VON WEB 2.0 ZU WEB3

Dezentralität ist die Zukunft

Wie Web3-Technologien das Internet auf den Kopf stellen.

Die Basis jeglicher Online-Aktivitäten ist die IT-Sicherheit. Das wurde in den vergangenen Monaten deutlich, in denen durch den schnellen Wechsel zahlreicher Unternehmen zu Remote Work die Zahl der Cyberattacken massiv stieg [1]. Das Abfischen milliardenfacher personalisierter Datensätze, die Einführung der Europa-Cloud „Gaia-X" und Sicherheitsprobleme, die Behörden, Bundestag, Universitäten und Industrie zeitweise lahmlegten, deuten auf das Offensichtliche: Das Internet selbst benötigt ein völliges Umdenken.

Web3-Technologien halten Informationen auf den Endgeräten der Nutzer vor, statt sich auf Server- oder Cloud-basierte Lösungen zu verlassen. Solche dezentralen Systeme erzielen eine erhöhte Datensicherheit durch die Verschiebung der Datenhoheit hin zum Nutzer und sind mangels zentraler Infrastruktur nahezu unangreifbar. Aber auf welchem Stand befinden sich Web3-Technologien aktuell, und was können sie bereits leisten?

Was ist das Web3 genau?

Das Web3 versteht sich als die nächste Evolutionsstufe des Internets. Es gibt keine einheitliche Definition, welche Technologien zum Web3 zählen; grundsätzlich umfasst der Begriff dezentrale Netzwerke, Dienste, Protokolle und Plattformen, die keine einzelne Instanz kontrolliert und denen dennoch jeder vertrauen kann. Das wird möglich, weil jeder Nutzer und Betreiber dieser Netzwerke denselben fest codierten Regeln folgt.

Bekannteste Vertreter sind Blockchains wie Ethereum oder Polkadot, die mit ihren Konsensprotokollen einen global verteilten und aus seiner Transaktionshistorie eindeutig determinierten Applikationszustand herstellen. Dafür betreiben einander völlig unbekannte Teilnehmer global replizierte State Machines, deren Zustandsänderungen von sogenannten Smart Contracts als Reaktion auf Transaktionen der Benutzer ausgelöst werden. Damit ist nicht nur die Grundlage für die Aufbewahrung und Übermittlung von Werten wie dem Bitcoin gelegt, sondern eine im klassischen Sinne Multimaster-schreibfähige Datenbank mit Leadership-freiem globalem Konsens über ihren Inhalt entstanden.

Während das Web 2.0 das Frontend von Webseiten revolutionierte, ist das Web3 eher eine Backend-Revolution. Backend-Entwickler müssen sich im Web3 auf einen tiefgreifenden Paradigmenwechsel vorbereiten: Für die Kontrolle von Informationen benötigt man keine zentrale Datenbank mehr!

IPFS: Ein dezentrales Dateisystem

Das „interplanetare Filesystem" IPFS besteht im Kern aus einem selbstorganisierenden P2P-Netzwerk, über das dessen Nutzer mit jedem anderen Nutzer auf der Welt Daten teilen können. Jeder Node im IPFS-Netzwerk verteilt dabei Datenfragmente so, dass sie – sofern ein bestimmter Datensatz von hinreichend vielen Nodes oft genug angefragt wird – immer verfügbar sind.

IPFS läuft per Default vollständig offline und benötigt keinen laufenden Service für minimale Funktionalität. Mithilfe eines einfachen, an POSIX-Filesystem-Kommandos angelehnten API lassen sich Dateien und Ordner lokal in einem IPFS-Repository speichern. Unter der Haube zerteilt IPFS alle Dateien in Blöcke konstanter Größe, berechnet einen einfachen Hash über den binären Inhalt eines Blocks und verbindet sie mithilfe eines Merkle-DAG zu einer kryptografisch abgesicherten Struktur, die die Integrität und Authentizität der Datei garantiert. Die Blöcke werden in einem lokalen ▶

Listing 1: Einfache FS-Operationen mit dem IPFS-CLI

```
mkdir website
echo "Hello, Web3 World" >> website/index.html

# Ordner rekursiv veröffentlichen
ipfs add -r website
> added QmdTNjHwkc79cxBvcrxxMZuaakxaU56jqvovdbHAM
    dyxx6 website/index.html
> added QmRj5vMxG9A1iMPiAiMyShJq8hFRZyhgnMxVwC
    VoP82tju website

# Eine Datei via IPFS beziehen und anzeigen
ipfs cat QmdTNjHwkc79cxBvcrxxMZuaakxaU56jqvovdbHAM
  dyxx6
> Hello, Web3 World

# Eine benannte Datei aus einem Verzeichnis über ein
# öffentliches HTTP-Gateway beziehen
curl https://ipfs.io/ipfs/QmRj5vMxG9A1iMPiAiMySh
  Jq8hFRZyhgnMxVwCVoP82tju/index.html
> Hello, Web3 World
```

Key-Value-Store abgelegt und ihre Prüfsummen in eine Hash-Tabelle geschrieben.

Sobald man IPFS als Daemon startet, verbindet es sich mit dem globalen „Schwarm" von anderen IPFS-Peers und integriert die lokale Hash-Tabelle in eine global verteilte „Distributed Hash Table", die DHT. Um eine Datei zu veröffentlichen, teilt man seinen Nachbarn die Content-ID (CID) der Datei mit, die als Hash über den Merkle-DAG aller ihrer Blöcke gebildet wurde. Benötigt ein Node eine Datei und kennt er deren CID, so versucht er, eine Route durch das P2P-Netzwerk zu einem Node zu finden, der den Inhalt hinter der Content-ID liefern kann. Zur Stunde null ist das der Rechner, der den Content erstmals veröffentlichte. Dafür bittet er alle mit ihm verbundenen Peers, in ihrem lokalen Teil der DHT nachzuschauen, ob sie die Datei haben oder jemanden kennen, der sie hat. Einfache FS-Operationen mit dem IPFS-CLI sind in Listing 1 zu sehen.

Zur Adressierung von Inhalten in IPFS genügt es, deren Content-ID zu kennen. Da diese als Hash über den binären Inhalt gebildet wird, können Clients nach dem Empfang der Daten prüfen, ob die CID zum Inhalt passt, und damit garantieren, dass der Node, der ihn geliefert hat, sie unterwegs nicht verändert hat. Das ermöglicht es dem IPFS-Netzwerk auch, Inhalte zu deduplizieren: Für Clients spielt es keine Rolle, wer ein Katzenbild zuerst hochgeladen hat.

IPFS selbst ist nur ein schmaler Filesystem-Layer über der Netzwerkbibliothek libp2p und sich selbst beschreibenden „Multiformaten". Die Entwickler rund um Protocol Labs und dessen Gründer Juan Benet legen größten Wert darauf, IPFS nicht auf ein Netzwerkprotokoll („TCP"), eine bestimmte Hashfunktion („Blake2b-256") oder eine spezielle Codierung („Protobuf") festzulegen. Mithilfe von Multiformaten lassen sich Adressen und Werte so darstellen, dass Clients genau wissen, wie sie zum Beispiel eine Binärcodierung interpretieren oder über welches Netzwerkprotokoll sie sich mit einem anderen Node verbinden sollen (Listing 2).

Der IPFS-Stack ist derzeit sowohl in Go als auch in JavaScript vollständig implementiert. Man kann daher einen fast uneingeschränkten IPFS-Node innerhalb eines Browserfensters starten und ohne weitere Abhängigkeit im Rahmen einer Website mit dem IPFS-Protokoll interagieren.

JAMStack: Statische Websites als Paradebeispiel für IPFS-Anwendungen

Die Fähigkeiten von IPFS ermöglichen es, statische Webseiten abzulegen und auszuliefern. Mit Ausnahme von Brave benötigen Browser für den Abruf von Webinhalten allerdings ein HTTP-Gateway, da sie ohne entsprechende Extension derzeit noch nichts mit dem IPFS-Protokoll anfangen können. Jeder IPFS-Daemon startet automatisch eines, sodass man jede auf IPFS gehostete Webseite auch über das Gateway seines eigenen Nodes abrufen kann; die URLs

- *https://ipfs.io/ipfs/QmRj5vMxG9A1iMPiAiMyShJq8hFRZyhgnMxVwCVoP82tju/*
- *http://localhost:8080/ipfs/QmRj5vMxG9A1iMPiAiMyShJq8hFRZyhgnMxVwCVoP82tju/*
- *https://dweb.link/ipfs/QmRj5vMxG9A1iMPiAiMyShJq8hFRZyhgnMxVwCVoP82tju*

liefern daher alle dasselbe Resultat.

Dank der Popularität von JAMStack-basierten SPAs/PWAs gewinnen statische Webseiten wieder massiv an Attraktivität. Der JAMStack ist eine der Frontend-Entwicklung entsprungene Applikationsarchitektur, die lediglich JavaScript, APIs und Markup voraussetzt und für den Betrieb einer Webseite abgesehen von einem Webserver beziehungsweise CDN von keinem weiteren Dienst abhängt. Static-Site-Generatoren (SSG) wie Gatsby, Hugo, Eleventy oder Next.js rendern unter Zuhilfenahme von Headless CMS und APIs zum Build-Zeitpunkt alle Seiteninhalte vor. Cloud-Dienste wie AWS Amplify, Netlify oder Cloudflare haben sich auf Git-basierte Workflows für SSG-Applikationen spezialisiert, die die relativ rechenaufwendigen Build-Phasen nach jedem Push durchlaufen und die statischen Build-Artefakte auf den jeweiligen CDNs der Anbieter bereitstellen. Nichts

Listing 2: Multiformate

```
//npm install multihashing multibase lodash.isequal
const multihashing = require('multihashing')
const multibase = require('multibase')
const crypto = require('crypto')
const isEqual = require('lodash.isequal')

const decoder = new TextDecoder()

const bytes = Buffer.from("Hello, dotnetpro");

// create an sha1 multihash and encode it as base64
const someMultihash = multihashing(bytes, 'sha1')
const encoded = multibase.encode('base64urlpad', someMultihash)
console.log(decoder.decode(encoded));
// UERR6F_mTb9hB29WhwmIy8igojmQzvQ==

//decode and check the hash without knowing its encoding / hash function
const b64Decoded = multibase.decode(encoded);
const decodedHash = multihashing.multihash.decode(someMultihash);
const hashFunction = decodedHash.name

const hashed = crypto.createHash(hashFunction).update(bytes).digest()
console.log(isEqual(b64Decoded.slice(2), hashed));
//true
```

liegt nun näher, als die JAMStack-Builds der SSGs auf IPFS zur Verfügung zu stellen.

IPNS, DNSLink und ENS: Dezentrale Namensdienste

Es fehlt aber ein entscheidender Baustein, bevor wir den Vertrag bei unserem Webhoster kündigen können: Da die CID des Root-Ordners einer statischen Website vom Inhalt abhängig ist, müsste man bei jeder Änderung der Webseite die neue CID bekannt machen. IPFS löst dieses Problem selbst mithilfe seines Subsystems IPNS. Hierfür erzeugt man zunächst aus dem öffentlichen Schlüssel eines vom eigenen IPFS-Node verwalteten Schlüsselpaars einen eindeutigen, unveränderlichen Hash-Wert, der als „Anker" für die Adressierung dient. Anschließend veröffentlicht man die aktuelle CID der Webseite über das IPNS-Protokoll (siehe Listing 3).

Unsere Webseite lässt sich nun von kompatiblen Browsern wie Brave mithilfe von IPFS-Browser-Extensions oder mit dem IPFS-CLI permanent unter

```
ipns://k2k4r8np412mc5e09bscwur1f8kz6anxlbixq3rtv2exmfau
  ipc7o4ku
```

erreichen und kann die dort hinterlegte CID des Inhalts auflösen. Das Problem: Diese Zeichenkette kann sich kein Mensch merken!

Eine Lösung dafür verbindet die Web3-Welt mit dem Domain Name System DNS. Um eine herkömmliche Internet-Domain wie „web3.stadolf.de" mit der gerade aktuellen CID des Inhalts zu verbinden, kann man der eigenen Domain einen DNSLink-TXT-Record hinzufügen:

```
dnslink.web3.stadolf.de="dnslink=/ipns/k51qzi5uqu5dmcpy
  yldjeeyqhisauj3pdkjkmiiud5z0180thr7kruoqgnz69r"
```

Die in IPFS integrierten IPNS-Clients und HTTP-Gateways suchen automatisch bei entsprechenden Anfragen nach mit *_dnslink* gepräfixten TXT-Einträgen und lösen die hinterlegten */ipfs/*- oder */ipns/*-Einträge auf:

```
ipfs cat /ipns/web3.stadolf.de/index.html
> "Hello, Web3 World"

curl https://ipfs.io/ipns/web3.stadolf.de/index.html
> "Hello, Web3 World"
```

Mit zwei einfachen Kniffen lässt sich eine herkömmliche statische Website nur mithilfe von DNS und IPFS hosten. Dafür legt man einen zusätzlichen *ALIAS*-Record für die eigene Domain an, der Clients direkt auf ein beliebiges, möglichst stabiles IPFS-HTTP-Gateway weiterleitet:

```
ALIAS web3.stadolf.de=gateway.ipfs.io
```

Der DNS-Server leitet jede Anfrage an die Domain hiermit auf das offizielle IPFS-HTTP-Gateway um, belässt aber den Host-Header bei der ursprünglichen Domain. Das Gateway

Listing 3: IPNS-Operationen mit dem IPFS-CLI

```
# erstellt einen neuen IPFS Schlüssel für IPNS
ipfs key gen web3articlekey
> k51qzi5uqu5dmcpyyldjeeyqhisauj3pdkjkmiiud5z0180thr
    7kruoqgnz69r

# veröffentlicht die /ipfs/-Adresse unter dem
# Schlüssel
ipfs name publish --key=web3articlekey /ipfs/
  QmRj5vMxG9A1iMPiAiMyShJq8hFRZyhgnMxVwCVoP82tju

# löst die im Netzwerk bekannte aktuelle CID hinter
# dem Schlüssel auf
ipfs name resolve k51qzi5uqu5dmcpyyldjeeyqhisauj3pdk
  jkmiiud5z0180thr7kruoqgnz69r
> /ipfs/QmRj5vMxG9A1iMPiAiMyShJq8hFRZyhgnMxVwCVoP82tju
```

prüft, ob es einen DNSLink-Eintrag für *web3.stadolf.de* findet, und löst ihn zu einem IPNS-Eintrag auf. Dieser antwortet mit der aktuellen CID des Contents, der sich wie gewohnt über IPFS ausliefern lässt. Mit DNSLink erreicht man zwar, dass eine Website unter einem menschenlesbaren Namen erreichbar ist und über ein dezentrales Netzwerk ausgeliefert wird, man muss sich aber auf das stark reglementierte und zentralisierte DNS-Protokoll verlassen.

Vollständige Dezentralität lässt sich mit dem Ethereum Naming Service ENS [2] erreichen. Hier registriert man Domänen als NFT (Non-fungible Token) auf der Ethereum-Blockchain und kann mithilfe einfacher Transaktionen ähnlich wie im DNS TXT-Einträge hinterlegen, die von Clients interpretiert werden. Jeder ENS-NFT erzeugt eine vom Nutzer wählbare Domäne unterhalb der nicht offiziell registrierten Top Level Domain *.eth*.

Innerhalb des Ethereum-Ökosystems bedienen sich zahlreiche Teilnehmer des ENS, um Blockchain-Adressen Namen, E-Mails oder Twitter-Handles zuzuordnen, man kann sie aber auch für IPFS- oder IPNS-Einträge nutzen. Browser, die die Namensauflösung von *.eth*-Domains unterstützen, lösen beim Aufruf einer ENS-Domain eine Weiterleitung zu einem HTTP-Gateway mit dem entsprechenden CID-Link aus. Alle anderen Clients können die vom ENS-Projekt bereitgestellte Web2-Brücke .eth.link [3] benutzen, um *.eth*-Domains von ihrem Browser aus zu erreichen.

Dienste wie Fleek.co [4] haben diesen recht komplexen Prozess vollständig automatisiert, damit sich Nutzer auf die Gestaltung ihrer Webseiten fokussieren können. Analog zu Netlify oder Github Actions verbindet sich Fleek mit den Git-Repositories des Nutzers, stößt nach Pushes oder beim Öffnen von Pull Requests den Build Cycle an und veröffentlicht das Ergebnis auf IPFS. Natürlich unterstützt Fleek auch vollautomatische Updates von IPNS-Einträgen und ENS-Domains und dient bei Bedarf auch als Gateway für herkömmliche Domains. ▶

Listing 4: Nutzerprofil (DIDs) lokal erstellen und via Ceramic/IDX veröffentlichen

```
import CeramicClient from "@ceramicnetwork/
  http-client";
import { IDX } from "@ceramicstudio/idx";
import { Ed25519Provider } from "key-did-provider-
  ed25519";
import KeyResolver from "key-did-resolver";
import { DID } from "dids";

const seed = Buffer.from("c7943d32...d6edfcd","hex");
//32 random bytes
const did = new DID({
  provider: new Ed25519Provider(seed),
  resolver: KeyResolver.getResolver(),
});

(async () => {
  console.log(await did.authenticate());

  const ceramic = new CeramicClient(
    "https://ceramic-clay.3boxlabs.com");
  ceramic.setDID(did);

  const idx = new IDX({ ceramic, aliases: {} });
  const streamId = await idx.set("basicProfile", {
    name: "dotnetpro tester",
    description: "demoing IDX for dotnetpro",
    emoji: "<3",
  });
  console.log(streamId);
})();

/*
> did:key:z6MkenQ8XxAT7fRa1fFMkw31M7ydDHLfdyJsqzvmyV4K
    WZfj
> StreamID(k2t6wyfsu4pg1ai6537smnd8hu66z0xhfqt3buuimdz
    afig8dgrm3x750m2kpc)
$ npx idx index:get did:key:z6MkenQ8XxAT7fRa1fFMkw31M7
    ydDHLfdyJsqzvmyV4KWZfj basicProfile
> {
>   name: 'dotnetpro tester',
>   description: 'demoing IDX for dotnetpro',
>   emoji: '<3'
> }
*/
```

Pinning und Permastores

Um die Informationsblöcke des IPFS langfristig aufzubewahren und vorzuhalten, bedarf es weiterer Technologien, denn IPFS selbst gibt keinerlei Garantien über die Permanenz von Informationen ab. Node-Betreiber können bestimmte CIDs auf ihren Nodes „pinnen“ und damit vor der Garbage Collection des Protokolls schützen.

Dienste wie Pinata [5] sind darauf spezialisiert, Pinsets gegen geringe Gebühren auf ihren Nodes vorzuhalten. Wer große Pinsets selbst aufbewahren möchte, kann mithilfe von IPFS Cluster [6] ein eigenes Storage-Netzwerk aufbauen, das Inhalte auf mehrere Nodes repliziert.

Deutlich komplexer ist das Filecoin-Netzwerk [7], das mit seinem ausgeklügelten „Proof of Space Time“-Konsens sogenannte Storage Miner dazu incentiviert, Inhalte aus dem IPFS für beliebig lange Zeiträume kontrolliert dezentral aufzubewahren. Seit seinem Mainnet-Launch im August 2020 haben Storage Miner über acht Millionen Terabyte Plattenplatz [8] bereitgestellt und der Wert des Filecoin-Utility-Tokens hat sich zeitweise verzehnfacht.

Da die direkte Interaktion mit dem Filecoin-Netzwerk selbst technisch relativ anspruchsvoll ist, bieten Dienste wie Powergate [9] von Textile oder Fleeks Space Daemon [10] für Clients deutlich vereinfachte APIs an.

Web3-Datenbanken, Nutzerprofile und Identitäten

Viele Projekte entwickeln derzeit auf Grundlage von IPFS Protokolle und Bibliotheken, die Schlüsselfunktionen von Webapplikationen ins Dezentrale verlagern. Ein vielversprechender, aber nie zu völliger Stabilität gelangter Ansatz ist OrbitDB [11], das es mithilfe von Lamport-Clocks, IPFS-Keys, CRDT-versionierten Dokumenten und dem Pubsub-Protokoll aus libp2p ermöglicht, Event-Logs oder Key-Value-Stores zwischen beliebigen Clients zu replizieren.

Mithilfe von OrbitDB entwickelte das Team von 3Box Labs [12] 2019 die erste von Blockchains unabhängige, dezentrale Lösung für Nutzerprofile und auf asymmetrischen Schlüsselpaaren basierende Authentifizierung. Als klar wurde, dass ein konsensloses Netzwerk in der Praxis dafür nicht stabil genug ist, entwickelten sie ihren Ansatz weiter zum Ceramic Network [13], dessen Mainnet Anfang 2021 seinen Betrieb aufnahm. Ceramic nutzt eine Merkle-Proof-basierte Blockchain-Verankerung von mehreren Dokumenten, um sehr schnell den aktuellen Stand eines Dokuments aus seiner Änderungshistorie herzustellen, und garantiert dabei dessen Schema-Konformität.

Die Sicherheit und Dezentralität von Ceramic prädestiniert das Protokoll für den vermutlich wichtigsten aller Anwendungsfälle, den 3Box Labs selbst unter dem Namen IDX [14] umsetzt: die Ablage eines dezentralen, vertrauenswürdigen Nutzerprofils unter voller, selbstsouveräner Kontrolle des Nutzers (Listing 4).

Im Vergleich zu klassischen, namensadressierten Cloud-Filesystemen sind alle diese Protokolle reichlich komplex, verbrauchen je nach verwendeter Blockchain Energie, verursachen Transaktionsgebühren und führen zu einer spürbaren Latenz während der Content-Auflösung. Web3-Technologien

machen das Internet also demokratischer und sicherer, aber selten schneller. Andererseits entstehen durch die Netzwerk-Incentives völlig neue Möglichkeiten für ökonomische Teilhabe: Wer Pinning-Cluster, Filecoin- oder Ethereum-Nodes betreibt, kann damit gutes Geld verdienen, das vorher in die Taschen der etablierten Cloud-Giganten floss.

Die Herausforderungen für Web3-Technologien

Aus dem Aspekt der Dezentralisierung erwächst ein ethisches Problem des Web3. Seine unkompromittierbaren Protokolle ermöglichen es, unstoppbare Applikationen, Dokumente und Webseiten zu bauen. Es ist unmöglich, jemanden im Web3 davon abzuhalten, Inhalte zu publizieren.

Einerseits ist es natürlich zu begrüßen, dass kein Staat, Großkonzern oder lupenreiner Demokrat der Welt mehr in der Lage sein wird, bestimmte Inhalte im Netz zu löschen oder zu blockieren, indem er einfach den entsprechenden Server abschaltet oder Netzwerk-Traffic zensiert. Diesen Vorteil genießen im Umkehrschluss aber natürlich auch Urheber weniger wünschenswerter Inhalte wie Fake-Newsseiten oder kriminelle Marktplätze.

Bevor es also zur nächsten Revolution des Internets kommen kann, müssen noch grundlegende Fragen geklärt und technologische Hürden genommen werden. Der Grundstein ist gelegt und die wichtigsten Anwendungsmöglichkeiten stehen Entwicklern bereits heute offen. Das Web3 wird uns früher oder später in ein neues Zeitalter des Internets führen – eine Zukunft, in der jeder einen Wert zum Netz beitragen kann und dafür belohnt wird. ■

[1] Tagesschau.de, Zahl der Cyberangriffe auf Höchststand, www.dotnetpro.de/SL2112Web3_1
[2] ENS, https://ens.domains
[3] https://eth.link
[4] Fleek, http://fleek.co
[5] Pinata, https://pinata.cloud
[6] IPFS Cluster, https://cluster.ipfs.io
[7] Filecoin, https://filecoin.io
[8] Filfox, Filecoin explorer, Storage Power Distribution, https://filfox.info/en/
[9] Textile, Powergate, https://docs.textile.io/powergate
[10] Fleek, Space Daemon, https://fleek.co/space-daemon/
[11] OrbitDB, https://orbitdb.org
[12] 3Box Labs, https://3boxlabs.com
[13] Ceramic Network, https://ceramic.network
[14] IDX, https://idx.xyz

Stefan Adolf
ist Developer Ambassador bei Turbine Kreuzberg. Der Fullstack-Entwickler mit Schwerpunkt auf Applikationen, IoT und Integration ist Organisator und Speaker auf Konferenzen und agiert durch seine Expertise in dezentraler Technologie und Web3 als Tech Lead in Venture-Projekten.

dnpCode A2112Web3