**LAB 5: Contract Deployment**

**Overview**

The purpose of this lab is to guide you through all installation and compilation process required to deploy our lottery contract to actual Rinkeby Test Network.

**Prerequisites**

This lab will use the GENI environment that you set up. If you haven’t set up the environment, please refer to the document on setting up environment.

You should have completed Lab 5 – Contract Deployment, received address of the lottery contract you deployed.

**Part 1 – Install Google Chrome**

Step 1: On the vPark VNC environment, right click the desktop and select “Open Terminal Here”.

Step 2: In the terminal, type in command

Download Google Chrome:

wget https://dl.google.com/linux/direct/google-chrome-stable\_current\_amd64.deb

Install Google Chrome:

sudo apt install ./google-chrome-stable\_current\_amd64.deb

Starting Google Chrome:

google-chrome&

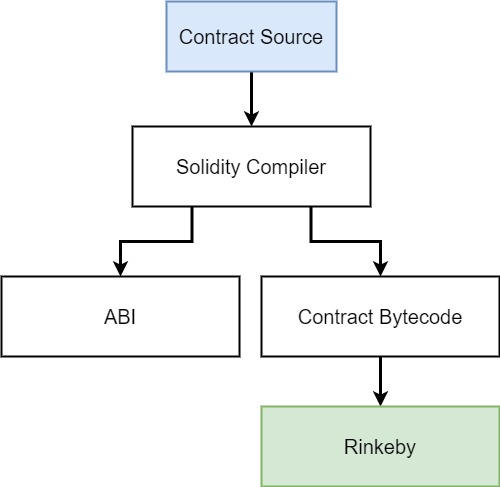
**Part 2- Install Meta Mask**

Refer to Lab 1- Metamask Setup document.

**Contract Deployment Requirement**

At present we've now got a pretty well-working lottery contract that exists inside of the remix editor online. Now it's fun to play around with the integrated JavaScript VM here in Remix. However, at some point in time we're probably going to want to take this contract and actually deploy it out to either the network or the main network so we can actually use this contract with real projects that we want to build. So, it's not enough just to edit this thing inside remix.

We need to figure out how we can actually deploy the contract. So that's the real issue that we need to solve. Now we've got Remix. Here's our contract source but how do we actually deploy it for deployment. There's a variety of different options. Remember at the core of whatever deployment process we end up using it's always going to go through a very similar process. We're going to take our contract source. That source is going to be placed into the solidity compiler. Just so you know this compiler is running inside the remix tool right now. The solidity compiler is going to split out both an API and the contract bytecode and this bytecode right here is what actually ends up getting deployed to some network. So clearly. All right Remix maybe does like half the job.



But how do we actually get this code or that deployed or that bytecode over to our network that we want to deploy to for that. There is a very common tool that you're going to see mentioned and a tremendous amount of documentation online in blog posts and all that kind of good stuff and that tool is called Truffle. Truffle is kind of a one stop shop for development of theory in contracts. This Truffle project or the Truffle CLI is a command line tool that we can use to aid in contract creation. So, like actually writing out source code and doing some local testing So programmatic automated testing to make sure that our contract behaves the way we want it is also useful for deployment as well. You're going to see a lot of mentions of truffle as soon as you start working with any type of theory and project and it's a very common tool to use on any professional project as well.

Well there's just one little issue here just one little issue. Here's the thing about Truffle, Truffle is still undergoing rapid development. Some things about truffle don't really work well like some functionality in this truffle tool is a little bit rough around the edges. Other things don't work at all. So, truffle has some features that you would really expect it to have but they don't actually work. They don't exist. They have not yet been implemented by the authors of the truffle library. And above all features inside a truffle sometimes break.

So, for this first project that you and I are going to put together in order to deploy our lottery contract code right here to the actual Rinkeby test network or to any network whatsoever rather than using truffle or any other fancy tools you and I are going to put together a custom node project from scratch that's going to aid us in contract creation in doing actual deployment to the Rinkeby Test network.

A screenshot of a cell phone

Description automatically generated

So, there are tools out there to automate this process and make it very easy to get set up and started. But I'm saying that hey sometimes they're not quite dependable. So, I want you to understand what these tools are doing behind the scenes. So, you and I are going to build our own custom boiler plate that is going to aid in all of these common tasks.

We're going to set it up all manually. So, you have a great idea of what these tools are actually doing. It's really going to be a lot of fun and it's going to give you a lot of insight into the purpose of truffle and web 3 and all these other libraries.

**Boilerplate Requirements**

**A screenshot of a cell phone

Description automatically generated**

**Part 3- Install Node.js**

The first thing that we need to be able to do inside of our boiler plate project that we're going to put together is to make sure that you and I can actually write some solidity code that can then be accessed in some fashion from the JavaScript side of our project. Remember everything we do in this course is going to be JavaScript themed. So, I want to be able to write some JavaScript code to deploy my contract and to eventually work with my contract after it has been deployed. It's all JavaScript. However, we are writing solidity contracts in order to get a bridge between the solidity side of our contract and then somehow interact with it from the JavaScript side. So, install Node.js so that we can run our JavaScript i.e. compile.js and deploy.js files.

Step 1: On the vPark VNC environment, right click the desktop and select “Open Terminal Here”.

Step 2: In the terminal, type in command

sudo apt install nodejs

Step 3: Once installed, verify it by checking the installed version using the following command:

node -version

**Part 4 – Install NPM**

Npm makes it easy for JavaScript developers to share and reuse code, makes it easy to update the code.

Step 1: In the terminal, type in command

sudo apt install npm

**Part 5 – Contract Compilation**

So we’re going to flip back over to our terminal. Create a new project directory and that's where we're going to place all the code that's going to work with our contract compile deploy all this stuff. So we are going to make a new directory and we'll call it lottery.

* In terminal, type in command mkdir lottery
* Change directory into lottery cd lottery

And then finally we will generate a new node project by running the command npm init . We'll see a couple of pop ups here ask us for some information. Right now there's really not any extra information we need to enter. So we’re going to enter a few times and eventually we end up with somebody who says Is this OK. And we get kicked back to the command line and now we should see a new package that is on file inside this new directory.

We generated a new package not just on file inside of a new lottery project directory. All right now before we start just diving in here and writing any code I want to give you a quick overview of some of the different project files and folders that we're going to create to organize our code.

A screenshot of a cell phone

Description automatically generated

Next inside of our root project directory is the **package.json** file. Like any other JavaScript project that you might have worked on over time, the package.json file will behave in the same way so we're going to create some number of scripts inside of it and it's also going to record different dependencies that we install into our project.

Next up is **compile.js** which will be a small script file whose sole purpose will be to look inside that contracts directory and compile each of the contracts that exist inside of it. So, this is going to be kind of a one of project file or a one of file that is used solely to compile our code.

And then finally we have **deploy.js** which is going to be a little script file whose sole purpose will be to take some compiled code and deploy it to a theory network that might be the main network or Ropsten or Rinkeby or even a private local network as well.

Let's get started by creating this contract's directory in the lottery. Inside lottery, create a new directory called contracts. You have to make sure you are inside of our lottery project directory. Type in command cd lottery. In terminal, type in command mkdir contracts

Change the directory to contract. Type in command cd contracts .Inside contract create lottery.sol . Type in command nano lottery.sol

Flip back over to remix where we have all our lottery source code. Copy all of it and flip back over to terminal and paste it in lottery.sol .

Once you paste it, perform the following things:

* **Ctrl + O to save.**
* **Enter**
* **Ctrl + X to exit.**

Now thinking about our project, itself we're going to start working very quickly on the compile.js file. The reason we're going to go directly to a compiled file is the deployment process is going to expect that we have a compiled contract ready to go. So in order to do any deployment we first have to solve this compilation step to do the actual compilation step inside of our project.

We're going to make access to the solidity compiler. Remember we're going to pass some contract source code into that compiler and that is going to spit out the API which is our JavaScript interpretation layer of what the contract is and also the contract byte code which is what we actually deploy off to the theory of network.

To install the solidity compiler, we're going to install a tiny little node module back at my terminal. I'm going to make sure that I'm inside of my project directory and then I'm going to install the solidity compiler as an NPM package.

Type in command cd – to go back to project directory. Type in npm install --save solc@0.4.17

Inside of the root project directory, create the compile.js file

nano compile.js

Copy the following code in compile.js .

const path = require('path');

const fs = require('fs');

const solc = require('solc');

const lotteryPath = path.resolve(\_dirname, 'contracts', 'Lottery.sol');

const source = fs.readFileSync(lotteryPath, 'utf8');

console.log(module.exports = solc.compile(source, 1).contracts[':Lottery']);

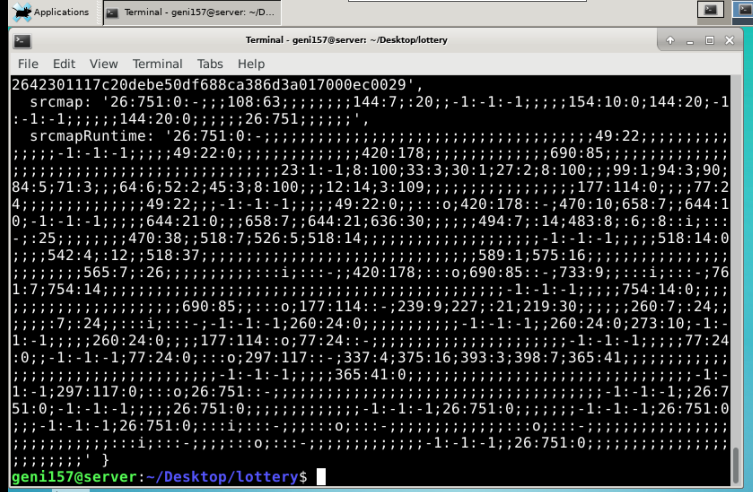
Once you paste the code, perform following actions:

* **Ctrl + O to save.**
* **Enter**
* **Ctrl + X to exit.**

In the terminal, inside of the lottery project directory, run the command:

**node compile.js**

If you solidity code is compiled correctly, you will see the screen like below:



**Part 6: Web3 Installation**

We're going to take that JavaScript interface or our API and we're going to feed that into Web3. Remember Web 3 is a library that we use to get programs at programmatic access to a deployed contract on the block chain. So, Web 3 right here is kind of our portal into what is going on the Ethereum network.

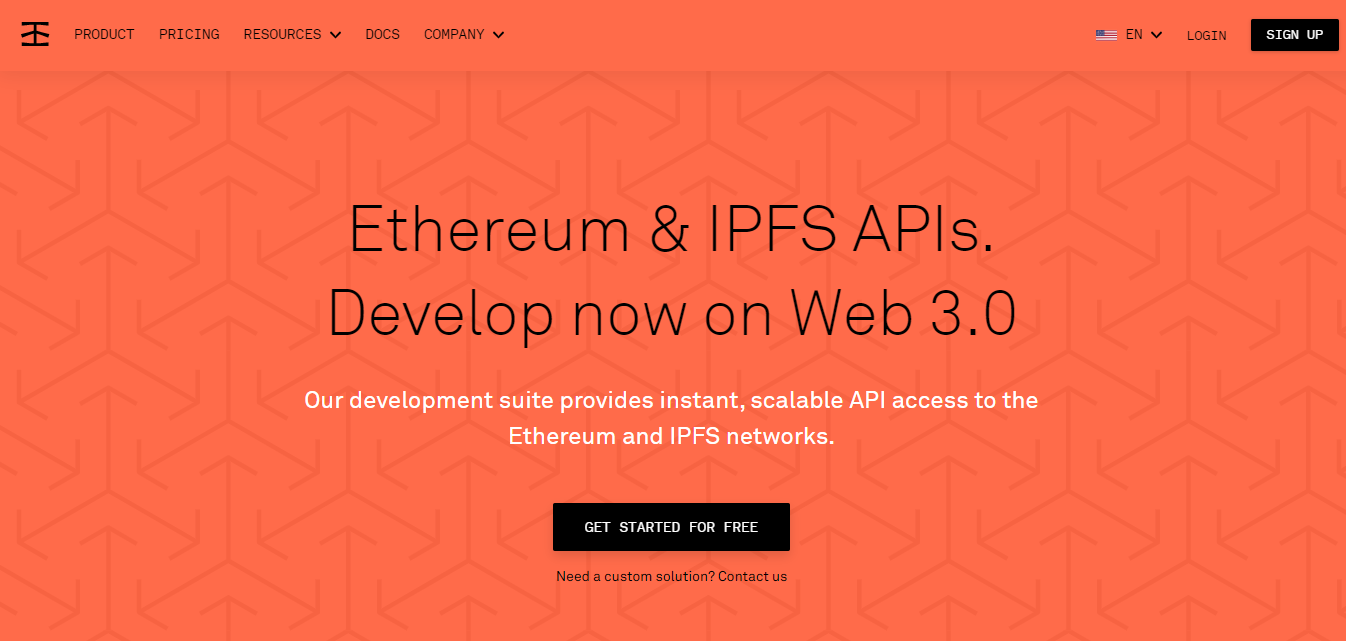
Inside of root project directory install Web3. Type in command:

npm install --save web3

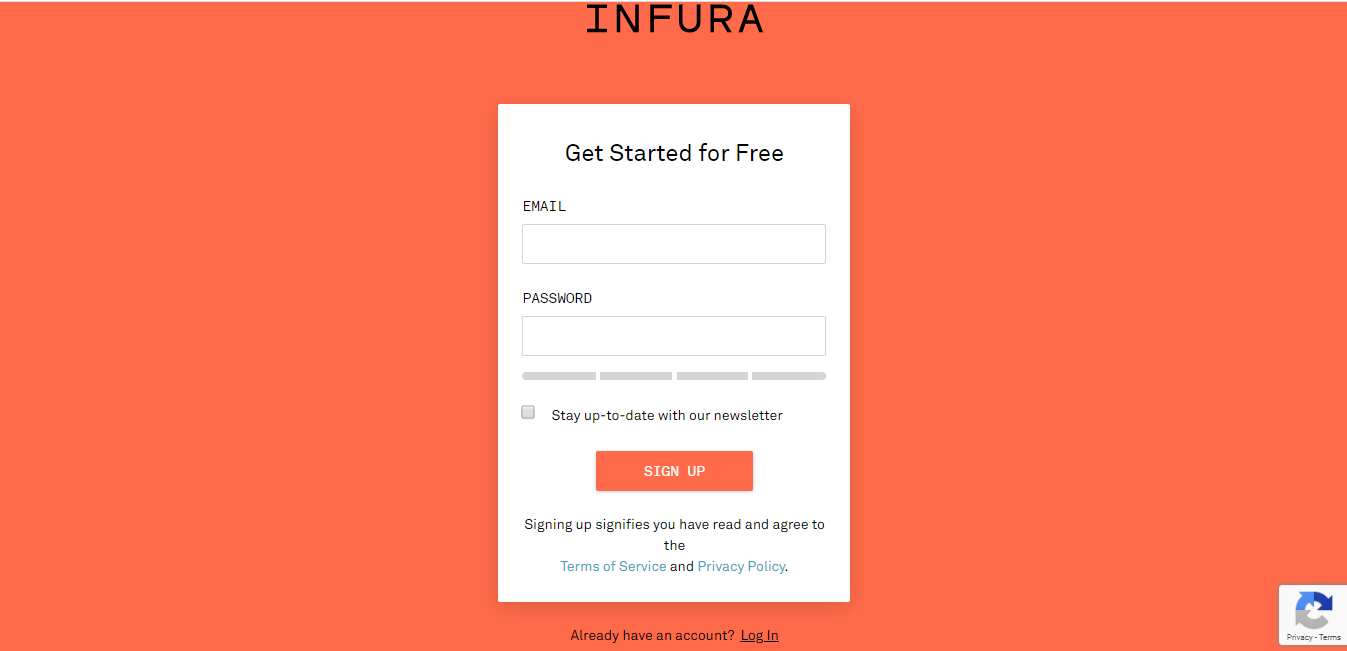
**Part 7: Infura Signup**

Infura provides Infrastructure as a service for Ethereum that lets your users run your application without requiring them to set up their own Ethereum node.

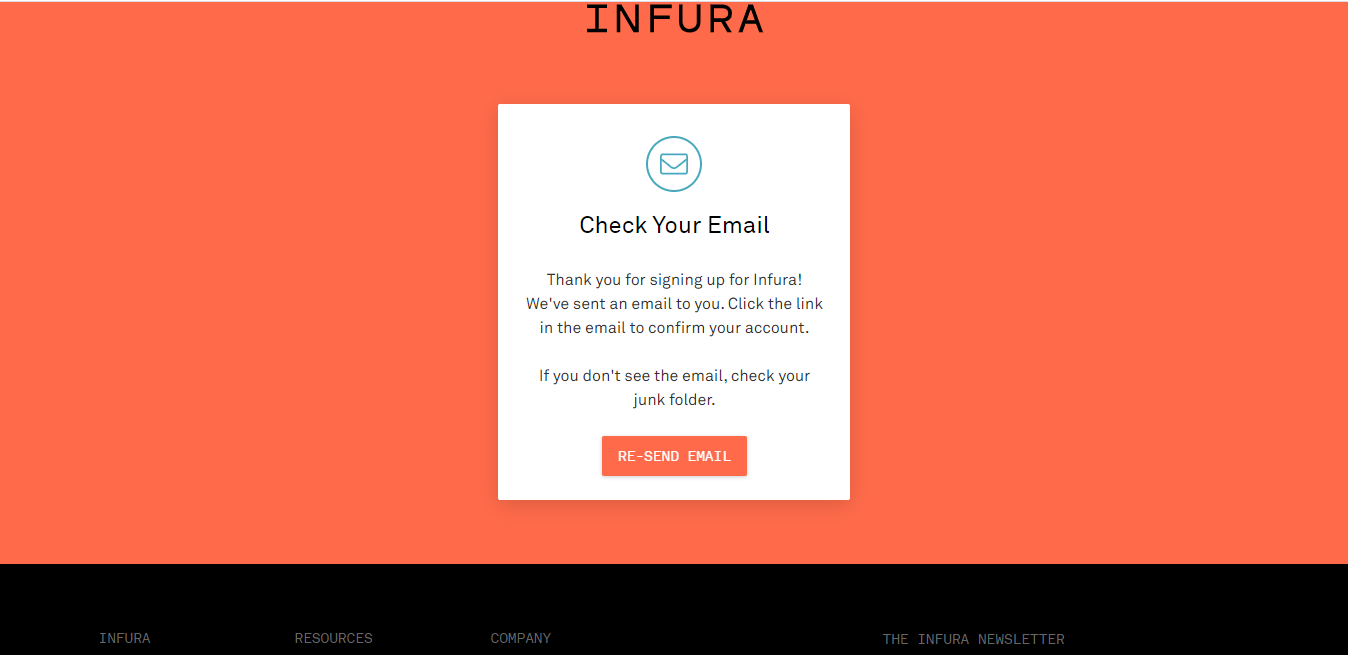
Go to the link **Infura.io**



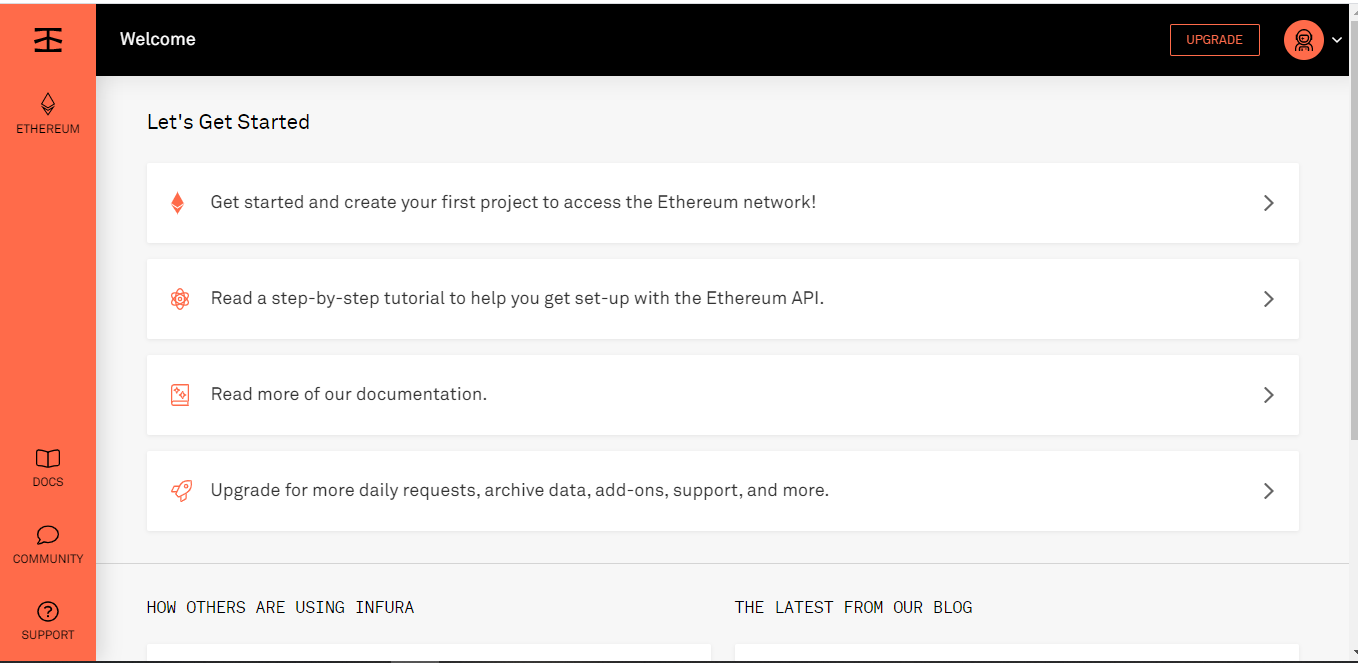
Click GET STARTED FOR FREE



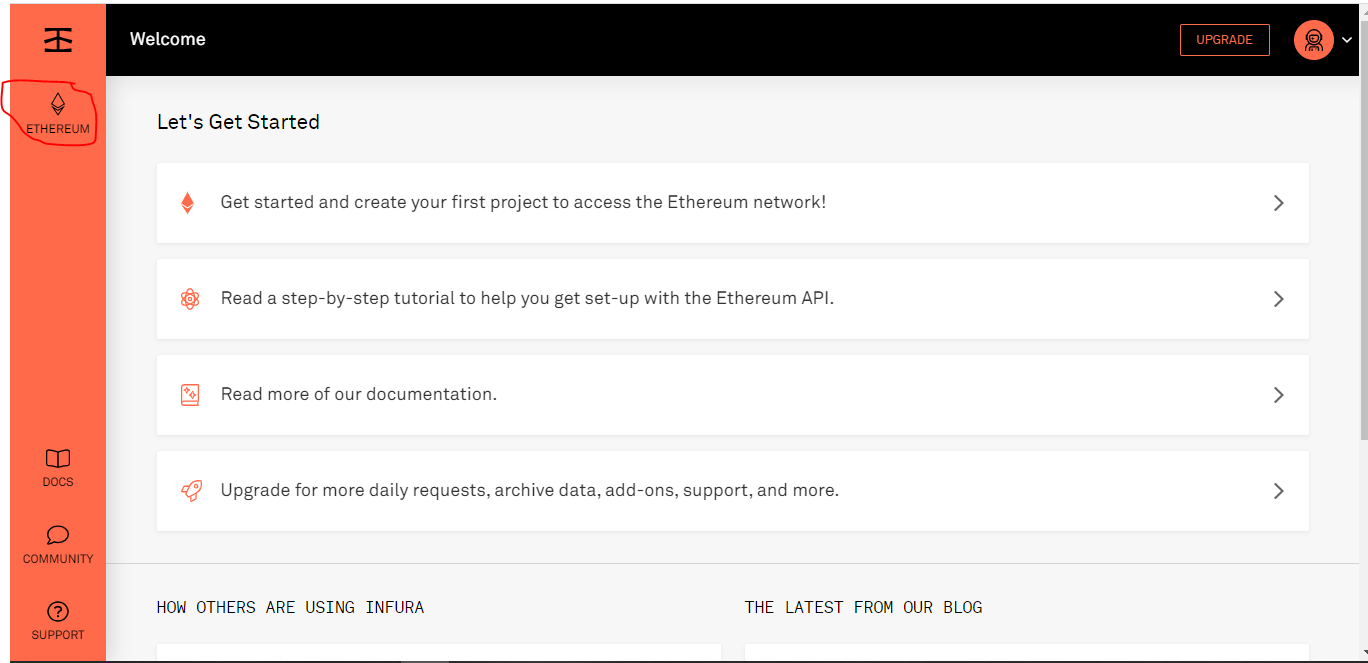
Type in your email and choose a password for you Infura account. Click SIGN UP.

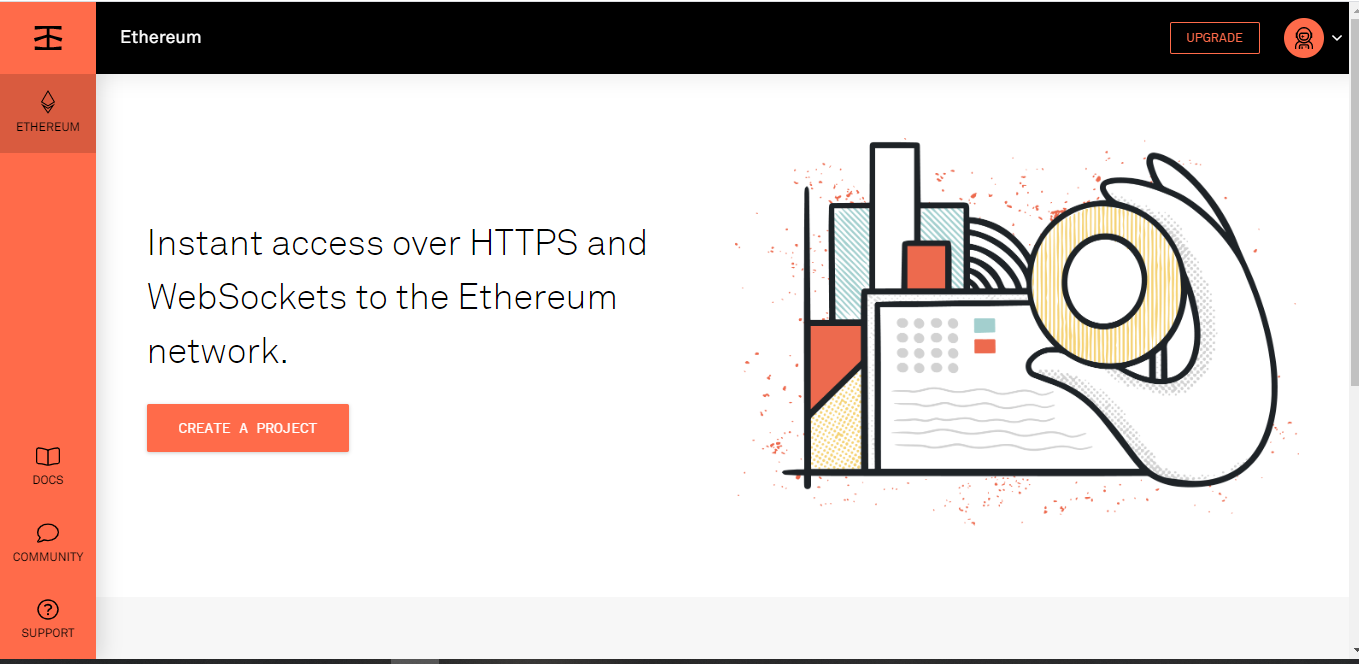


Check your email and confirm the email address through the link presented on the email from Infura. Once you click on the link, you will see your account dashboard.

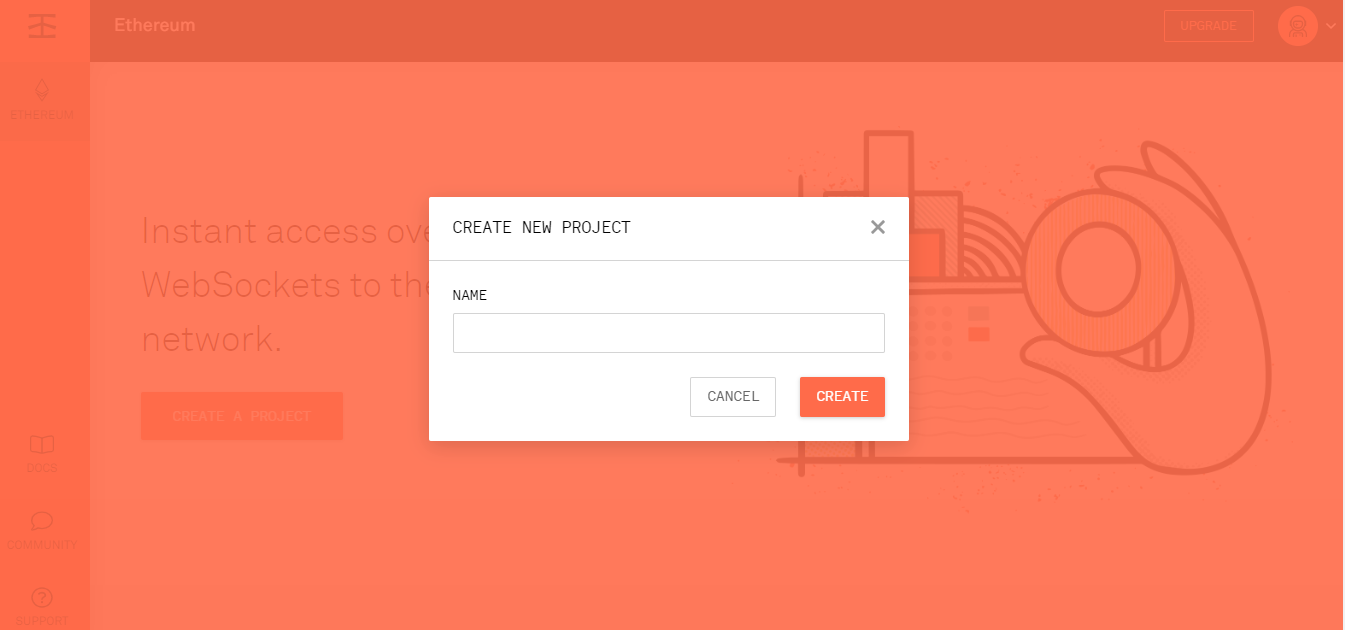


Click on Ethereum Icon on the left side of the dashboard.

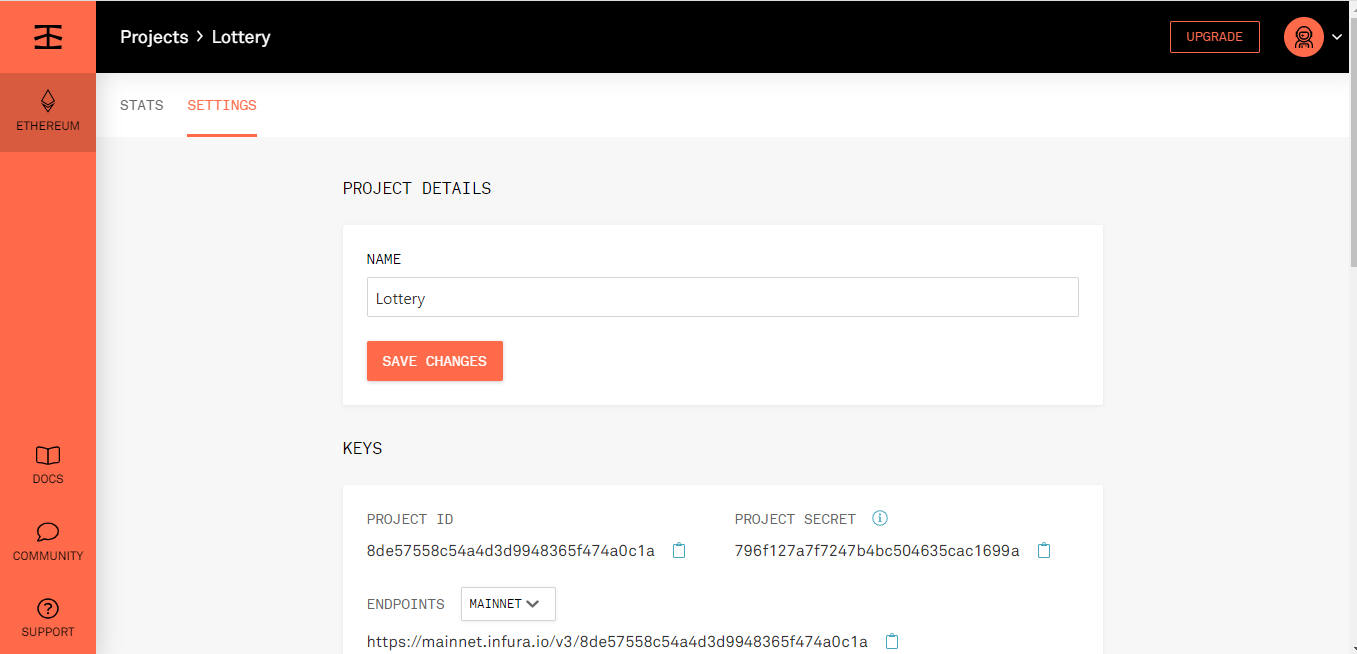




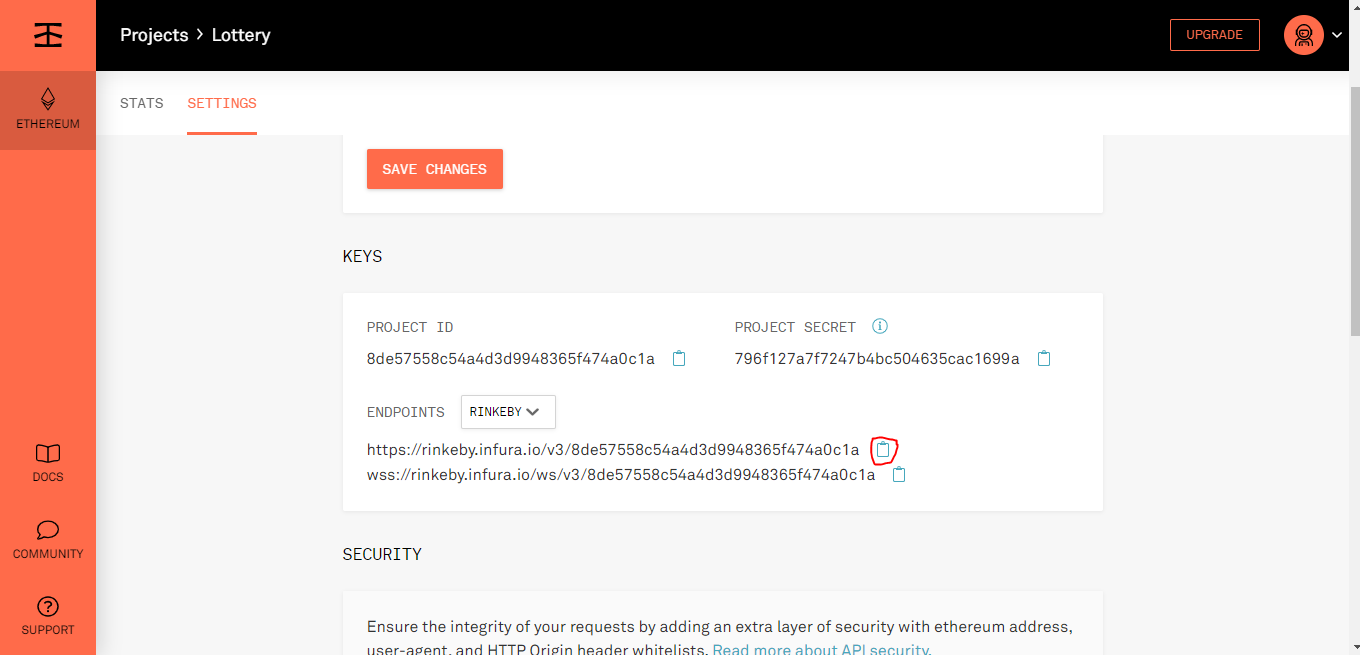
Click CREATE A PROJECT.



You can enter whatever name you want because it is not impact any of your solidity contract. But, let’s be consistent and enter Lottery as a project name. Enter Create.



Scroll down and notice the ENDPOINTS TAB under KEYS. Since we are deploying to Rinkeby network, Change MAINET to RINKEBY.



Copy the link at the first line under the endpoints section. We will need this while writing our deployment code so keep it as a backup.

**Part 8: Wallet Provider Setup**

We're going to first flip on over toward terminal and install one module that's going to help us in creating a provider that will connect to the network. The provider is how our web 3 instance talks to some particular network. We will install a special provider that allows us to not only connect to some outside API or some outside node, but it also allows us to simultaneously unlock an account. We have to set up this provider manually. And when we set up the provider we need to specify exactly what network we want to connect to. And we also need to use it to somehow unlock an account that will be used for all the different deployment requests that we make. So that's why we are installing this module the truffle HD wallet provider is going to be this part right here. It's going to allow us to connect to the Rinkeby network hosted through Infura and it's going to simultaneously allow us to very easily unlock and account to use.

Inside of root project directory, install Truffle HDWallet Provider. Type in command:

npm install –save truffle-hdwallet-provider

**Part 7 : Contract Deployment**

Insider of root project directory create deploy.js file. Type in command:

nano deploy.js

Copy the following code in compile.js :

const HDWalletProvider = require('truffle-hdwallet-provider');

const Web3 = require('web3');

const { interface, bytecode } = require('./compile');

const provider = new HDWalletProvider(

'game saddle oyster laundry equal loop lunch allow cactus endless hover unfair',

'https://rinkeby.infura.io/orDImgKRzwNrVCDrAk5Q'

);

const web3 = new Web3(provider);

const deploy = async () => {

const accounts = await web3.eth.getAccounts();

console.log('Attempting to deploy from account', accounts[0]);

const result = await new web3.eth.Contract(JSON.parse(interface))

.deploy({ data: '0x' + bytecode })

.send({ gas: '1000000', from: accounts[0] });

console.log('Contract deployed to', result.options.address);

};

deploy();

NOTE: **The twelve words** 'game saddle oyster laundry equal loop lunch allow cactus endless hover unfair' **should be replaced by the backup secret phrase of the account you want to use to deploy the contract. The link below i.e.** 'https://rinkeby.infura.io/orDImgKRzwNrVCDrAk5Q' **that also should be replaced by Infura link of your project.**

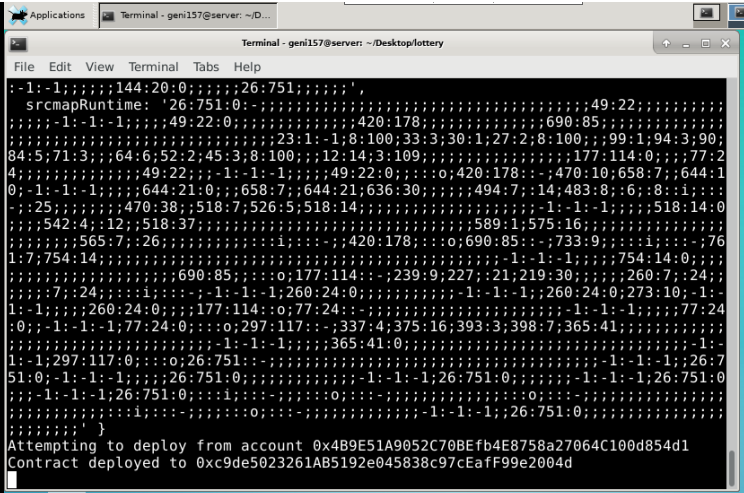
Once you complete the code, perform following actions:

* **Ctrl + O to save.**
* **Enter**
* **Ctrl + X to exit.**

In the terminal, inside of the lottery project directory, run the command:

node deploy.js

Once your contract is deployed, you will see the address where you contract is deployed on screen like below:



Once you have the output like above, that means you have address where your contract is deployed, you can simply **Ctrl + C** to go back to terminal because it might take a lot of time to completely run deploy.js. For us, once we get deployed contract address you can stop process.

The address on the second line on the screen above is the address of our deployed contract. So, this is where it exists on the Rinkeby test network now. At this point you might be saying OK well that's great. We deployed it to be what. But like what good is that to us. Well in an ideal world if we had some idea of how to interface with these contracts from a web application we could then build some HTML application or HTML JavaScript CSS application that could somehow interact with this contract.