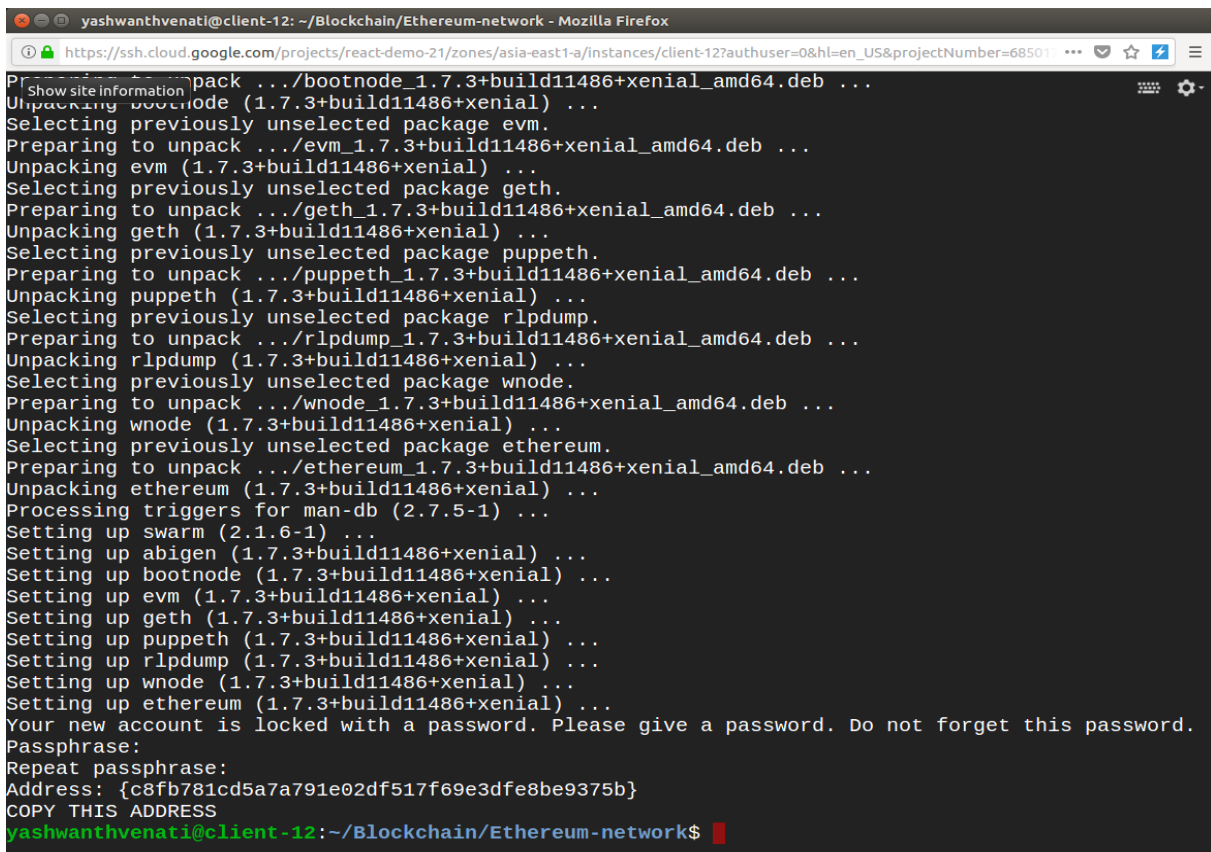


Private Blockchain and Measurements

The main goal of this project is to setup a private blockchain and take some measurements. Also, we should be able to deploy contracts from our machine.

1.Setup a Node

- ◆ On a Linux machine (Ubuntu 16.04) in the cloud, do the following
- ◆ Download the code from the GitHub repo and run the **install.sh**, this script installs all the required software's needed.
- ◆ After installation, it creates a new account and prompts you for password, enter the password
- ◆ It gives an account ID, after you enter password, note this ID as we need it in the next steps.



```
yashwanthvenati@client-12: ~/Blockchain/Ethereum-network - Mozilla Firefox
https://ssh.cloud.google.com/projects/react-demo-21/zones/asia-east1-a/instances/client-12?authuser=0&hl=en_US&projectNumber=68501
Pack .../bootnode_1.7.3+build11486+xenial_amd64.deb ...
Unpacking bootnode (1.7.3+build11486+xenial) ...
Selecting previously unselected package evm.
Preparing to unpack .../evm_1.7.3+build11486+xenial_amd64.deb ...
Unpacking evm (1.7.3+build11486+xenial) ...
Selecting previously unselected package geth.
Preparing to unpack .../geth_1.7.3+build11486+xenial_amd64.deb ...
Unpacking geth (1.7.3+build11486+xenial) ...
Selecting previously unselected package puppeth.
Preparing to unpack .../puppeth_1.7.3+build11486+xenial_amd64.deb ...
Unpacking puppeth (1.7.3+build11486+xenial) ...
Selecting previously unselected package rlpdump.
Preparing to unpack .../rlpdump_1.7.3+build11486+xenial_amd64.deb ...
Unpacking rlpdump (1.7.3+build11486+xenial) ...
Selecting previously unselected package wnode.
Preparing to unpack .../wnode_1.7.3+build11486+xenial_amd64.deb ...
Unpacking wnode (1.7.3+build11486+xenial) ...
Selecting previously unselected package ethereum.
Preparing to unpack .../ethereum_1.7.3+build11486+xenial_amd64.deb ...
Unpacking ethereum (1.7.3+build11486+xenial) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up swarm (2.1.6-1) ...
Setting up abigen (1.7.3+build11486+xenial) ...
Setting up bootnode (1.7.3+build11486+xenial) ...
Setting up evm (1.7.3+build11486+xenial) ...
Setting up geth (1.7.3+build11486+xenial) ...
Setting up puppeth (1.7.3+build11486+xenial) ...
Setting up rlpdump (1.7.3+build11486+xenial) ...
Setting up wnode (1.7.3+build11486+xenial) ...
Setting up ethereum (1.7.3+build11486+xenial) ...
Your new account is locked with a password. Please give a password. Do not forget this password.
Passphrase:
Repeat passphrase:
Address: {c8fb781cd5a7a791e02df517f69e3dfe8be9375b}
COPY THIS ADDRESS
yashwanthvenati@client-12:~/Blockchain/Ethereum-network$
```

2. Create a Genesis

- ◆ Create as many nodes you like and note the account ID's
- ◆ Open the **genesis.json** from the downloaded repository files and in the alloc section replace the content with ID and the initial balance you want to allocate to that particular account.
- ◆ Add this for all the nodes you have created and want to be a part of same network.
- ◆ All the genesis files should be same on all the nodes

```
1  {
2
3      "config":{
4          "chainId":15,
5          "homesteadBlock":0,
6          "eip155Block":0,
7          "eip158Block":0
8      },
9      "difficulty":"200000",
10     "gasLimit":"2100000000",
11     "alloc":{
12         "9da3813aad90cd9f5b8db5299b39594470728596":{
13             "balance":"11111111111111111111111111111111"
14         },
15         "b3105e05ce0af42842a5f37760967fac2fe83c8a":{
16             "balance":"22222222222222222222222222222222"
17         },
18         "05d0d3591e54527c21884d865a7490e594bc1cb8":{
19             "balance":"33333333333333333333333333333333"
20         },
21         "8a2876d58dba4c59a306c2e81915260e09ae570c":{
22             "balance":"44444444444444444444444444444444"
23         },
24         "a67ec96a48a46fdddc07d85e5dbf97145a109fc":{
25             "balance":"55555555555555555555555555555555"
26     }
```

3. Initialize the Node with genesis

- ◆ Now to initialize the node with genesis block
- ◆ Run **init.sh**, this script initializes the node with genesis and allocates the balance to the account in the genesis file.
- ◆ Do this in all the nodes

4. Start the node

- ◆ To start the node, run the start-node.sh with corresponding node ID as the command line argument.
- ◆ This script starts the geth cli.
- ◆ Enter your password you have entered in the step 1

5. Connect the nodes

- ◆ To connect the nodes, you need the nodeID and its IP address.
- ◆ In the geth cli, enter admin.nodeInfo.enode
- ◆ You will get the enode ID, copy the enode ID, we need this to connect to the particular node.
- ◆ From the node from which you want to connect to this node, enter the command as below, replacing the enode with your enode and corresponding IP address of the node.

[admin.addPeer\("enode://fa1c0022c4b88a7fe3b94a9dee015efe8deb0b14997ae503213114640395475a7a5babbfcd2df9b4b069c515eecf07c72e7c9f0d345f7d1a39c50fde06a1b6ab@35.199.158.85:30303"\)](#)

- ◆ Do this in all the nodes, to verify this
- ◆ Enter admin.peers in the geth cli, you will receive the list of nodes it is connected with.

6. Start Mining

- ◆ To start mining, enter miner.start() in any of the node
- ◆ Initially it may take time because it creates DAG, after that you will see the blocks getting mined

7. Stop Mining

- ◆ To stop mining, enter miner.stop() in the node you started mining and this stops mining.

8. Create a smart contract

- ◆ To create a smart contract go to solidity IDE from your browser and write a contract.
- ◆ Compile it and copy the web3 code and paste it in the geth cli.
- ◆ To submit a smart contract successfully a miner should be running, so start a miner as shown in previous step.
- ◆ The contract is submitted and it gives hash and address and also after the contract is created it gives hash and address of the contract.
- ◆ We can use this address to run the contract in future.

7. Run a smart contract

- ◆ Run the smart contract using the variable.method to execute the contract.
- ◆ This displays the output if any depending upon the contract.

8. Deploy a contract from local system

















- ◆ To deploy a contract from the local system to the private network.
- ◆ This can be done using Truffle framework, it compiles the solidity contracts and deploys them on the private network.
- ◆ We need the IP address of the node and RPC should be enabled on the node and its port number.
- ◆ Create a folder and run truffle init.
- ◆ A contracts, migrations and other folders are created along with truffle.js
- ◆ These config details should be placed in truffle.js
- ◆ The contracts should be written in contracts folder, these contracts are automatically compiled using truffle I,e no need to use the online IDE.
- ◆ Write a contract in the folder and with proper config in truffle.js
- ◆ Run truffle compile
- ◆ Run truffle migrate --network live
- ◆ The contract is compiled and deployed in the corresponding node in the truffle.js, you have you start mining in the node, to mine the contract and create it after submission.
- ◆ After successful creation the hash and address are displayed in the console.
- ◆ You can run the contract.

Reports

I have used 15 google compute machines running ubuntu 16.04 for this experiment.

Each machine is from different location, few are with different configuration.

The Following is the configuration name and location of the machines.

<input type="checkbox"/>	Name	Zone	Machine type
<input type="checkbox"/> 	yashwanth	us-central1-b	2 vCPUs, 7.5 GB
<input type="checkbox"/> 	yash2	us-east1-b	1 vCPU, 3.75 GB
<input type="checkbox"/> 	yashwanth-2	us-central1-a	1 vCPU, 1.7 GB
<input type="checkbox"/> 	client-3	asia-east1-a	1 vCPU, 1.7 GB
<input type="checkbox"/> 	client-4	australia-southeast1-c	1 vCPU, 3.75 GB
<input type="checkbox"/> 	client-5	us-west1-c	1 vCPU, 3.75 GB
<input type="checkbox"/> 	client-6	europa-west1-c	1 vCPU, 3.75 GB
<input type="checkbox"/> 	client-7	asia-south1-a	1 vCPU, 1.7 GB
<input type="checkbox"/> 	client-8	asia-northeast1-b	1 vCPU, 1.7 GB
<input type="checkbox"/> 	client-9	europa-west2-c	1 vCPU, 1.7 GB
<input type="checkbox"/> 	client-10	southamerica-east1-a	1 vCPU, 1.7 GB
<input type="checkbox"/> 	client-11	asia-south1-c	1 vCPU, 1.7 GB
<input type="checkbox"/> 	client-12	asia-east1-a	1 vCPU, 1.7 GB
<input type="checkbox"/> 	client-13	europa-west3-c	1 vCPU, 1.7 GB
<input type="checkbox"/> 	client-14	us-east1-b	1 vCPU, 1.7 GB
<input type="checkbox"/> 	client-15	us-east4-a	1 vCPU, 1.7 GB

The tests are conducted by creating few smart contracts and deploying them on the network with

- ◆ 5 Nodes
- ◆ 10 Nodes
- ◆ 15 Nodes

The block creation time is as shown below

For 5 Nodes

	Test
Node 1(Miner)	564.407 μs
Node 2	5.309 ms
Node 3	4.426 ms
Node 4	4.235 ms
Node 5	7.841 ms

For 10 Nodes

	Test
Node 1	6.584 ms
Node 2 (Miner)	367.532 μs
Node 3	4.781 ms
Node 4	5.549 ms
Node 5	8.271 ms
Node 6	6.961 ms
Node 7	5.832 ms
Node 8	3.165 ms
Node 9	5.236 ms
Node 10	4.308 ms

For 15 Nodes

	Test
Node 1	6.142 ms
Node 2 (Miner)	252.956 μ s
Node 3	4.464 ms
Node 4	6.820 ms
Node 5	8.915 ms
Node 6	6.560 ms
Node 7	6.113 ms
Node 8	6.649 ms
Node 9	6.975 ms
Node 10	7.021 ms
Node 11	6.322 ms
Node 12	3.862 ms
Node 13	6.322 ms
Node 14	6.949 ms
Node 15	20.871 ms

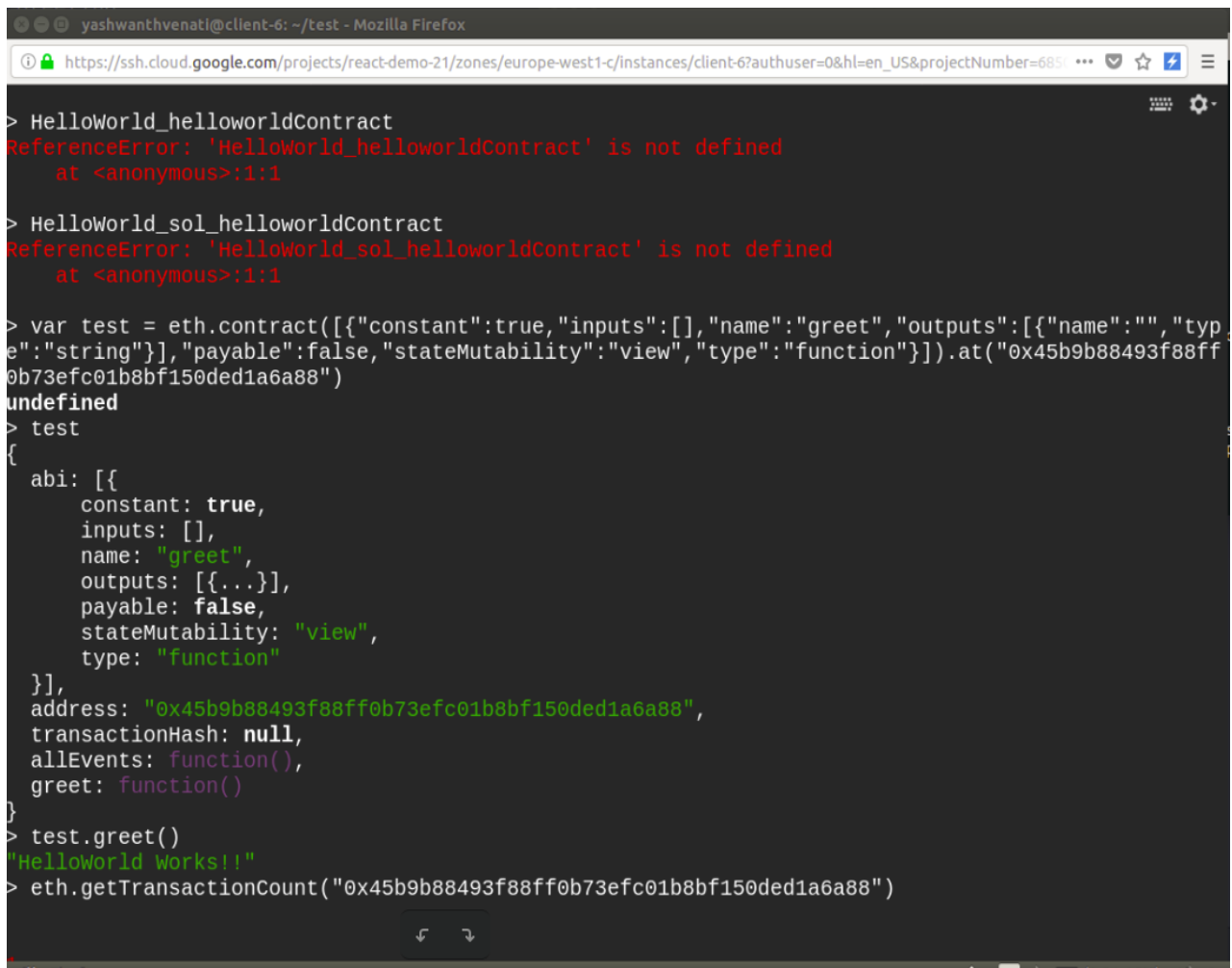
All the nodes on average take around 6.5 ms to create a block.

Few interesting things are

1. The miner always takes very less time to create a block because, it is the one which runs and creates the block in the network, which is then replicated onto other nodes.
2. Node 15 in test 3 takes more than 20ms, because it has a DAG being generated, i.e. there is a cpu intensive process running on this node and creation on block took longer compared to other nodes in the network.

3. Although, the nodes are geographically distributed randomly, there should be a delay to other nodes outside USA, but the nodes in Asia have very less time

Few Screenshots of the experiment



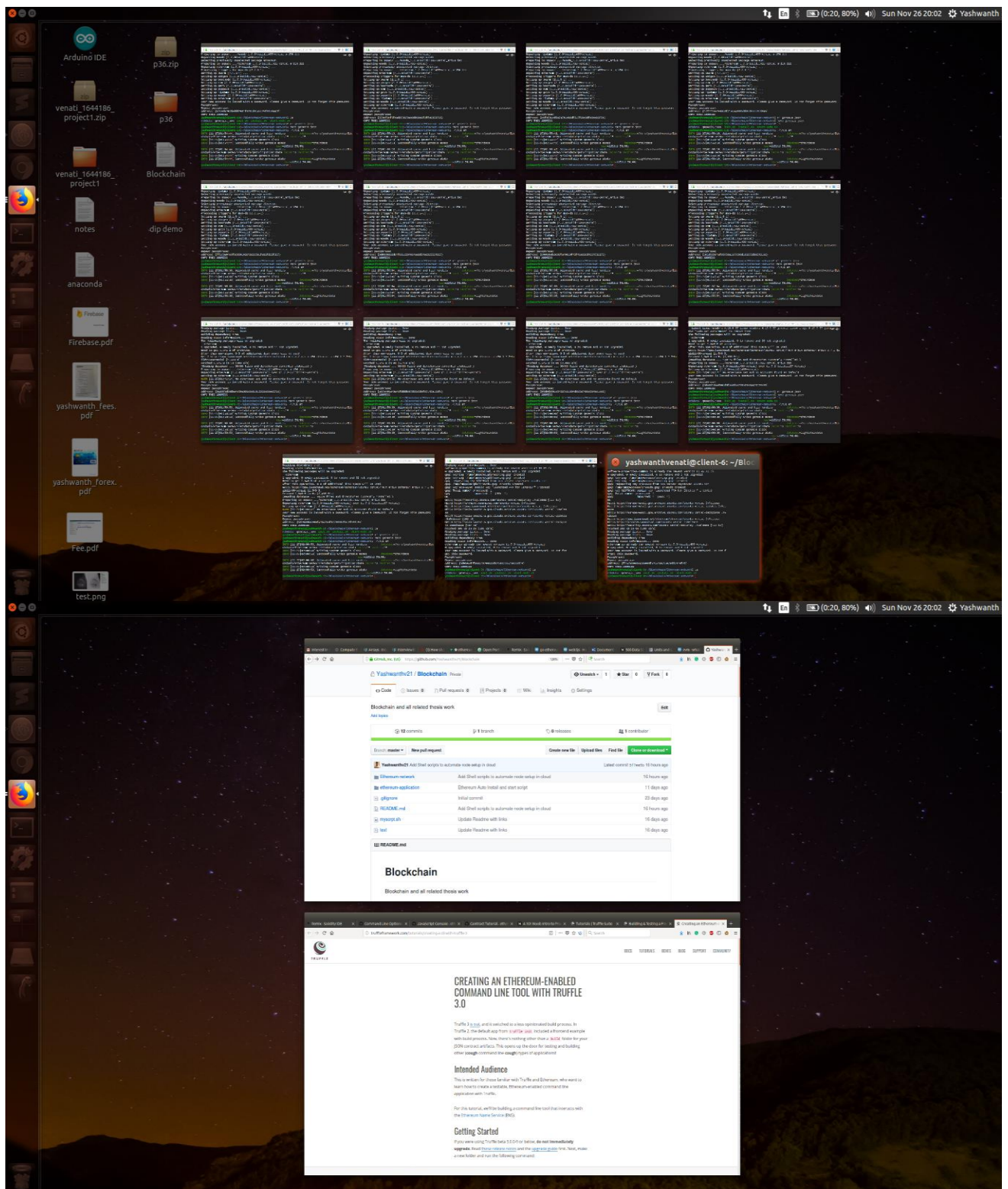
```
yashwanthvenati@client-6: ~/test - Mozilla Firefox
https://ssh.cloud.google.com/projects/react-demo-21/zones/europe-west1-c/instances/client-6?authuser=0&hl=en_US&projectNumber=685...

> HelloWorld_helloworldContract
ReferenceError: 'HelloWorld_helloworldContract' is not defined
    at <anonymous>:1:1

> HelloWorld_sol_helloworldContract
ReferenceError: 'HelloWorld_sol_helloworldContract' is not defined
    at <anonymous>:1:1

> var test = eth.contract([{"constant":true,"inputs":[],"name":"greet","outputs":[{"name":"","type":"string"}],"payable":false,"stateMutability":"view","type":"function"}]).at("0x45b9b88493f88ff0b73efc01b8bf150ded1a6a88")
undefined
> test
{
  abi: [{
    constant: true,
    inputs: [],
    name: "greet",
    outputs: [{...}],
    payable: false,
    stateMutability: "view",
    type: "function"
  }],
  address: "0x45b9b88493f88ff0b73efc01b8bf150ded1a6a88",
  transactionHash: null,
  allEvents: function(),
  greet: function()
}
> test.greet()
"HelloWorld Works!!"
> eth.getTransactionCount("0x45b9b88493f88ff0b73efc01b8bf150ded1a6a88")
```

The HelloWorld contract deployed to the network



15 machines running in our private network

```
yashwanthvenati@client-4: ~/Blockchain/Ethereum-network - Mozilla Firefox
https://ssh.cloud.google.com/projects/react-demo-21/...

yashwanthvenati@yashwanth-2: ~/Blockchain/Ethereum-network - Mozilla Firefox
https://ssh.cloud.google.com/projects/react-demo-21/zones/...

yashwanthvenati@client-3: ~/Blockchain/Ethereum-network - Mozilla Firefox
https://ssh.cloud.google.com/projects/react-demo-21/zones/...

yashwanthvenati@client-5: ~/Blockchain/Ethereum-network - Mozilla Firefox
https://ssh.cloud.google.com/projects/react-demo-21/zones/us-central1-b/instances/client5-sta...

yashwanthvenati@yashwanth: ~/Blockchain/Ethereum-network - Mozilla Firefox
https://ssh.cloud.google.com/projects/react-demo-21/zones/us-central1-b/instances/yashwanth...

admin.peers
[
  {
    caps: ["eth/63"],
    id: "3807d7b295a7d48c388b14e1ad8b0b9e523c6954a22eab395cc218617e9cb07190528f",
    name: "geth/v1.7.2-stable-1db4ecdc/linux-amd64/go1.9",
    network: {
      localAddress: "10.128.0.2:30303",
      remoteAddress: "35.188.153.184:30303"
    },
    protocols: {
      eth: {
        difficulty: 100000,
        head: "x4d8b0dd6aced74cb0910299f1b0f52a0935147b06e5bc5debaa832c6d54c44e",
        version: 64
      }
    }
  }
]

admin.peers
[
  {
    caps: ["eth/63"],
    id: "3807d7b295a7d48c388b14e1ad8b0b9e523c6954a22eab395cc218617e9cb07190528f",
    name: "geth/v1.7.2-stable-1db4ecdc/linux-amd64/go1.9",
    network: {
      localAddress: "10.152.0.2:50534",
      remoteAddress: "35.188.153.184:30303"
    },
    protocols: {
      eth: {
        difficulty: 100000,
        head: "x4d8b0dd6aced74cb0910299f1b0f52a0935147b06e5bc5debaa832c6d54c44e",
        version: 64
      }
    }
  }
]

admin.peers
[
  {
    caps: ["eth/63"],
    id: "3807d7b295a7d48c388b14e1ad8b0b9e523c6954a22eab395cc218617e9cb07190528f",
    name: "geth/v1.7.2-stable-1db4ecdc/linux-amd64/go1.9",
    network: {
      localAddress: "10.140.0.2:30303",
      remoteAddress: "35.188.153.184:30303"
    },
    protocols: {
      eth: {
        difficulty: 100000,
        head: "x4d8b0dd6aced74cb0910299f1b0f52a0935147b06e5bc5debaa832c6d54c44e",
        version: 64
      }
    }
  }
]

admin.peers
[
  {
    caps: ["eth/63"],
    id: "08f6a0d0fd2ba14c908af938a6023f964146d8c34fecdd97d33adeb5d2993a9ff1e2f02b73727e3a3c36d93693945901c3c5c4f6c0c2a2f7227f63",
    name: "geth/v1.7.2-stable-1db4ecdc/linux-amd64/go1.9",
    network: {
      localAddress: "10.128.0.2:30303",
      remoteAddress: "35.189.58.213:50534"
    },
    protocols: {
      eth: {
        difficulty: 100000,
        head: "x4d8b0dd6aced74cb0910299f1b0f52a0935147b06e5bc5debaa832c6d54c44e",
        version: 64
      }
    }
  },
  {
    caps: ["eth/63"],
    id: "72b786736c302161ccc64c0afdb9c9f079dc62cb88b51a06ade1008ea27fa99cea8e7f9688b30cbe1d37e565ebbede224acda414911f30e5aacc8d1",
    name: "geth/v1.7.2-stable-1db4ecdc/linux-amd64/go1.9",
    network: {
      localAddress: "10.128.0.2:30303",
      remoteAddress: "104.104.188.66:51326"
    },
    protocols: {
      eth: {
        difficulty: 100000,
        head: "x4d8b0dd6aced74cb0910299f1b0f52a0935147b06e5bc5debaa832c6d54c44e",
        version: 64
      }
    }
  },
  {
    caps: ["eth/63"],
    id: "72b786736c302161ccc64c0afdb9c9f079dc62cb88b51a06ade1008ea27fa99cea8e7f9688b30cbe1d37e565ebbede224acda414911f30e5aacc8d1",
    name: "geth/v1.7.2-stable-1db4ecdc/linux-amd64/go1.9",
    network: {
      localAddress: "10.128.0.2:30303",
      remoteAddress: "35.189.159.85:30432"
    },
    protocols: {
      eth: {
        difficulty: 100000,
        head: "x4d8b0dd6aced74cb0910299f1b0f52a0935147b06e5bc5debaa832c6d54c44e",
        version: 64
      }
    }
  },
  {
    caps: ["eth/63"],
    id: "cc30e413c30bd52140f203ac918c06d8fb113d219902eaf906d0000c85f8775f8a447279057fae130759a0bdc145923b58377041f2970f9592",
    name: "geth/v1.7.2-stable-1db4ecdc/linux-amd64/go1.9",
    network: {
      localAddress: "10.128.0.2:30303",
      remoteAddress: "35.189.173.86:30308"
    },
    protocols: {
      eth: {
        difficulty: 100000,
        head: "x4d8b0dd6aced74cb0910299f1b0f52a0935147b06e5bc5debaa832c6d54c44e",
        version: 64
      }
    }
  }
]

admin.peers
[
  {
    caps: ["eth/63"],
    id: "3807d7b295a7d48c388b14e1ad8b0b9e523c6954a22eab395cc218617e9cb07190528f",
    name: "geth/v1.7.2-stable-1db4ecdc/linux-amd64/go1.9",
    network: {
      localAddress: "10.128.0.2:30432",
      remoteAddress: "35.188.153.184:30303"
    },
    protocols: {
      eth: {
        difficulty: 100000,
        head: "x4d8b0dd6aced74cb0910299f1b0f52a0935147b06e5bc5debaa832c6d54c44e",
        version: 64
      }
    }
  }
]
```

5 Nodes are connected

