Esto se puede realizar manualmente mediante cualquier cliente de IPFS, como [16] o [6], y devuelve un *Content identifier* (CID) [12] que representa esa versión de la aplicación.

Las aplicaciones web son una de los principales casos de uso de IPFS. Los sitios web estáticos se complementan con el *content addressing*, ya que su CID se mantiene y no requiere cambiar hasta que se actualice su contenido. Cualquier usuario puede publicar su sitio web en la red de IPFS a través de un nodo local de manera gratuita y poco tiempo. IPFS provee un tutorial en su página de cómo realizarlo [7]. A continuación se detallará las implicaciones que tiene desplegar una aplicación web de esta manera, y que alternativas existen para publicar en IPFS.

Al subir un archivo, por ejemplo código HTML, este se transforma en una representación de contenido direccionable mediante un CID. Desde ese momento, cualquier nodo que quisiese obtener el archivo, puede encontrarlo mediante su CID. Sin embargo, no se asegura la persistencia del archivo, y dejará de ser accesible luego de un tiempo. Se debe al [13] implementado por IPFS, que desecha datos para liberar almacenamiento de forma arbitraria. Es por esto que existe el concepto de [20]: *"pinear"* un archivo significa instruir al nodo IPFS para tratar a un archivo o directorio presente en la red de IPFS como esencial y, por lo tanto, no descartarlo. Sin embargo, *"pinear"* el archivo o directorio, no significa que estará disponible de forma indeterminada en el tiempo, debido a que todavía depende de que el nodo que esté pineando el contenido esté activo, o que otros nodos que hayan accedido al archivo aún lo tengan en su caché para permitir que otros puedan acceder a él. Además, para mejorar la disponibilidad de un archivo, el caso ideal sería que varios nodos "pineen.[el] archivo, de manera que otros nodos que quieran obtener el contenido puedan hacerlo desde cualquiera de estos nodos.

Para lograr que el contenido persista en la red sin necesidad de que el nodo local esté activo, existen opciones para delegar el **pineo** del archivo o directorio. Existen servicios de *pinning* y clusters colaborativos, que actuan pinneando los archivos en múltiples nodos, aumentando no solo su disponibilidad sino también su distribución, y por ende logrando un acceso más rápido al contenido.

Servicios de *pinning* La manera más fácil de asegurarse que los datos estén disponibles y se persistan es usar un servicio de *pinning* [14]. Estos servicios cuentan con varios nodos que pinnean archivos. De esta manera, ya no es necesario contar con un nodo local que los aloje. Algunos ejemplos de servicios de pinning incluyen Fleek [4] y Pinata [19].

Desde el punto de vista de las aplicaciones estrictamente comunitarias, estos servicios no van de la mano con su filosofía. Por un lado, los servicios de *pinning* tienen un modelo gratis con funcionalidad limitada o capacidad de almacenamiento limitado. Por otro lado, se depende de estos servicios, lo que en esencia centraliza el proceso de despliegue de la aplicación o sitio web. Si por algún motivo el servicio dejara de "pinear"los archivos, estos pueden dejar de estar disponibles en la red IPFS, e incluso pueden perderse por completo. Esto rompe completamente con la naturaleza de aplicaciones descentralizadas y pasa a tener una centralización tercerizada similar a utilizar AWS.

Clusters colaborativos Estos son grupos de nodos de IPFS que actuan colaborativamente para "pinear.[el] contenido que se agregue al cluster, por uno o múltiples peers. De esta manera, podemos lograr que los usuarios opten por colaborar con su nodo local para el "pinneo"de la aplicación. Así, se logra que la misma comunidad mantenga en servicio el mecanismo de despliegue de la aplicación, lo cuál es acorde a la filosofía de aplicaciones comunitarias.

Actualmente, esta alternativa es que es poco explorada, por lo tanto no existe una forma fácil de creación, seguimiento y descubrimiento de estos clusters. IPFS cuenta con una página con clusters conocidos con los cuales se puede colaborar, pero la cantidad de clusters es limitada.

Por otro lado, el principal problema es que los clusters obligan a los nodos a "pinear"la totalidad de sus archivos, lo cuál puede significar el uso de cientos de gigabytes en almacenamiento necesarios únicamente para colaborar. Pinear parte del contenido es imposible actualmente. Esto se debe a que, debido a la manera en la que fue diseñada la arquitectura de IPFS, un nodo puede mentir acerca de los archivos que tiene, por lo que hay una posibilidad de que una parte del contenido no esté en ninguno de los nodos, y por ende el contenido esté incompleto.

Acceso y mutabilidad Para buscar un contenido, un nodo de IPFS realiza una búsqueda a través de su CID, el cual es único. Debido a que es único, el CID cambiará si el contenido del sitio web o aplicación web cambia, ya que el contenido será distinto. Esto vuelve el proceso de despliegue altamente impráctico, ya que se necesitaría compartir un nuevo CID cada vez que se actualice una página.

Este problema puede ser resuelto con la ayuda de *punteros mutables*. Estos punteros son un objeto de IPFS que apunta a un CID determinado, previamente elegido por el usuario. El CID al que apunta el puntero puede ser cambiado, por lo tanto permiten compartir la dirección del puntero una única vez y actualizar el CID al cuál apunta cada vez que se haga un cambio.

IPNS El InterPlanetary Name System (IPNS) [10] es un sistema que permite crear punteros mutables y obtener su dirección en forma de CIDs conocidos como *names* o *nombres de IPNS*. Estos nombres de IPNS pueden considerarse como enlaces que pueden actualizarse, conservando al mismo tiempo la verificabilidad del content addressing.

Un nombre de IPNS es un hash de una clave pública. Está asociado a un *IPNS record* que contiene la ruta a la que se vincula, entre otra información. El titular de la clave privada puede firmar y publicar nuevos registros en cualquier momento.

Gráfico de IPNS

Es posible utilizar IPNS con uno de estos posibles enfoques:

- **Consistencia:** garantizar que los usuarios siempre resuelvan el último registro de IPNS publicado, a riesgo de no poder resolverlo.
- **Disponibilidad:** resolver un registro de IPNS válido, a costa de potencialmente resolver un registro desactualizado -o sea, con un CID previo.

El registro IPNS se encuentra a través de la **Distributed Hash Table** (DHT). Todos los nodos de IPFS participan alojando colaborativamente el contenido de la DHT. Por lo tanto, el DHT actúa como un "directorio"descentralizado, donde la clave pública es un identificador. Esta tabla ayuda a localizar el registro IPNS que apunta al contenido deseado, entre otras funciones. Para entender mejor cómo IPNS funciona se puede consultar la documentación de IPFS.

IPNS es una buena forma de obtener mutabilidad dentro de IPFS. Una vez que se aloja un contenido en IPFS y se apunta a él mediante un *IPFS name*, el mayor problema pasa a ser la manera de acceder a IPNS en sí. El hecho de que los names sean hashes alfanuméricos, y no nombres legibles o memorables para humanos, representa una dificultad adicional a la hora de alojar un sitio web al cuál los usuarios puedan acceder fácilmente. A continuación se analizará dos alternativas para solucionar este problema.

(explicar como se puede utilizar con un cluster colaborativo a traves de pub sub)

DNSLink IPNS no es la única forma de crear mutable pointers en IPFS. DNSLink [3] utiliza registros *DNS TXT* para asignar un nombre DNS (por ejemplo, un dominio) a una dirección IPFS o a un *IPNS name*. Como uno puede editar sus registros DNS, puede usarlos para que siempre apunten a la última versión de un objeto en IPFS.

DNSLink actualmente es mucho más rápido que IPNS, utiliza nombres legibles por humanos y también puede apuntar a nombres IPNS. A pesar de ello, tiene un problema muy fundamental y es que se utiliza el protocolo DNS, el cual tiene claras deficiencias con la filosofía de aplicaciones comunitarias.

La más importante es que, aunque DNS tenga claras ventajas, como que es un sistema distribuído y escalable, es también un sistema algo centralizado. Las autoridades centrales como *ICANN* gestionan las raíces del DNS. Esto hace que un registro DNS sea fácil de censurar, a nivel de registrador como también de los *ISPs*.

ENS ENS, Ethereum Name Service es el protocolo de nombres descentralizado que se basa en la Ethereum blockchain. Funciona de manera similar a DNS, en el sentido de que los nombres ENS resuelven a nombres legibles para humanos. Como esto se computa en la blockchain de Ethereum, es seguro, descentralizado y transparente.

Es posible configurar un registro ENS para que se resuelva en la dirección IPNS, proporcionando nombres legibles para humanos que son más fáciles de compartir y acceder, y solucionando el principal problema de IPNS hasta este punto.

Un registro de ENS puede apuntar a un *IPNS name*, haciendo uso de las ventajas de ambos sistemas. Cuando se quiera actualizar el contenido, no será necesario modificar el registro ENS en sí, ya que siempre se va a apuntar al mismo name de IPNS.

Cabe aclarar que adquirir un dominio ENS tiene un costo, es un costo el cual no suele ser muy elevado y trae todos los beneficios mencionados, especialmente para sitios web estáticos y también cualquier aplicación comunitaria. Pero sigue siendo un paso opcional en el despliegue de sitios web o web apps, ya que el IPNS name sigue siendo completamente accesible sin un registro de ENS apuntanto a él.

Despliegue continuo En un proyecto de aplicación web centralizada, es común automatizar el proceso de despliegue con cada cambio que se realiza. Normalmente este proceso se activa con cada nuevo commit en una rama de Git especifica, e incluye todas las etapas necesarias para convertir el contenido de un repositorio Git en código estático listo para ser desplegado. También puede incluir más pasos que incluyan actualizaciones en el backend.

Yendo al caso específico de aplicaciones web comunitarias, el script debe ser ejecutado en los nodos confiables, ya que una *Github action* no puede utilizar un nodo IPFS que requiera puertos abiertos. En este tipo de aplicaciones, al tener una jerarquía mayormente horizontal, no hay un servidor central que orqueste esta actualización, sino que se necesita que cualquier nodo confiable pueda actualizar su contenido e instruir a los nodos colaborativos para actualizar su contenido de

igual forma. Todo esto debe ser posible incluso cuando los nodos no reciben la actualización al mismo tiempo, es decir, no debe haber *race conditions*.

Una forma de lograr esto es, por ejemplo, utilizar un algoritmo de elección de líder u otro algoritmo distribuido para elegir el nodo responsable de indicar el nuevo contenido a pinear al resto de nodos en el cluster. Sin embargo, esta manera de realizar la actualización implica una capa adicional de complejidad que no es necesaria debido a la naturaleza de IPFS.

Como ya se ha mencionado, si dos nodos suben el mismo contenido, obtendrán el mismo CID. Esto puede ser utilizado para que cualquier nodo confiable pueda actualizar el contenido y el nombre de IPNS independientemente del resto de los nodos confiables. Cuando se detecte un cambio nuevo, el nodo puede obtener el código estático, y acto posterior, indicar al resto de los nodos del cluster que pineen el CID especifico. En el caso de que sea el primer nodo en detectar el cambio, deberá instruir al resto del cluster para que dejen de pinear el CID antiguo. En el caso en que otro nodo haya detectado la actualización antes, no deberá actualizar ningún pin del cluster debido a que el mismo CID ya va a estar presente en la lista de pins.

Compilación Las herramientas de compilado no siempre son deterministas en los archivos compilados que genera. Next.js, por ejemplo, genera diferentes archivos estáticos en dos compilaciones basadas en el mismo código fuente. Esto es un problema para el enfoque propuesto, debido a que si dos nodos compilan el mismo código, el CID puede ser diferente. Para mitigar esto, se decidió hacer uso de un *hook* que compile el código con cada *commit* en la rama principal una única vez por cambio realizado. De esta manera, los nodos confiables pueden detectar el cambio en la rama utilizada para alojar los archivos estáticos, y hacer *pull* sobre esos archivos y, por lo tanto, obtener un mismo CID.

service.json Para que un usuario pueda conectarse y contribuir como colaborador a un cluster, la herramienta de terminal ***ipfs-cluster-follow*** requiere una dirección de IPFS de la cuál obtener el archivo *service.json*. Este archivo de configuración contiene todos los datos necesarios para que un colaborador pueda unirse. Además, está sujeto a modificaciones, debido a que el archivo contiene las *multiaddresses* de cada nodo confiable en forma de lista, por lo que agregar o remover un nodo confiable implica modificar el archivo.

Es por esto que el proceso de despliegue también debe incluir este archivo. Debe incluirse la detección de una actualización, el pineo del nuevo *service.json* al cluster,

Enfoque En base a este análisis, podemos concluir que la mejor forma de desplegar una página web estática en IPFS es a través del uso de un cluster colaborativo compuesto por nodos confiables y nodos colaboradores, así como una dirección IPNS a la cuál actualizar cada vez que hay un cambio, y un registro ENS para traducir la dirección IPNS a un nombre legible.

Este enfoque tiene, sin embargo, desventajas o aspectos a mejorar:

Necesidad de tener nodos confiables Estos nodos van a ser los encargados de administrar el cluster, y actualizar el IPNS. La distinción entre nodos confiables y nodos colaborativos es necesaria para evitar que un potencial atacante pueda modificar el CID al que apunta el *IPNS name* o modificar el contenido que pinea el cluster colaborativo.

Actualización del contenido Por cada cambio que se realice en el directorio de la página, se deberá pinear el nuevo contenido al cluster, y por lo tanto todos los colaboradores tendrán que obtener todo el directorio nuevamente. Esto puede claramente volverse costoso con contenido de tamaño considerable.

Cache de IPNS El parámetro TTL de IPNS indica cuanto "viveün valor asociado a un nombre de IPNS en la cache de un nodo antes de forzar a este a volver a buscar el valor en la

DHT. El problema que tiene esto es que, si se pone un valor muy elevado, un nodo gateway no buscará la actualización hasta que se cumpla el periodo y por lo tanto el registro de IPNS no se actualizará. Por otro lado, si se pone un valor muy corto, siempre se buscará el valor en la DHT, generando latencia al no utilizar el cache disponible. Pero a su vez, el nombre de IPNS en un nodo siempre tendrá la última versión que encuentre.

Compartir claves privadas Cómo la actualización de un nombre de IPNS está firmada con una clave privada, todos los nodos confiables deberan tener la misma clave para poder potencialmente actualizar el registro IPNS y así evitar tener un único nodo con esa responsabilidad. Esto elimina un punto de falla único, pero aumenta las chances de que esa clave privada llegue a manos de un posible atacante.

Necesidad de re-publicar el nombre de IPNS cada cierto tiempo Kubo (una de las implementaciones de un nodo de IPFS) actualiza el nombre de IPNS cada hora para asegurarse de que siga estando en la DHT. Pero si se cae el nodo que publicó último puede que no se actualice mas, esto se tiene que tener en cuenta

Apertura de puertos IPFS cluster utiliza el puerto 9096 para la comunicación entre nodos, el cual se tiene que abrir para un correcto funcionamiento. Se podría usar hole punching (?).

8.3.2. Infraestructura de aplicación

8.4. Blockchain

En este trabajo se utilizó la red de Ethereum, al ser una blockchain popular nos permite demostrar y comparar los casos de uso contra nuestra solución en IPFS. Ethereum está compuesta de nodos distribuidos que comparten poder de cómputo lo cuál permite el desarrollo de aplicaciones descentralizadas. Cuenta con una moneda que funciona a modo de incentivo, es decir, que los nodos reciben ganancias por formar parte de la red. Esto conlleva a que los usuarios de la red necesiten pagar para utilizarla a través de transacciones.

8.4.1. Infraestructura de despliegue

8.4.2. Infraestructura de aplicación

8.5. FrontEnd

Astraweb

9. Metodología

El desarrollo se dividió en sprints semanales para los cuales utilizamos un tablero Kanban en Github Projects donde fuimos agregando las tareas a realizar para cada caso de uso. Se realizaron reuniones semanales fijas que se usaron como punto de control, donde se revisó lo hecho durante la semana y definimos pasos a seguir para las siguientes. También nos fue útil para detectar posibles ajustes o cambios de rumbo que fueron surgiendo a lo largo del trabajo.

La modalidad fue en su mayoría virtual y asincrónica (excepto por la reunión semanal antes mencionada en la cual los integrantes del trabajo nos reunimos sincrónicamente). Nos mantuvimos en constante comunicación a través de un servidor de Discord y, también se realizaron sesiones de *pair* y *mob-programming* en distintas ocaciones.

10. Experimentación y/o validación

10.1. Costos

¿Cuánto nos cuesta desplegar y mantener un servicio en cada ecosistema?

10.1.1. IPFS

10.1.2. Blockchain

De los casos de uso esperamos responder las siguientes incógnitas:

Swarm Al deployar el sitio web es necesario contar con *postage stamps* que son la manera de pagar por el uso del almacenamiento en Swarm. Cada actualización que se realice al sitio requiere de *postage stamps* y, además, estos tienen fecha de vencimiento por lo que es necesario volver a pagar frecuentemente. Hay que tener en cuenta que dichos *postage stamps* se pagan en la criptomoneda BZZ que fluctúa de valor con respecto al dólar estadounidense.

La obtención del sitio web no requiere de costo alguno, por lo que desde el punto de vista de un usuario de la aplicación no sería necesario pagar.

¡TODO: medir cuánto es el costo aproximado en USD o BZZ¿

Ethereum Se utiliza la moneda ETH para pagar por el despliegue de cada transacción, esto incluye tanto el despliegue de cada *smart contract* como también la edición de un artículo (en el caso del repositorio de conocimiento). Por lo tanto, el usuario final de la aplicación termina pagando por creación y edición de cada artículo en el repositorio de conocimiento, y por cada mensaje enviado en el mensajero en tiempo real. Por otro lado, para las operaciones de lectura no se tiene que pagar nada.

¡TODO: medir cuánto es el costo aproximado en USD o ETH¿

10.2. Facilidad de desarrollo

¿Qué tan fácil es desplegar en cada ecosistema?

10.2.1. IPFS

10.2.2. Blockchain

Swarm En Swarm existe la herramienta de terminal [‘swarm-cli‘](https://github.com/ethersphere/swarm-cli) con la cual se puede interactuar con un nodo de Swarm. También el equipo de Swarm provee una *Github Action* que permite la posibilidad de automatizar el despliegue generando un pipeline que utilice dicha herramienta.

En cuanto a un ambiente de pruebas o staging, si bien no existe un gateway público que interactúe con la *testnet*, es posible levantar uno propio que sí lo haga apuntando a la *testnet* de Sepolia usando la herramienta [‘gateway-proxy‘](https://github.com/ethersphere/gateway-proxy).

Ethereum Con la librería web3.js se puede interactuar con un nodo de Ethereum y realizar un despliegue de la aplicación. Además, existen las herramientas de Hardhat con las cuales se puede levantar una red de prueba que facilita el desarrollo local.

10.3. Viabilidad

¿Que tan viable es crear una aplicación comunitaria para cada uno de estos ecosistemas?

10.3.1. IPFS

10.3.2. Blockchain

Swarm Resulta más conveniente para sitios web o recursos estáticos. No es posible la ejecución de código.

Ethereum Su punto fuerte es la ejecución de código, por lo cual es útil para funcionar como backend para aplicaciones web. Por el costo de almacenamiento de los smart contracts no es recomendable para sitios o recursos estáticos como imágenes o videos.

10.4. Performance

10.4.1. IPFS

10.4.2. Blockchain

10.5. Resumen

11. Cronograma

Realizamos un cronograma tentativo de la totalidad del trabajo, incluyendo el desarrollo de cada caso de uso, el despliegue en cada ecosistema y su documentación asociada.

Cada caso de uso incluye una etapa de *Discovery* en la cuál definiremos su alcance y lo desglosaremos en tareas más concretas.

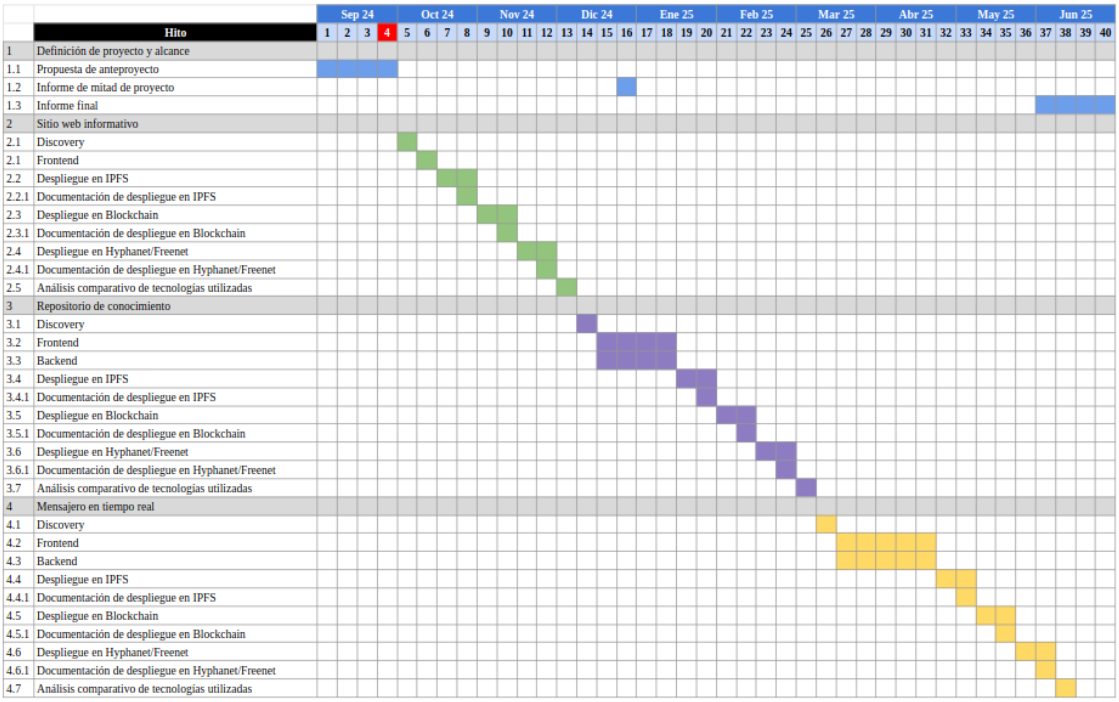

Figura 1: Cronograma tentativo

1. **Definición de proyecto y alcance**
   - 1.1. Propuesta de anteproyecto (Semanas 1 a 4)
   - 1.2. Informe de mitad de proyecto (Semanas 16 a 17)
   - 1.3. Informe final (Semanas 37 a 40)
2. **Sitio web informativo** (Semanas 5 a 13)
   - 2.1. Discovery
   - 2.2. Frontend
   - 2.3. Despliegue en IPFS
     - 2.3.1. Documentación de despliegue en IPFS
   - 2.4. Despliegue en Blockchain
     - 2.4.1. Documentación de despliegue en Blockchain
   - 2.5. Despliegue en Hyphanet/Freenet
     - 2.5.1. Documentación de despliegue en Hyphanet/Freenet
   - 2.6. Análisis comparativo de tecnologías utilizadas
3. **Repositorio de conocimiento** (Semanas 14 a 25)
   - 3.1. Discovery
   - 3.2. Frontend
   - 3.3. Backend
   - 3.4. Despliegue en IPFS
     - 3.4.1. Documentación de despliegue en IPFS
   - 3.5. Despliegue en Blockchain

12. Riesgos materializados

Cambio de Hyphanet a Freenet A mitad del proyecto resolvimos cambiar el tercer ecosistema elegido (Hyphanet) por su versión más moderna (Freenet). Esto fue debido a que encontramos que la documentación era escasa, los programas realizados para el ecosistema eran unos pocos y cada uno tenía una forma distinta de implementar ciertas cosas. La API tampoco provee facilidades a la hora de gestionar archivos, manejo de comunicaciones, entre otras cosas que consideramos necesarias para los casos de uso.

Freenet en desarrollo Un riesgo que teníamos en cuenta eran las modificaciones que podría sufrir Freenet al estar aún en desarrollo. Esto fue de la mano con que la documentación publicada no está actualizada a la última versión.

Baja de Freenet como ecosistema Dada la promesa del equipo de Freenet de lanzar una versión estable en el corto plazo -pero que ya llevaba más de un año en ese estado- decidimos poner como límite el mes de febrero de 2025. Llegada la fecha, no hubo ningún anuncio de la versión estable por lo que decidimos descartar el ecosistema y, en cambio, agregar métricas de performance a los otros ecosistemas.

13. Lecciones aprendidas

- Trabajar con tecnologías emergentes resulta un desafío al encontrarse en desarrollo constante y frecuente. Esto quiere decir que la documentación es escasa, nula o se encuentra desactualizada.
- Al trabajar con distintos ecosistemas, modularizar en distintos paquetes/librerías cada uno facilita la integración, las pruebas y el cálculo de métricas.

14. Impactos sociales y ambientales

15. Trabajos futuros

16. Conclusiones

16.1. Conclusión del análisis

16.2. Conclusión general

17. Referencias

[1] Brnakova, J. (s.f.). *Game decommissioning: When beloved games shut down*. https://www.revolgy.com/insights/blog/game-decommissioning-when-beloved-games-get-shut-down-and-online-worlds-disappear
[2] *Censorship of Wikipedia*. (s.f.). https://en.wikipedia.org/wiki/Censorship_of_Wikipedia
[3] *DNSLink*. (s.f.). https://docs.ipfs.tech/concepts/dnslink/#dnslink
[4] *Fleek*. (s.f.). https://fleek.xyz/docs/platform/hosting/
[5] *Freenet*. (s.f.). https://freenet.org
[6] *Helia: IPFS node implementation in Javascript*. (s.f.). https://github.com/ipfs/helia
[7] *Host a single-page website with IPFS Desktop*. (s.f.). https://docs.ipfs.tech/how-to/websites-on-ipfs/single-page-website/#set-up-a-domain
[8] Hurtado, J. S. (s.f.). *Qué es Blockchain y cómo funciona la tecnología Blockchain*. Consultado el 25 de septiembre de 2024, desde https://www.iebschool.com/blog/blockchain-cadena-bloques-revoluciona-sector-financiero-finanzas
[9] *Hyphanet*. (s.f.). https://www.hyphanet.org/index.html
[10] *InterPlanetary Name System*. (s.f.). https://docs.ipfs.tech/concepts/ipns
[11] *IPFS*. (s.f.). https://ipfs.tech
[12] *IPFS Content Identifiers*. (s.f.). https://docs.ipfs.tech/concepts/content-addressing/#what-is-a-cid
[13] *IPFS Garbage Collector*. (s.f.). https://docs.ipfs.tech/concepts/persistence/#garbage-collection
[14] *IPFS Pinning services*. (s.f.). https://docs.ipfs.tech/concepts/persistence/#pinning-services
[15] Janardhan, S. (s.f.). *More details about the October 4 outage*. https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/
[16] *Kubo: IPFS node implementation in Go*. (s.f.). https://docs.ipfs.tech/install/command-line/
[17] *libp2p*. (s.f.). https://libp2p.io/
[18] *OrbitDB*. (s.f.). https://orbitdb.orghttps://orbitdb.org
[19] *Pinata*. (s.f.). https://pinata.cloud/ipfs
[20] *Pinning*. (s.f.). https://docs.ipfs.tech/concepts/glossary/#pinning

18. Anexos

Acá irían los anexos de todo nuestro trabajo :)