**Upload and use IPFS data (token URI)**

**Using IPFS**

In a [**previous lesson**](https://updraft.cyfrin.io/courses/advanced-foundry/how-to-create-an-NFT-collection/foundry-setup) we discussed tokenUris and I walked you through an example of viewing the TokenURI of a token on OpenSea.

{

"attributes": [

{

"trait\_type": "Background",

"value": "Mint"

},

{

"trait\_type": "Skin",

"value": "Dark Gray"

},

{

"trait\_type": "Body",

"value": "Turtleneck Pink"

},

{

"trait\_type": "Face",

"value": "Blushing"

},

{

"trait\_type": "Head",

"value": "Headband"

}

],

"description": "A collection 8888 Cute Chubby Pudgy Penguins sliding around on the freezing ETH blockchain.",

"image": "ipfs://QmNf1UsmdGaMbpatQ6toXSkzDpizaGmC9zfunCyoz1enD5/penguin/420.png",

"name": "Pudgy Penguin #420"

}

❗ **NOTE** Notice how the image property has *its own* IPFS hash! This is storing what the NFT looks like!

When this course was originally filmed, the Pudgy Penguins collection had been using IPFS's Gateway to access their images within the TokenURI

"https://ipfs.io/ipfs/QmNf1UsmdGaMbpatQ6toXSkzDpizaGmC9zfunCyoz1enD5/penguin/420.png";

This works, and is often leveraged due to browser compatibility with IPFS, but it's worth noting that this is pointing to a centralized server. If that server goes down, the image data will be unretrievable via the tokenURI call!

A more decentralized way to retrieve the image data is by pointing to the IPFS network itself.

"ipfs://QmNf1UsmdGaMbpatQ6toXSkzDpizaGmC9zfunCyoz1enD5/penguin/420.png";

**Doggies**

With a better understanding of IPFS and decentralized storage in hand, let get back to our BasicNFT contract. If you want, you can take and image I've provided (or your own) and upload it to IPFS to acquire your own hash. Alternatively, if you want to make things easy on yourself:

"ipfs://bafybeig37ioir76s7mg5oobetncojcm3c3hxasyd4rvid4jqhy4gkaheg4/?filename=0-PUG.json";

If you view this in your browser or through the IPFS Desktop App, you should see the tokenURI of our Doggie NFT, it's even got an image assigned already.

❗ **PROTIP** If you do decide to upload your own data to IPFS, you'll need to upload your image first to acquire an imageURI/hash. You'll then upload a tokenURI json containing this pointer to your image.

{

"name": "PUG",

"description": "An adorable PUG pup!",

"image": "https://ipfs.io/ipfs/QmSsYRx3LpDAb1GZQm7zZ1AuHZjfbPkD6J7s9r41xu1mf8?filename=pug.png",

"attributes": [

{

"trait\_type": "cuteness",

"value": 100

}

]

}

Now, we could just paste the about tokenURI as a return value of our tokenUri function, but this would mint every Doggie identical to each other. Let's spice things up a little bit and allow the user to choose what their NFT looks like. We'll do this by allowing the user to pass a tokenUri to the mint function and mapping this URI to their minted tokenId.

contract BasicNFT is ERC721 {

uint256 private s\_tokenCounter;

mapping(uint256 => string) private s\_tokenIdToUri;

constructor() ERC721("BasicNFT", "BFT") {

s\_tokenCounter = 0;

}

function mintNFT(string memory tokenUri) public returns (uint256) {

s\_tokenIdToUri[s\_tokenCounter] = tokenUri;

}

function tokenURI(

uint256 tokenId

) public view override returns (string memory) {

return s\_tokenIdToUri[tokenId];

}

}

Great! All that's missing is to mint the NFT and increment our token counter. We can mint the token by calling the inherited \_safeMint function.

contract BasicNFT is ERC721 {

uint256 private s\_tokenCounter;

mapping(uint256 => string) private s\_tokenIdToUri;

constructor() ERC721("BasicNFT", "BFT") {

s\_tokenCounter = 0;

}

function mintNFT(string memory tokenUri) public returns (uint256) {

s\_tokenIdToUri[s\_tokenCounter] = tokenUri;

\_safeMint(msg.sender, s\_tokenCounter);

s\_tokenCounter++;

}

function tokenURI(

uint256 tokenId

) public view override returns (string memory) {

return s\_tokenIdToUri[tokenId];

}

}

**Wrap Up**

Wow, our "BasicNFT" contract looks awesome! It's actually even a little more advanced than you'd think because it allows a user to mint an NFT that looks like .. anything they want!

I challenge you to try deploying this to a testnet and experimenting with minting your own NFTs. Become familiar with hashing your data in IPFS and have fun with customizing your NFT properties.

When you're done, or if you're having trouble, I'll see you in the next lesson to discuss deploying!