# One Click Buying of Images

Marco Crisafulli, Daniel Steudler

# Elevator Pitch

Bildergalerie mit automatischer Pay per View von Bildern. Mit Microservices Architektur und Abrechnung via SmartContract und Blockchain

# Idee

Ursprünglich:

- User sucht sich das Bild in einer Gallerie aus. Mit einem Klick auf das Thumbnail wird ein Bezahlvorgang via Etherum ausgelöst und das mit dem Public Key des Käufers verschlüsselt.
- Dem Uploader wird Guthaben gutgeschrieben

# User Stories

- Ich als Benutzer möchte auf ein Thumbnail klicken um das Bild zu öffnen.
- Ich als System möchte beim öffnen eines Bildes ein Fixbetrag an den Bildurheber überwiesen, aus dem Wallet des Benutzers.
- Ich als Bildurheber möchte ein Bild zur verfügung stellen und für Views bezahlt werden.

- Ich als System möchte überprüfen ob der Benutzer noch genügend Tokens hat.

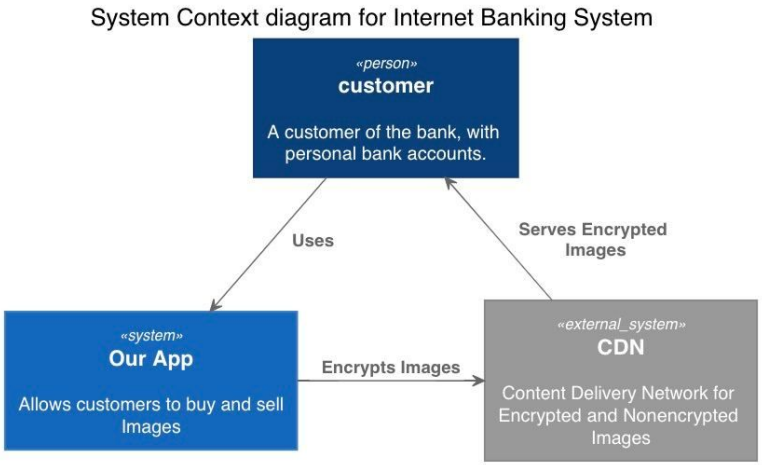
- Als Bildurheber möchte ich Einen Ledger mit all meinen Assets.
- Als Benutzer möchte ich einen Ledger mit Guthaben Tokens mit denen ich Bilder Views kaufen kann.

- Als System möchte ich bei einer Anfrage in einer Datenbank herausfinden wer der Urheber ist und Ihm einen Betrag in Tokens überweisen (welche der Anfrager übermittelt), dafür bekomme einen Link auf das Bild zurück welchen ich dem Anfrager präsentieren kann.

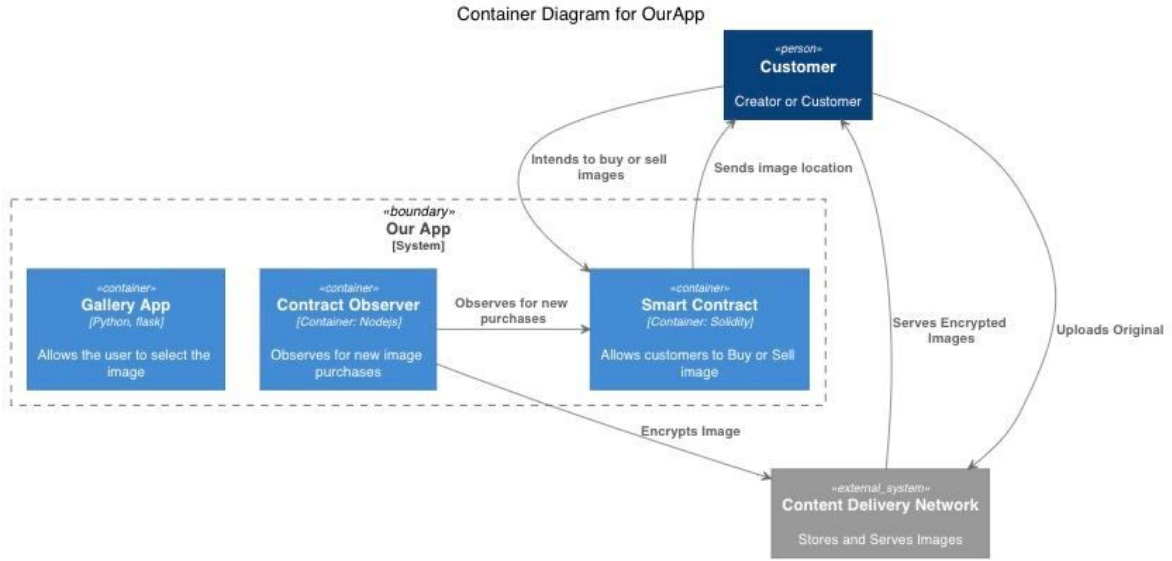
# Technische Ambitionen

- Wir wollten eine Microservices Architektur
- Möglichst Skalierbar
- Unterschiedliche Sprachen
  - Python
  - JS

# System Context

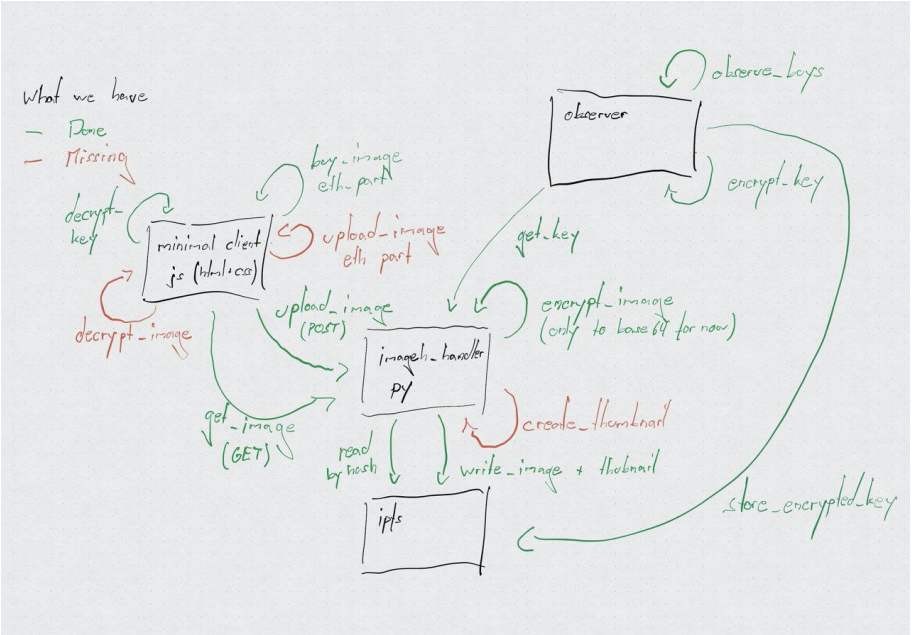


# System Component



# Was hat sich geändert seit dem Initialen Konzept

- IPFS als CDN
- Bild wird Symetrisch verschlüsselt (im POC nicht verschlüsselt sondern nur base64 codiert)
- Dem Käufer wird nur der Key und die Location des Bildes mitgeteilt. Der Symetrische key soll mit dem Publickey des Käufers verschlüsselt werden.



# Docker Aufbau

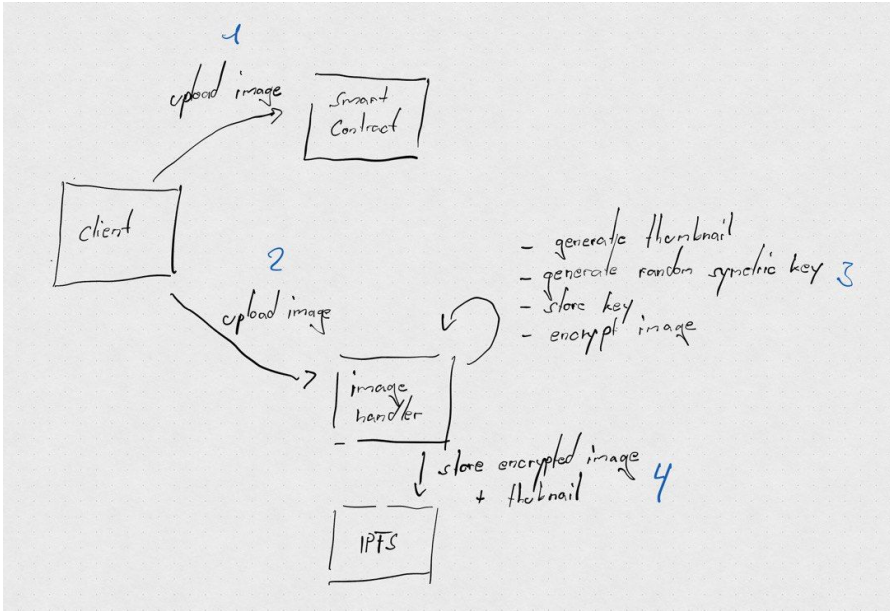
- Docker Container
  - Webclient
    - Bilder kaufen und verkaufen
    - Bilder entschlüsseln und Darsteller
    - Javascript
  - Observer
    - Beobachtet Chain
    - Javascript
  - Image Handler
    - Bilder Upload
    - Bilder Verschlüsseln
    - Python
  - IPFS
    - Bilder Storage
    - Key Storage

# CDN Aufbau

- /images/< seller wallet >/image
  - Das symmetrisch verschlüsselte Bild wird hier gespeichert
- /keys/< buyer public key>/< image hash >
  - Der asymmetrisch verschlüsselte Key wird hier gespeichert
- /thumbnails/< image hash >
  - Ein unverschlüsseltes Thumbnail des Bildes wird hier gespeichert

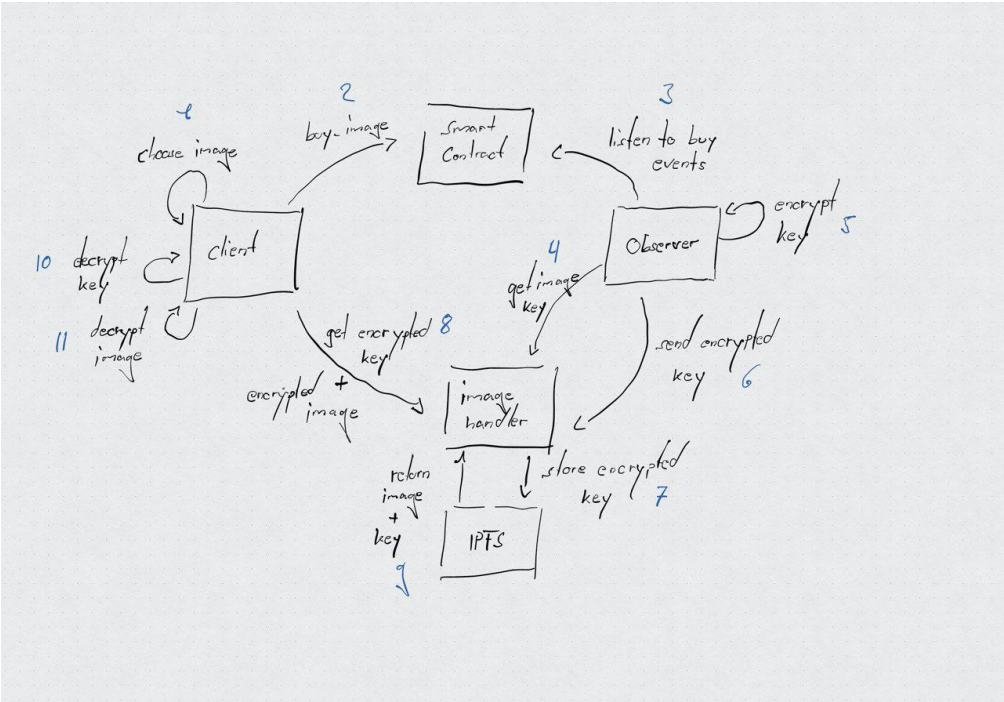
# How it works: Seller

1. Verkäufer lädt Bild hoch
2. Bild wird im CDN abgelegt und eine Eindeutige ID für das Bild erstellt
3. Das Bild wird dem Wallet des Besitzers zugeschrieben



# How it works: Buyer

1. Ein Bild wird ausgewählt
2. Ein Kauf des Bildes löst den Smart Contract aus
3. Der Observer beobachtet die Transaktionen des Contracts
4. Der Observer verschlüsselt den symmetrischen Key und speichert den im CDN
5. Der User kann den Key im Browser mit seinem Private Key entschlüsseln
6. Der User kann das Bild mit dem symmetrischen Key im Browser entschlüsseln



# Smart Contract

Methoden

- buyImage
- uploadImage

Felder

- Image
- Amount -> Price of the Image
- Buyer (Public Key)
- Images (Map aller bilder die es gibt, Key = Imagehash, Value = Image)

Events

- ImageBuy (Welcher Pubkey hat welchen ImageHash gekauft)

# Contract ImageStore

```
contract ImageStore{
    address payable private owner;
    uint32 _amount = 1 wei;
    mapping(string=>Image) public images;
    struct Image{
        address payable creatorAddress;
        //value=publicKey
        mapping(address=>Buyer) buyers;
    }

    struct Buyer{
        string publicKey;
        bytes32 imageLocation;
    }

    event ImageBuy(string _imageHash, string _publicKey);
```

# Function uploadImage

```
function uploadImage(string memory imageHash, address payable creatorAddress) public {
        require (images[imageHash].creatorAddress == address(0), "image already exists");
        images[imageHash] = Image(creatorAddress);
    }
```

# Function buyImage

```
function buyImage(string memory imageHash, string memory publicKey) public payable {
    require (images[imageHash].creatorAddress != address(0), "image must exist to buy");
    //require (msg.sender.balance <= _amount, "must have sufficent funds");
    images[imageHash].buyers[msg.sender].publicKey = publicKey;
    bytes memory tempImageLocation = concat(bytes(imageHash),
bytes(images[imageHash].buyers[msg.sender].publicKey));
    images[imageHash].buyers[msg.sender].imageLocation = keccak256(tempImageLocation);
    images[imageHash].creatorAddress.transfer(_amount);
    emit ImageBuy(imageHash, publicKey);
}
```

# Stolpersteine und Tradeoffs

- Vue -> HTML, CSS, Vanilla JS
- Drop von MetaMask und eigenem ETH Node
- Ausweichen auf Symetrisches Verschlüsselungs Verfahren
- Verschlüsselung mit AES hat uns Kopfzerbrechen bereitet
  - Verschlüsseln mit Python
  - Entschlüsseln mit JS
- Upload Image Smart Contract Methode funktioniert aus unbekannten Gründen nicht (aus Client, aus Remix funktioniert)
- Speichern von Symetrischen Keys irgendwo (private IPFS, redis, etc)

# Tool / Lib Liste

JS

- Web3 (als eth Client)
- Infura (als eth Provider)
- Eth-crypto (für Ent-/Verschlüsselungs asymmetrisch)
- Axios (als REST Client)

Python

- Fastapi (als REST Endpoint)
- Pycryptodome (für Ent-/Verschlüsselung symmetrisch)
- Pillow (für Bilder)
- Ipfshttpclient (als IPFS Client)

# DEMO

# Ausblick

- Scale it
  - Frontend Skaliert wunderbar
  - Observer aufteilen in Observer und Actors (Actors Pattern)
  - Image Handler ist Stateless
- Thumbnail erzeugen
- Interne DHT für Symmetric Key
- Gallerie Frontend