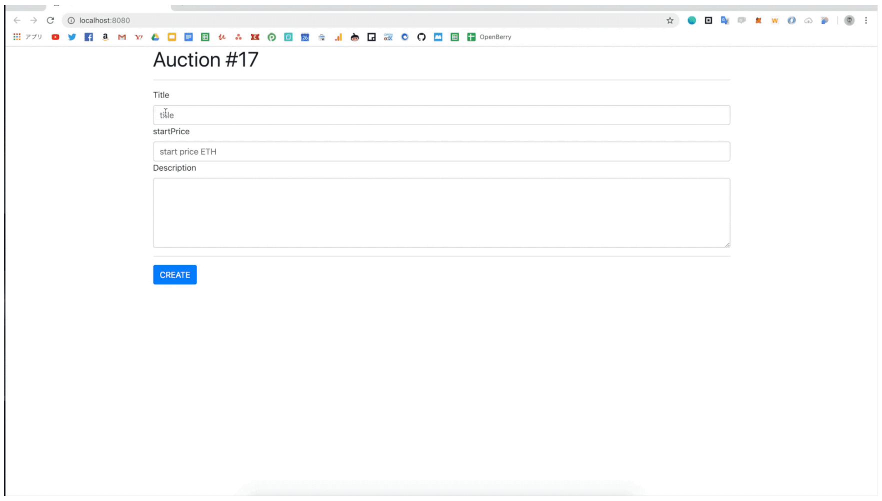
Ethereum Solidity + Vue.js Tutorial Simple Auction Dapp within 10 minutes

Hayata Satomi Follow
Feb 26 · 6 min read

You might have heard of auctions, a place where you can buy and sell things. Although it is very convenient, it costs sellers about 10% of their earnings to pay the service charge to the auction's managing company.

If the company itself cheats some transactions and get the money, how will you know that you have been cheated?

This is where a decentralized setup is an optimal solution.
If there is no third party, sellers can earn more, in a secure way.



Function List

1. Creating an auction
2. Placing a bid
3. Finalizing an auction

Tools

1. Smart Contract
   Solidity, Remix, Metamask
2. Frontend
   Web3.js, Vue.js, Vue-cli, Boostrap-vue

Prerequisites

1. Getting Started with MetaMask
2. Compile and Deploy Using Remix IDE
3. Introduction to Smart Contracts and Solidity

Github

If you only want to see the code, you can find it here:
https://github.com/openberry-ac/Auction

Why Build an Auction Dapp?

The auction system is a good application to build for the purpose of learning about smart contracts and decentralized applications, especially for beginners.

Smart contracts will help you exchange money, properties, shares, or anything of value in a transparent, conflict-free way while avoiding the services of a middleman, like the auction system.

Making the Project

Work flow

1. Creating the Smart Contract
2. Building web app & Web3.js setting
3. Defining the Methods: Frontend Coding

Creating the Smart Contract

Since our contract will be based on Ethereum, we will use Solidity, a programming language used for creating smart contracts.

In Remix, create a new file named **AuctionBox.sol** and add the following code:

```
// We will be using Solidity version 0.5.3
pragma solidity 0.5.3;
// Importing OpenZeppelin's SafeMath Implementation
import "https://github.com/OpenZeppelin/openzeppelin-

contract AuctionBox{

    Auction[] public auctions;

    function createAuction (
        string memory _title,
        uint _startPrice,
        string memory _description
        ) public{
        // set the new instance
        Auction newAuction = new Auction(msg.sender,
        // push the auction address to auctions array
        auctions.push(newAuction);
    }

    function returnAllAuctions() public view returns(
        return auctions;
    }
}

contract Auction {

    using SafeMath for uint256;

    address payable private owner;
    string title;
    uint startPrice;
    string description;

    enum State{Default, Running, Finalized}
    State public auctionState;

    uint public highestPrice;
    address payable public highestBidder;
    mapping(address => uint) public bids;

    /** @dev constructor to creat an auction
      * @param _owner who call createAuction() in Auc
      * @param _title the title of the auction
      * @param _startPrice the start price of the auc
      * @param _description the description of the au
      */

    constructor(
        address payable _owner,
        string memory _title,
        uint _startPrice,
        string memory _description

        ) public {
        // initialize auction
        owner = _owner;
        title = _title;
        startPrice = _startPrice;
        description = _description;
        auctionState = State.Running;
    }

    modifier notOwner(){
        require(msg.sender != owner);
        _;
    }

    /** @dev Function to place a bid
      * @return true
      */

    function placeBid() public payable notOwner retur
        require(auctionState == State.Running);
        require(msg.value > 0);
        // update the current bid
        // uint currentBid = bids[msg.sender] + msg.v
        uint currentBid = bids[msg.sender].add(msg.va
        require(currentBid > highestPrice);
        // set the currentBid links with msg.sender
        bids[msg.sender] = currentBid;
        // update the highest price
        highestPrice = currentBid;
```

```
            highestBidder = msg.sender;

            return true;
        }
```

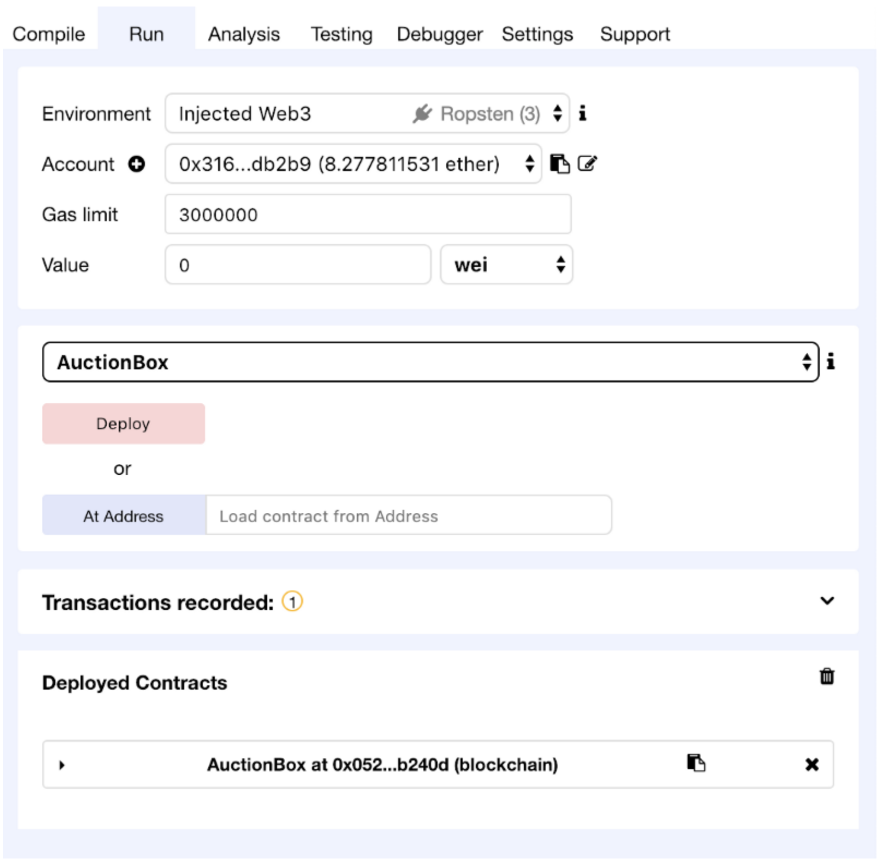
In an actual setting, many items can be auctioned. In order to make it easy to access all the auctioned items, we created a bundle or box called “AuctionBox” which includes all the auction addresses!

To make this system work, we need to prepare two types of contracts, called Auction contract and AuctionBox contract.

After writing it in Remix, deploy it to the **Ropsten Test Network.**

Let’s check it out.

To verify that our contract was deployed, you should see this in Remix: Here, we need to select **AuctionBox** contract.



Environment: Injected Web3 — Ropsten (3)
Account: 0x316...db2b9 (8.277811531 ether)
Gas limit: 3000000
Value: 0 wei

AuctionBox

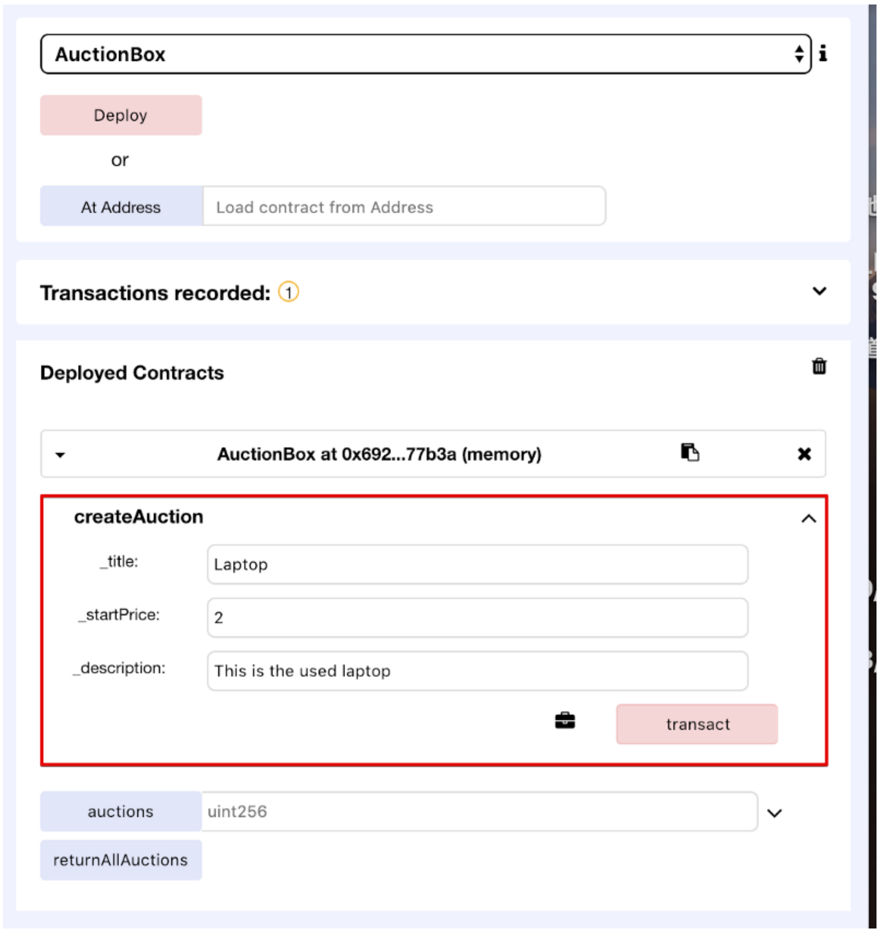
Transactions recorded: 1

Deployed Contracts

AuctionBox at 0x052...b240d (blockchain)

Let's create an auction !

After deploying, let's try to make an auction using **createAuction** method.



_title:	Laptop
_startPrice:	2
_description:	This is the used laptop

If you success, you can click **returnAllAuctions** and see its contract address!

Building the Web App

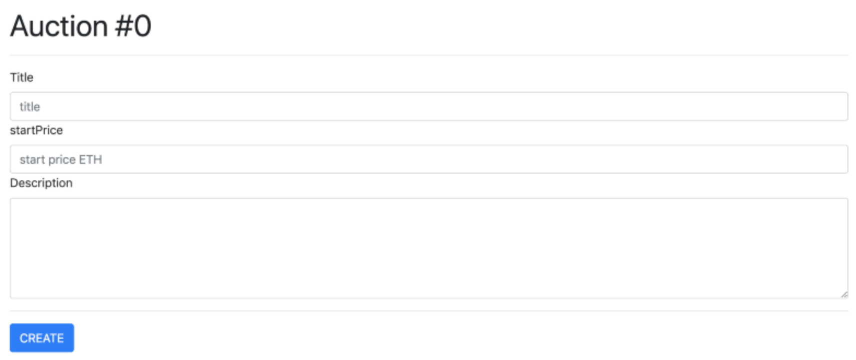
Our smart contract now works, but there's no fun in just looking at numbers so we'll be making a simple web application.

Setting Up

To speed up things, a template project is provided which can be found here. Now let's follow the commands below, in your Terminal (or Command Prompt/Powershell for Windows):

```
# git clone the project template
git clone -b boilerplate --single-branch
https://github.com/openberry-ac/Auction.git

# go inside the folder
cd Auction

# install packages needed in the web application
npm install

# install web3, this is for connecting the contract
npm install -s web3@1.0.0-beta.37

# To run the app
npm run dev
```

Then it should automatically render on http://localhost:8080/



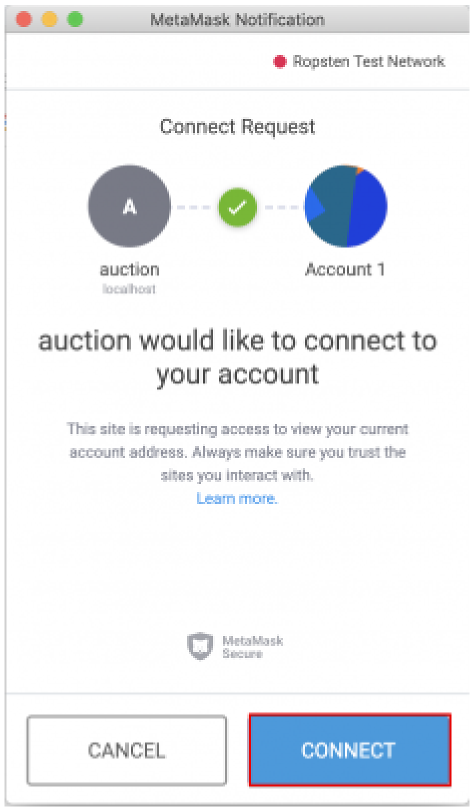
Setting up web3.js

Open the file named **web3.js** in the **contracts** folder, then paste it as following code:

```
import Web3 from 'web3';

if (window.ethereum) {
  window.web3 = new Web3(ethereum);
  try {
    // Request account access if needed
    ethereum.enable();
  } catch (error) {
    // User denied account access...
  }
} else if (window.web3) { // Legacy dapp browsers...
  window.web3 = new Web3(web3.currentProvider);
```

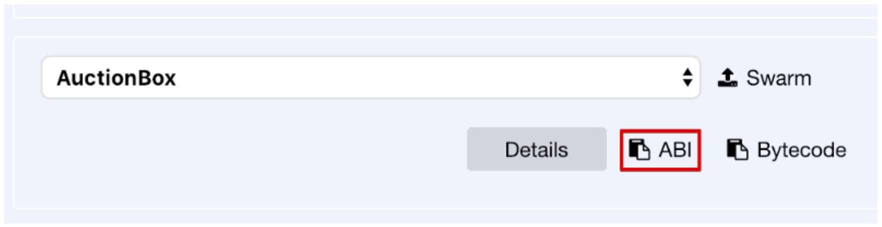
This basically gets the `web3` instance that the Metamask extension initializes, so we can use it in our app too. We need to call this later when retrieving our smart contract instance.

You might encounter a MetaMask pop-up window in your browser that asks for access permission. You should just click the 'Connect' button right here:

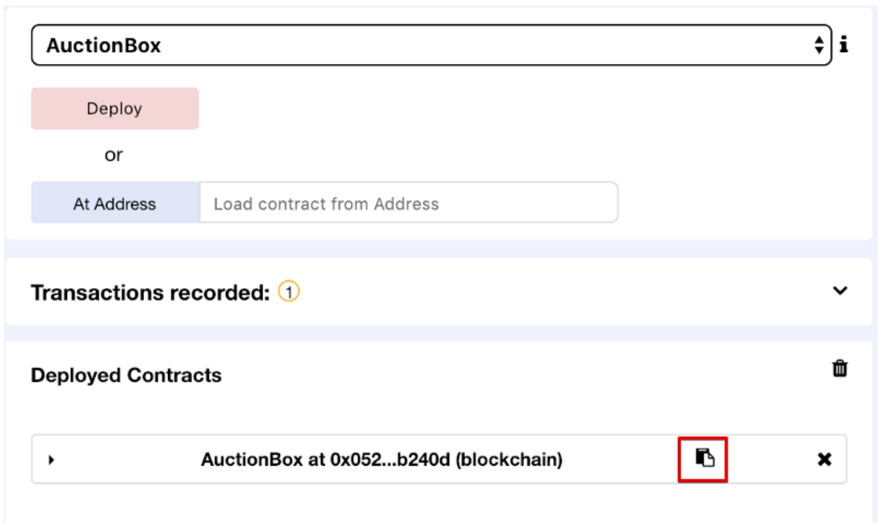

MetaMask Notification
Ropsten Test Network
Connect Request
A
auction
localhost
Account 1
auction would like to connect to your account
This site is requesting access to view your current account address. Always make sure you trust the sites you interact with.
Learn more.
MetaMask Secure
CANCEL
CONNECT

Connecting to Our Smart Contract Instance

Now, we need our smart contract's ABI and the contract address to connect it to our web app. To get the ABI, go back to Remix, go to the **Compile** tab, and click **ABI** beside the Details button as shown in the picture:



To get the contract address, go to the **Run** tab, and click **Copy** button as shown in the picture:



You can also get **Auction** contact's **ABI** by changing the selected contract to Auction and clicking ABI.

After getting it, open the files named **AuctionBoxInstance.js** and **AuctionInstance.js** in the **contracts** folder, then paste it as the variable **abi** 's value and the contract address, like this:

```
import web3 from './web3';

const address = ''// THE CONTRACT ADDRESS
const abi = []// THE ABI

const instance = new web3.eth.Contract(abi, address);
```

```
import web3 from './web3';

const abi = []// THE ABI
// Here is just only abi because we haven't created auc
export default (address) => {
  const instance = new web3.eth.Contract(abi, address);
```

Defining the Methods

You might notice that the user interface is there, but the buttons aren't functional. That's because we have not defined our functions yet, which we will be doing now. Open **App.vue** in the src folder and let's get started to fill in the blanks!

beforeMount

Here, we will get the number of auctions that you created before.

```
beforeMount() {
  // get auctionBox method: returnAllAuctions()
  auctionBox.methods
    .returnAllAuctions()
    .call()
    .then((auctions) => {
      console.log(auctions);
      // set the amount of auctions
```

Create Auction Function

Here, we create an auction using the user's inputs, which will be shown at the bottom of the page.

```
createAuction() {
  // get accounts
  web3.eth.getAccounts().then((accounts) => {
    // convert 'ether' to 'wei'
    const startPrice = web3.utils.toWei(this.startPric
    // createAuction in AuctionBox contract
    this.isLoad = true;
    return auctionBox.methods.createAuction(this.title
      .send({ from: accounts[0] });
  }).then(() => {
    // initialize forms
    this.isLoad = false;
    this.title = '';
    this.startPrice = '';
    this.description = '';
    // get the previous auction
    return auctionBox.methods.returnAllAuctions().call
  }).then((auctions) => {
    const index = auctions.length - 1;
    console.log(auctions[index]);
    // get the contract address of the previous auctic
    this.auctionAddress = auctions[index];
    // set the address as the parameter
    const auctionInstance = auction(auctions[index]);
    return auctionInstance.methods.returnContents().ca
  })
```

Place Bid Function

Here, we handle the event wherein the user places a bid.

```
handleSubmit() {
  // convert 'ether' to 'wei'
  const bidPriceWei = web3.utils.toWei(this.bidPrice,
  // get the wallet adddress
  const fromAddress = web3.eth.accounts.givenProvider.
  // set the address as the parameter
  const selectedAuction = auction(this.auctionAddress)
  this.isBid = true;
  // placeBid in Auction contract
  selectedAuction.methods
    .placeBid()
    .send({
      from: fromAddress,
      value: bidPriceWei,
    })
```

Finalize Auction Function

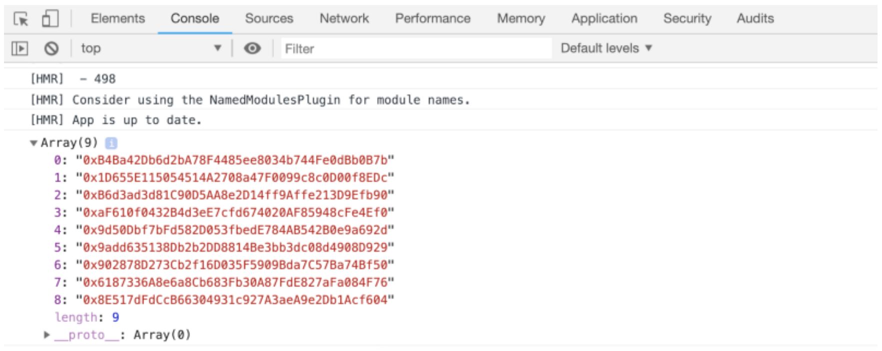
Here, we handle the event wherein the user finalizes an auction.

```
handleFinalize() {
  // get accounts
  web3.eth.getAccounts().then((accounts) => {
    // set the address as the parameter
    const selectedAuction = auction(this.auctionAddres
    this.isFin = true;
    // finalizeAuction in Auction contract
    selectedAuction.methods
      .finalizeAuction()
      .send({ from: accounts[0] })
      .then(() => {
```

✓ Let's check it out.

We have already created an auction in Remix, you can see its contract address in console.

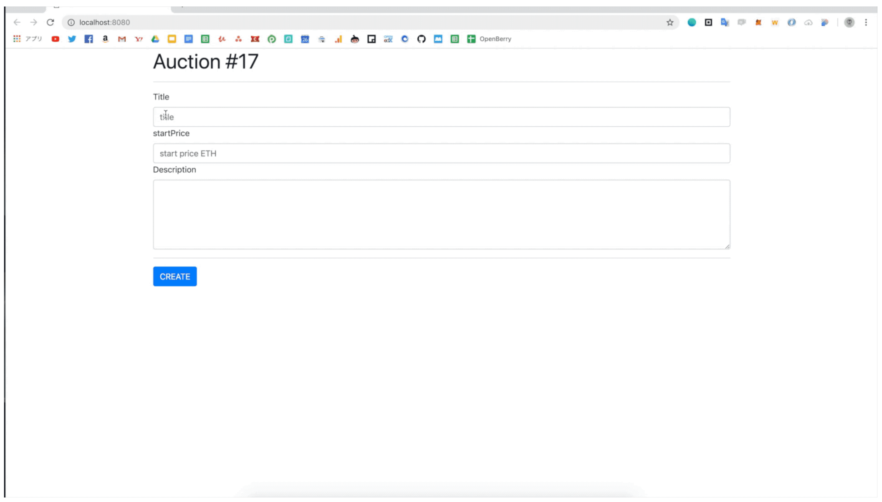
In my case, I created 9 auctions, so I can see this in console:



```
[HMR]  - 498
[HMR] Consider using the NamedModulesPlugin for module names.
[HMR] App is up to date.
▼Array(9)
   0: "0xB4Ba42Db6d2bA78F4485ee8034b744Fe0dBb0B7b"
   1: "0x1D655E115054514A2708a47F0099c8c0D00f8EDc"
   2: "0xB6d3ad3d81C90D5AA8e2D14ff9Affe213D9Efb90"
   3: "0xaF610f0432B4d3eE7cfd674020AF85948cFe4Ef0"
   4: "0x9d50Dbf7bFd582D053fbedE784AB542B0e9a692d"
   5: "0x9add635138Db2b2DD8814Be3bb3dc08d4908D929"
   6: "0x902878D273Cb2f16D035F5909Bda7C57Ba74Bf50"
   7: "0x6187336A8e6a8Cb683Fb30A87FdE827aFa084F76"
   8: "0x8E517dFdCcB66304931c927A3aeA9e2Db1Acf604"
   length: 9
  ▶__proto__: Array(0)
```

Complete!!

Refresh your browser to see the changes. This time, the whole web app is complete, and everything is functional! You should then be able to use it like this:



NOTE: The one who made the action cannot place a bid in his own auction. So, you need to switch to another account (other than the auction starter) to place a bid.

NOTE: To display the auction card, please create another auction in your browser.

Summary

You learned how to create a smart contract and how to interact with it using web3.js. You also learned how to setup your own project using Vue.js, and created a simple application.

So what's next?

You might want to add the logic for the "deadline" because it is important in a real auction application. For now, we skipped it to avoid complexity and to lessen the time, but if you want to try to build it, this will help you a lot :)

If you want the full code of this tutorial, you can check it here: https://github.com/openberry-ac/Auction

On a side note, you might want to check out openberry's previous tutorial, **"ERC721 + Vue.js CryptoKitties-like Dapp in under 10 minutes"**.

ERC721 + Vue.js CryptoKitties-like Dapp in under 10 minutes

You might have heard of CryptoKitties, an Ethereum-based platform where you can collect…

medium.com

openberry is a tutorial marketplace, designed to allow anyone to learn blockchain programming.

openberry | blockchain tutorial marktplace

openberry is a tutorial marketplace, designed to enable anyone to learn blockchain programming.

www.openberry.ac

openberry (@openberry_ac) | Twitter

The latest Tweets from openberry (@openberry_ac). openberry is a tutorial…

twitter.com