Creating a private blockchain on Ec2 remote server

Contents

[Code 2](#_Toc124892301)

[GitHub Repo 2](#_Toc124892302)

[Docker Implementation 2](#_Toc124892303)

[Implementation 3](#_Toc124892304)

[Creating an EC2 instance 3](#_Toc124892305)

[Creating a Security Group 5](#_Toc124892306)

[Installing Ethereum on EC2 instance 6](#_Toc124892307)

[Setup directory to create Private Network 8](#_Toc124892308)

[Create Accounts 9](#_Toc124892309)

[Copy Bash files to server 9](#_Toc124892310)

[Create Bootnode 10](#_Toc124892311)

[Setup genesis block 10](#_Toc124892312)

[Initialize node 12](#_Toc124892313)

[Start Bootnode using bash file 12](#_Toc124892314)

[Start Node1 using Bash file 13](#_Toc124892315)

[Start Node2 using bash file 14](#_Toc124892316)

[Start Node3 using bash file 14](#_Toc124892317)

[Checking Peers of node1 15](#_Toc124892318)

[Peers count on node2 and node3 15](#_Toc124892319)

[Validation of Blockchain 15](#_Toc124892320)

[Curl Operation on localhost 15](#_Toc124892321)

[Curl Operation from local System 15](#_Toc124892322)

[Connection to local environment 16](#_Toc124892323)

[Download keystore 16](#_Toc124892324)

[Connecting Remote Blockchain to Metamask 17](#_Toc124892325)

[Importing accounts into Metamask using Keystore files 17](#_Toc124892326)

[Contracts 18](#_Toc124892327)

[Solution Architecture 19](#_Toc124892328)

[Process Flow 19](#_Toc124892329)

[Flight Contracts 20](#_Toc124892330)

[Running Status 20](#_Toc124892331)

[Booking Status 20](#_Toc124892332)

[Airline Contracts 20](#_Toc124892333)

[Transaction Contracts 20](#_Toc124892334)

# Code

We have created a repository on github and we have checked in all code for this project there. Bash files used for running blockchain node are also checked in under directory **bashfile**. Details of commands used for creating this blockchain network is recorded in **readme.md** file.

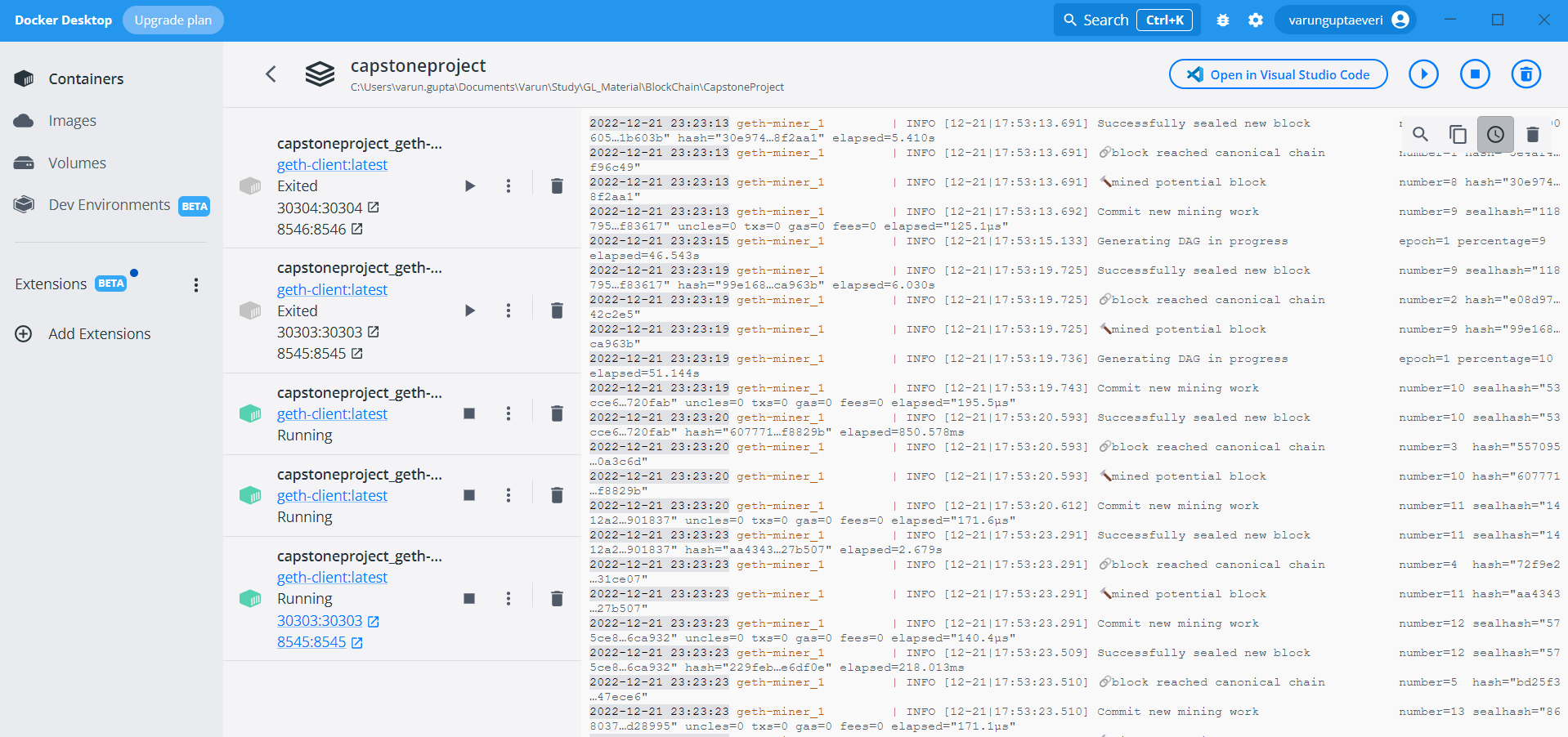
## GitHub Repo

[VarunKesarwani/PrivateBlockChain: Setting private block chain network (github.com)](https://github.com/VarunKesarwani/PrivateBlockChain)

## Docker Implementation

We have also implemented a blockchain creation process using docker. This implementation was done for local machine, which would help in development and testing of contracts on local machine before deploying these contracts to remote server.

We have implemented a single node implementation of blockchain using docker-compose.yml present in github repository. As per this implementation we have a mining block and a main node connected to boot node. This main node exposes an RPC port to connect local environment like Metamask and Remix.



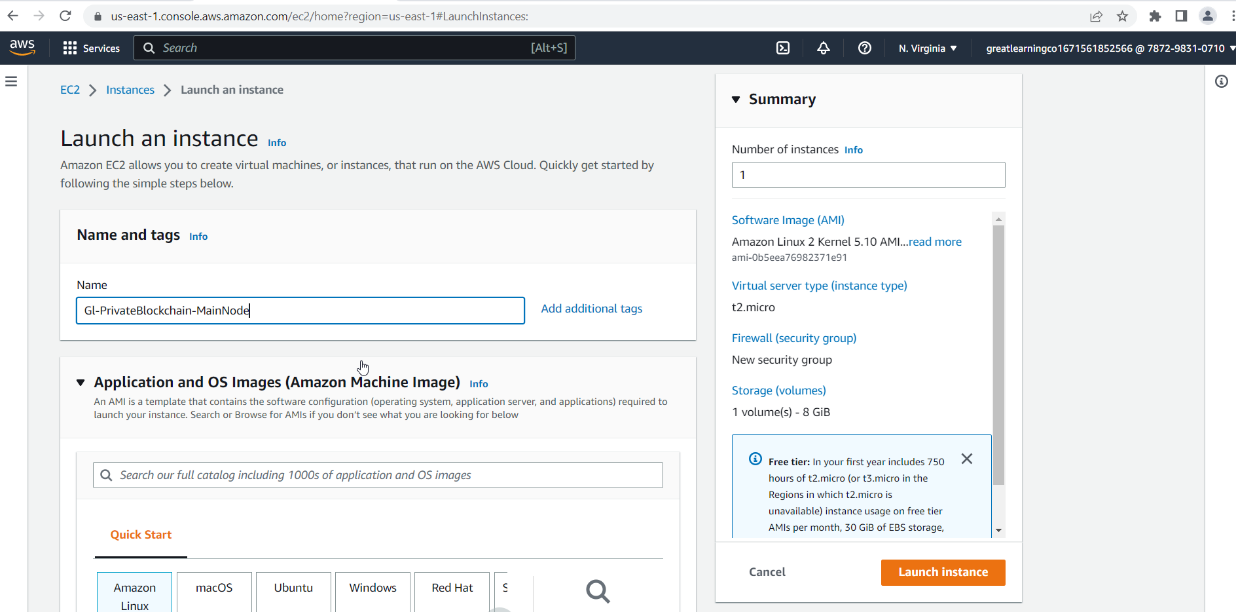
Graphical user interface, application

Description automatically generated

# Architecture

# Implementation

## Creating an EC2 instance

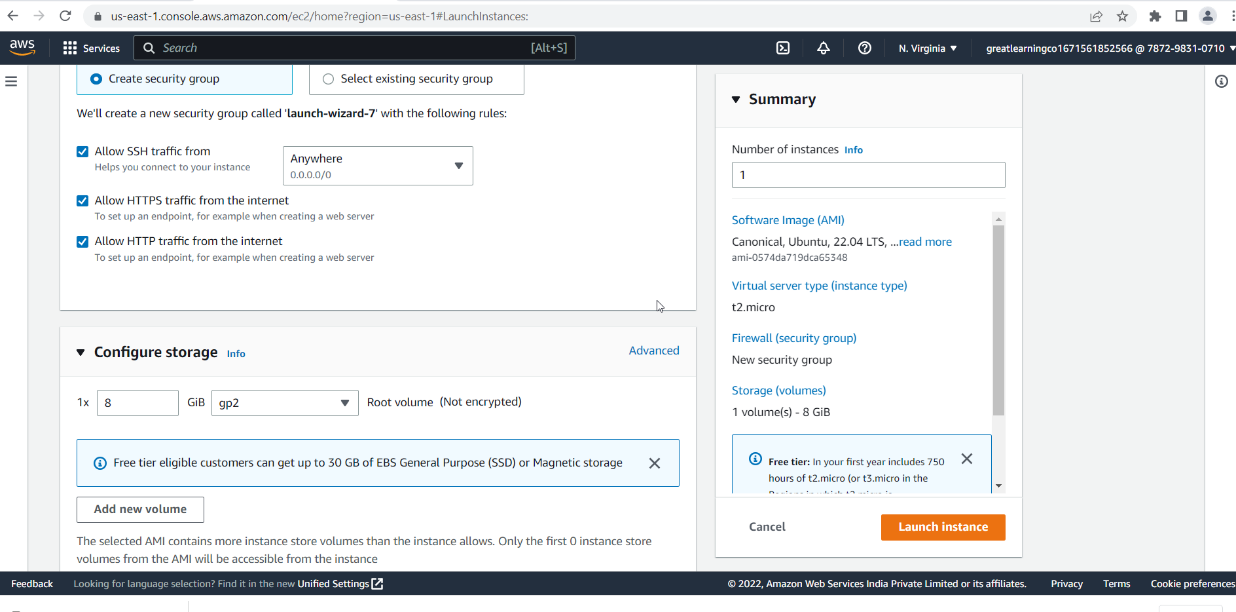


Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

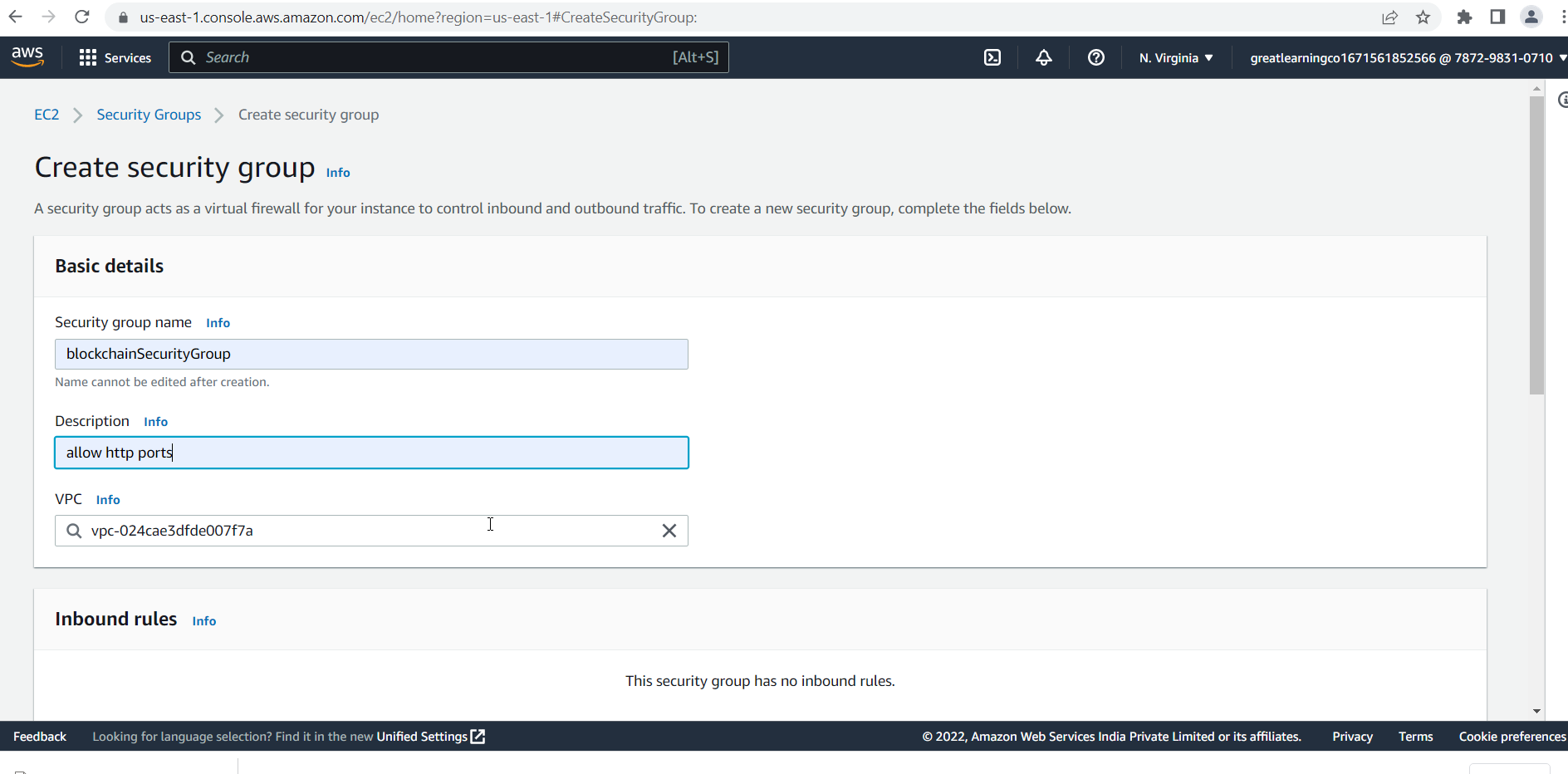


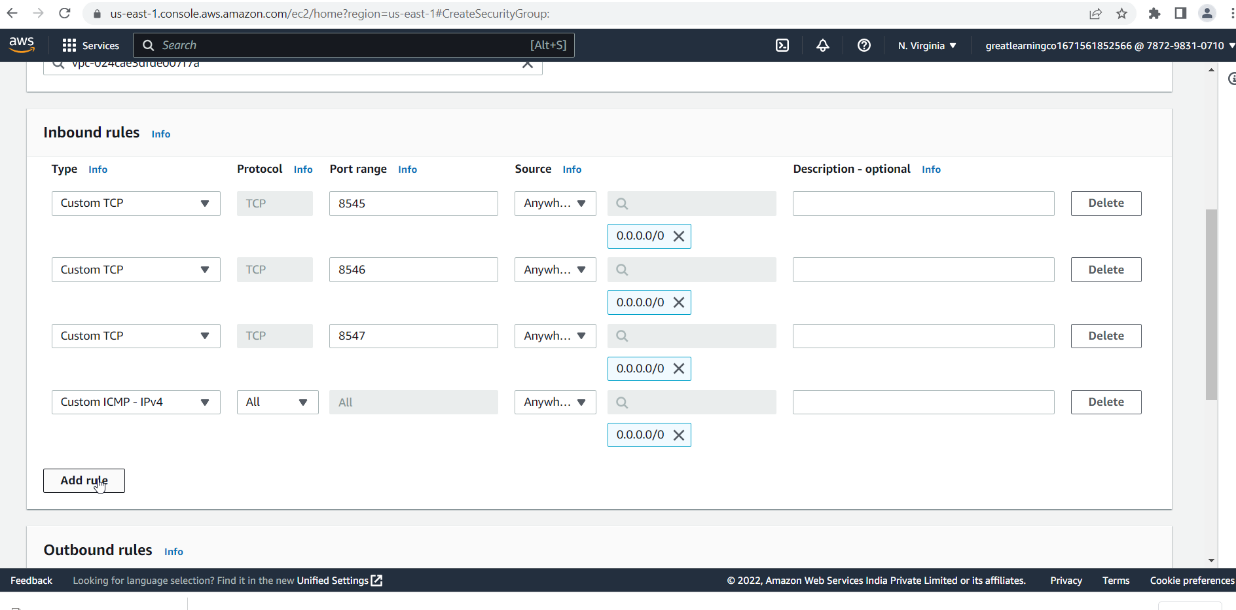
Graphical user interface, text, application, email

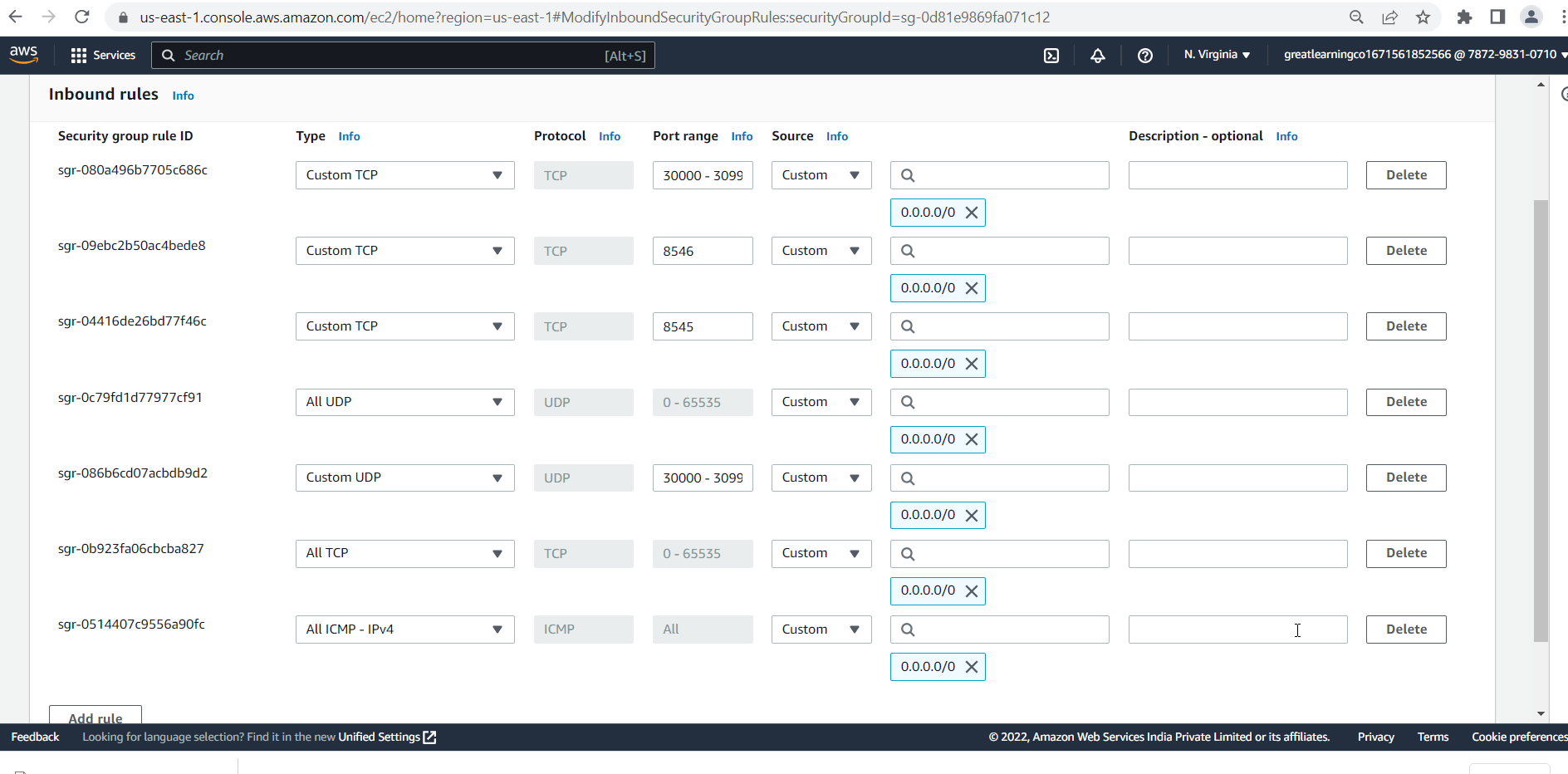
Description automatically generated

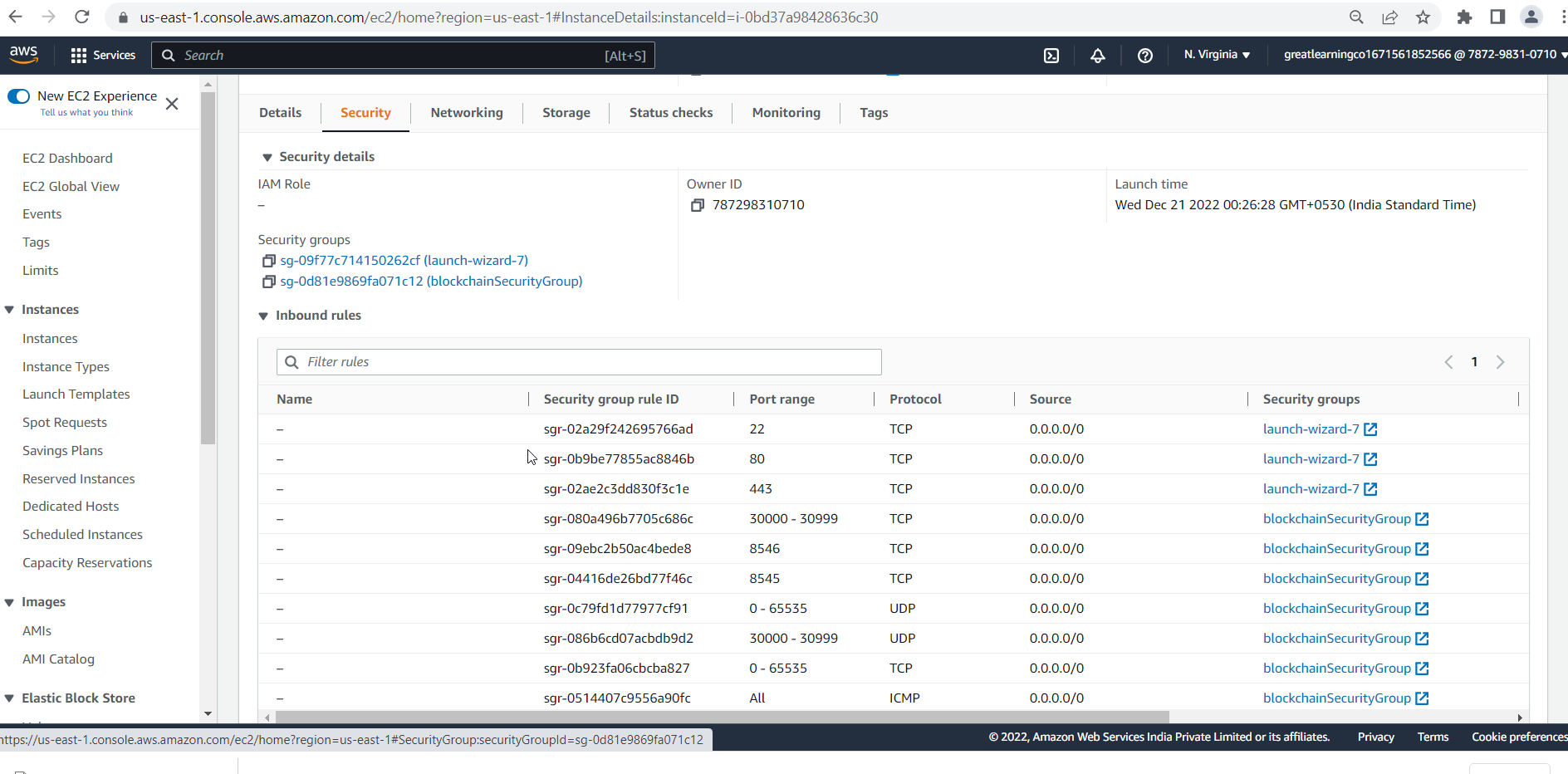
## Creating a Security Group

A new security group was created which provided external access to ports 8545, 8546 and 8547. These ports would be mapped to individual nodes.







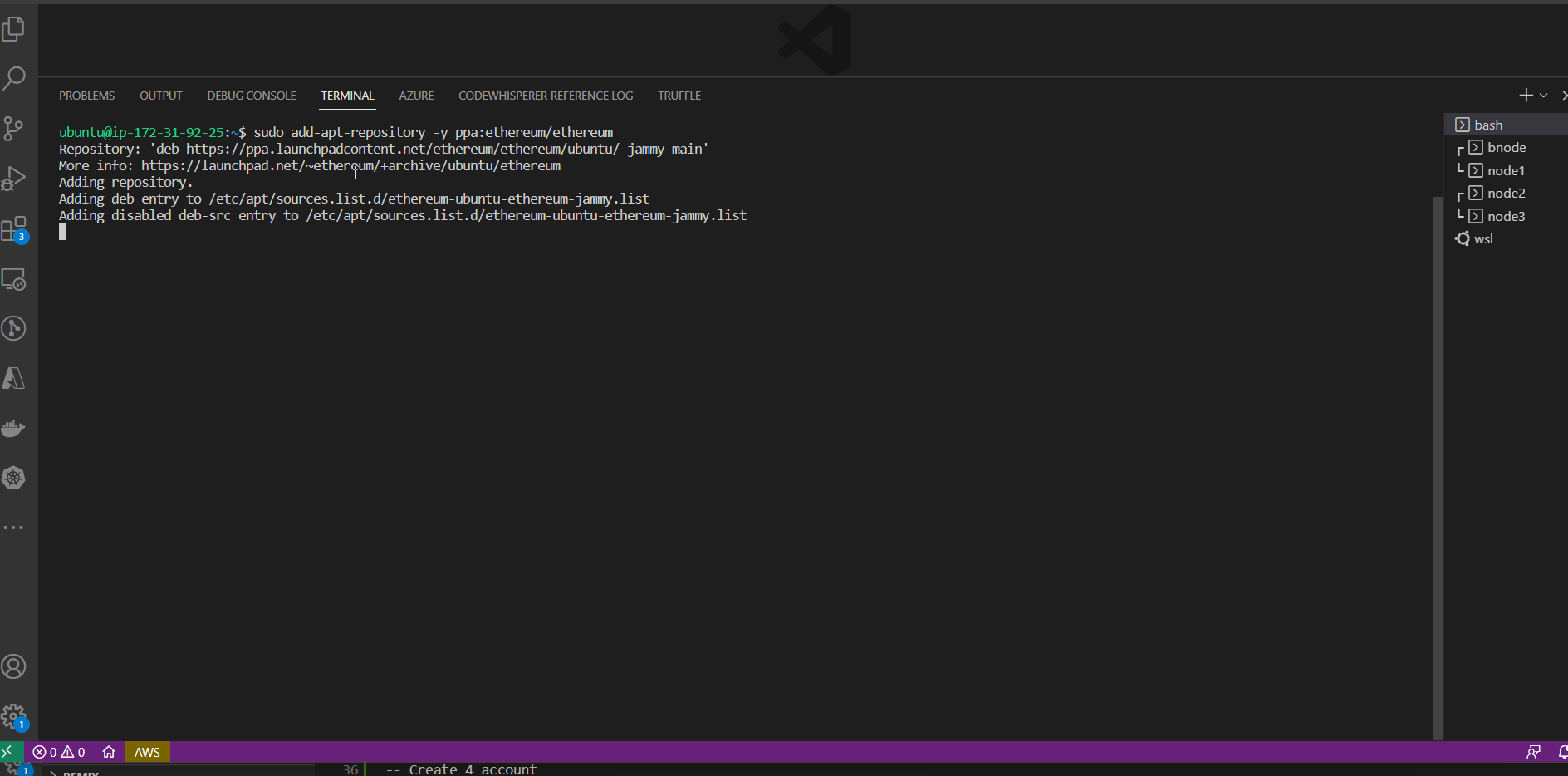


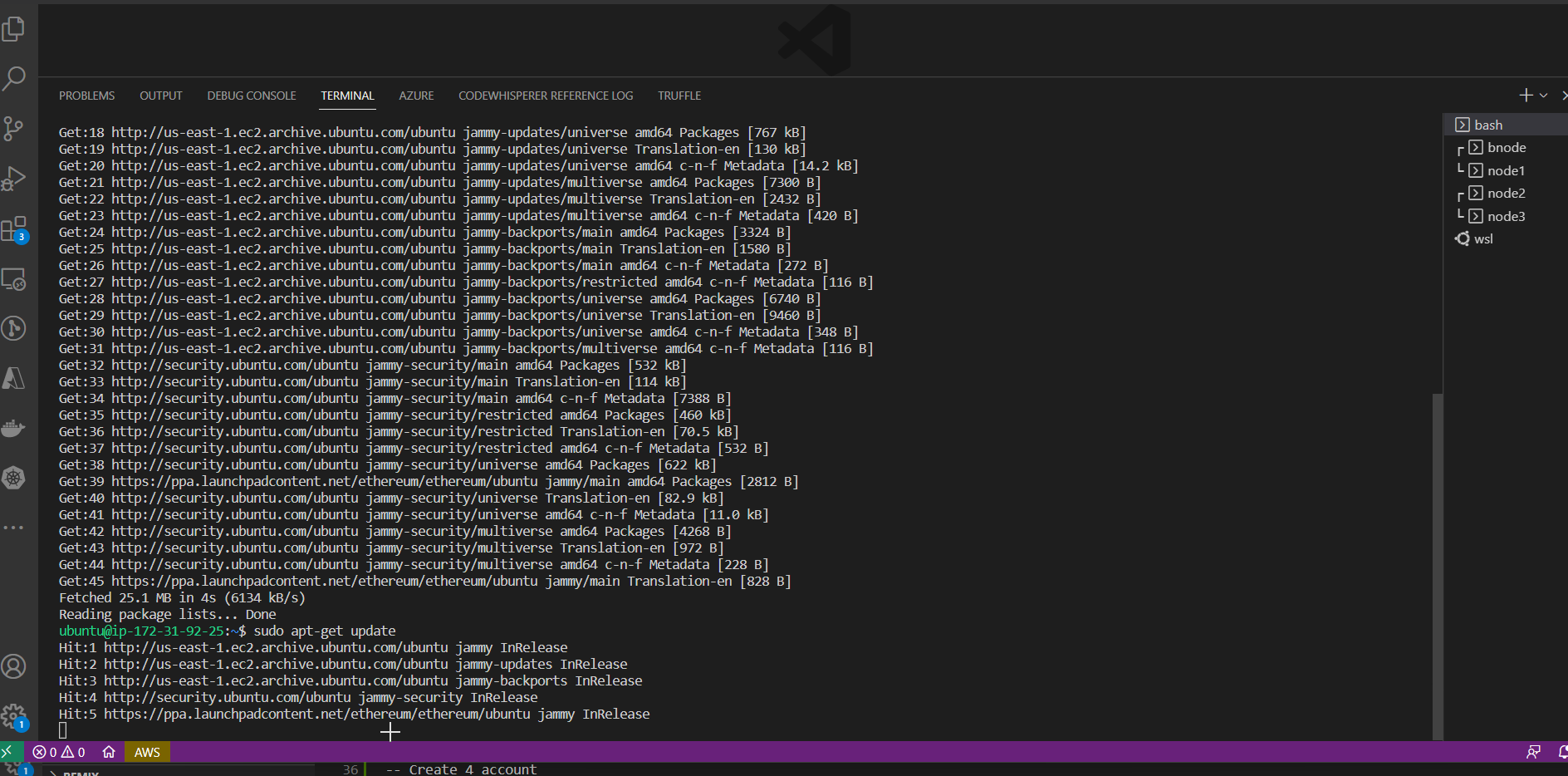
## Installing Ethereum on EC2 instance

After connecting to EC2 instance, we used below mentioned command for installing Ethereum.

* *sudo add-apt-repository -y ppa:ethereum/ethereum*
* *sudo apt-get update*
* *sudo apt-get install Ethereum*

We also installed Netstat tool to validate opening of ports, when Ethereum network is running





Text

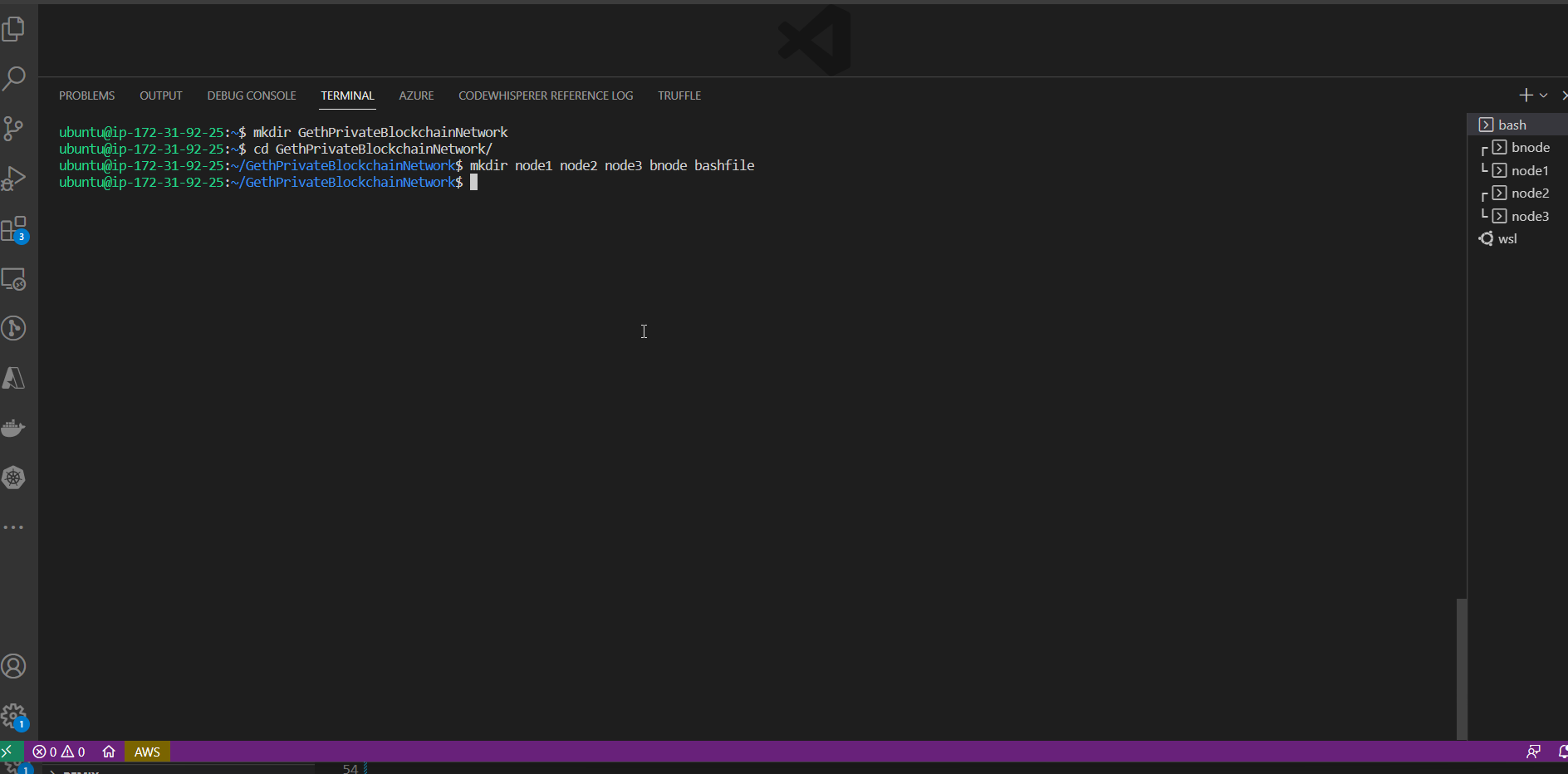
Description automatically generated

Text

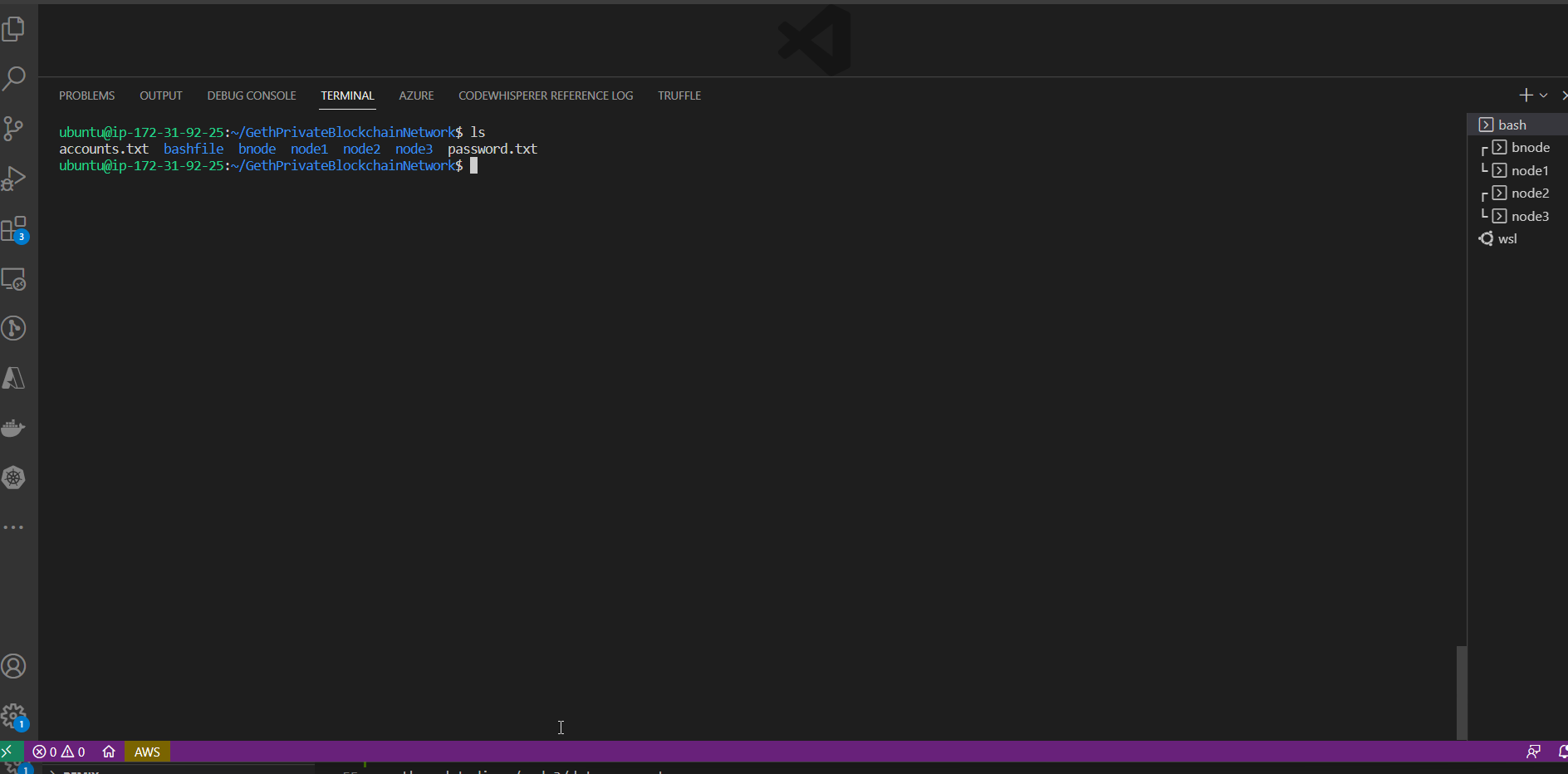
Description automatically generated

## Setup directory to create Private Network

In order to Setup 3 nodes network and a boot node, we have created the following directories.



We have also created two files, named accounts.txt and password.txt, to hold account related data.



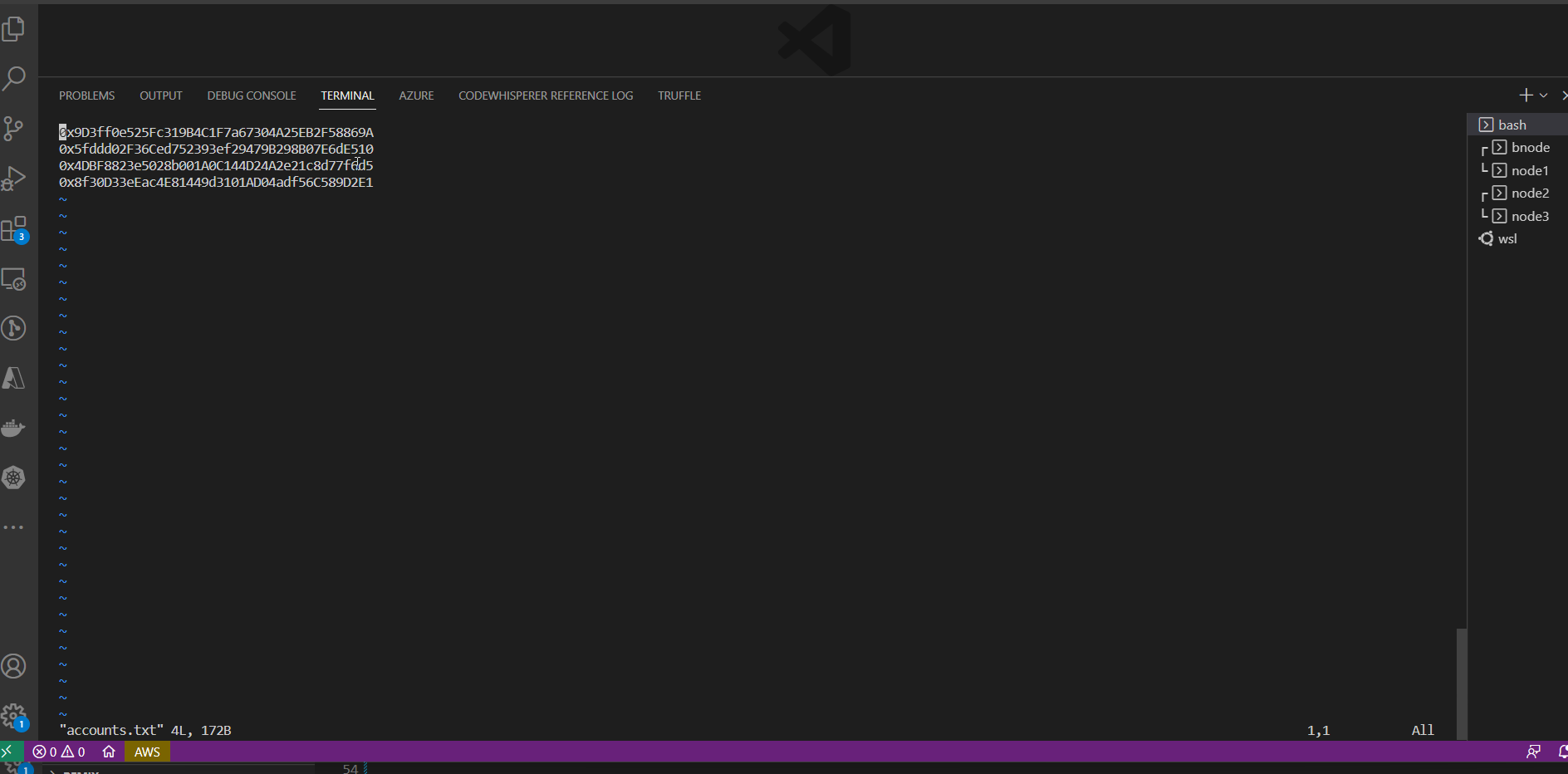
## Create Accounts

We created multiple accounts on all 3 nodes, 4 accounts for node1, 2 accounts for node2 and 2 accounts for node3.

A screenshot of a computer

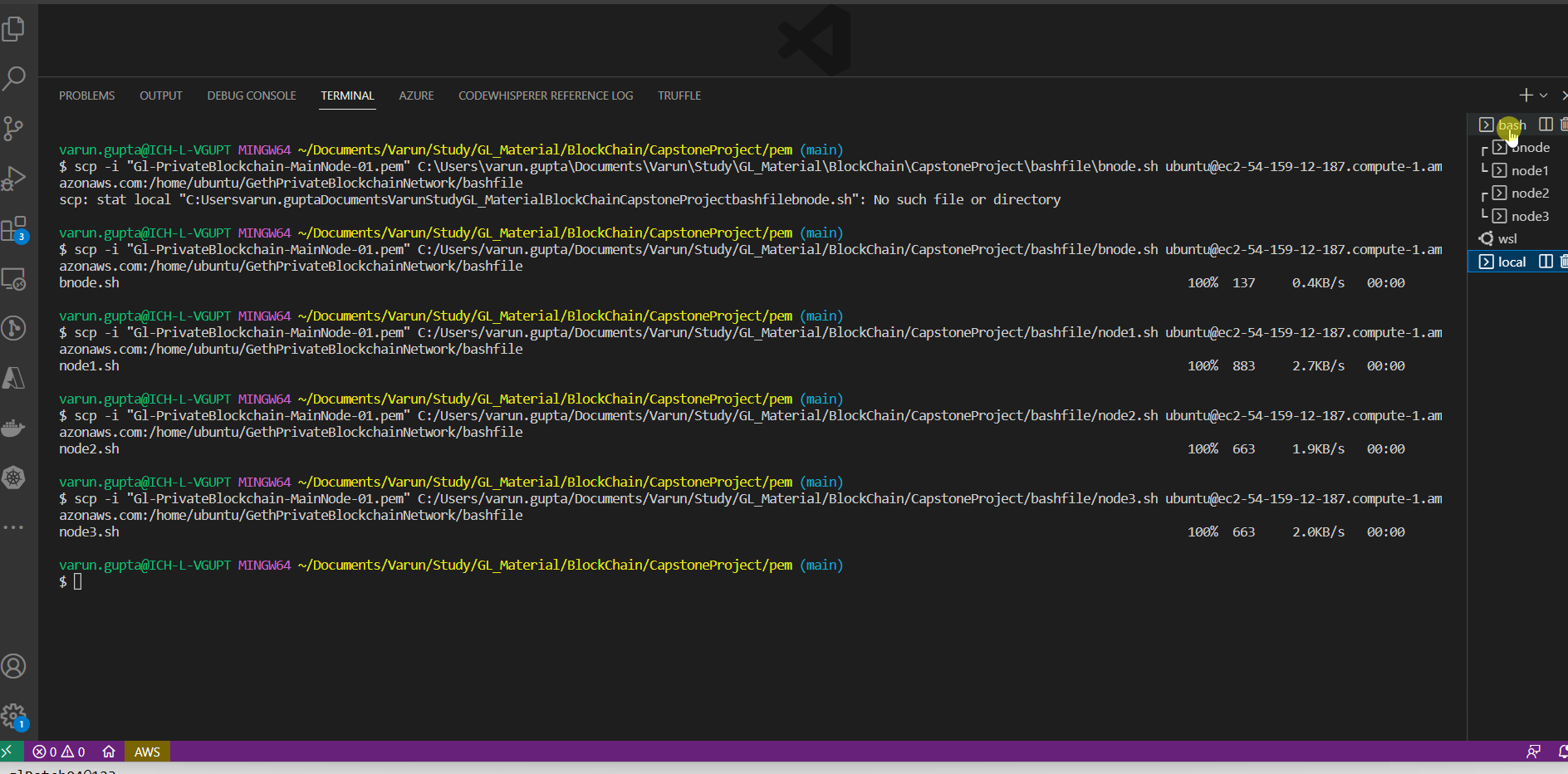
Description automatically generated with medium confidence

We stored these accounts details on accounts.txt file for future reference.



## Copy Bash files to server

We have created individual bash files for running the nodes and boot nodes. The node bash files accept 3 parameters, which are bootnode, private ip and unlock account. Similarly we have created a bash file to start boot node and this file excepts private ip address



## Create Bootnode

A screenshot of a computer

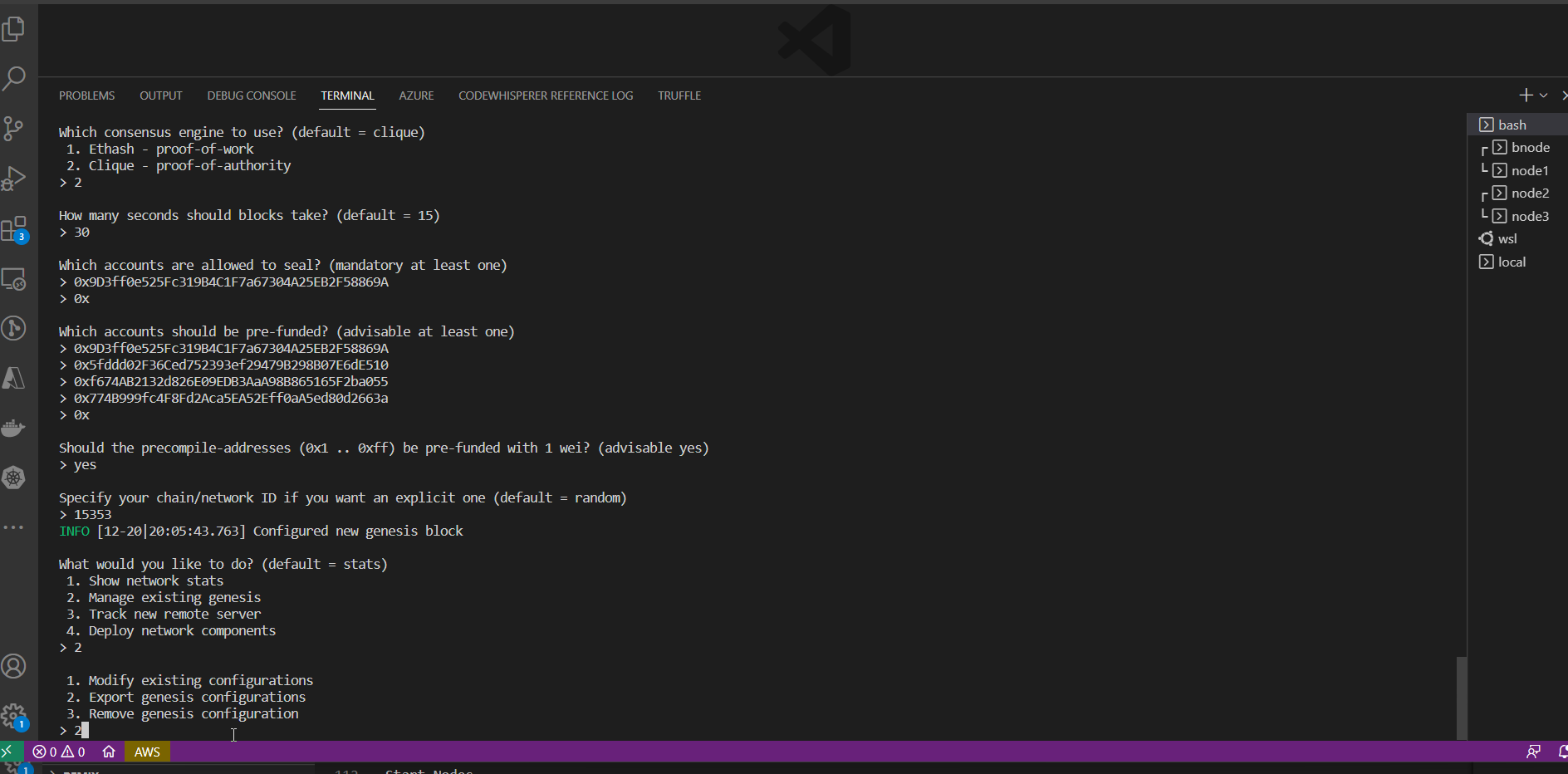
Description automatically generated with medium confidence

## Setup genesis block

We setup genesis block with Chain ID as 15353, Sealer account is node1’s account 1.

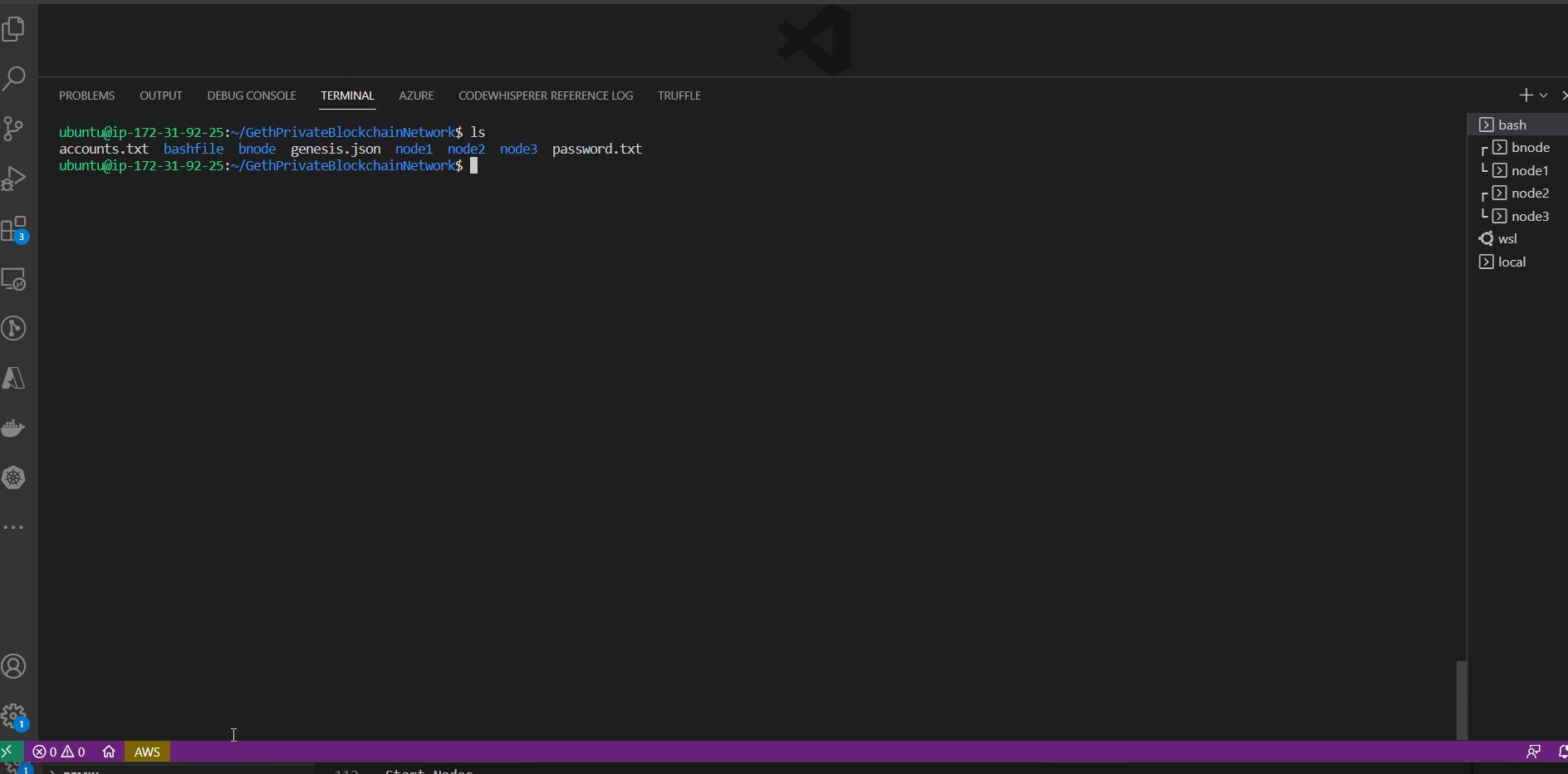
A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated with medium confidence



## Initialize node

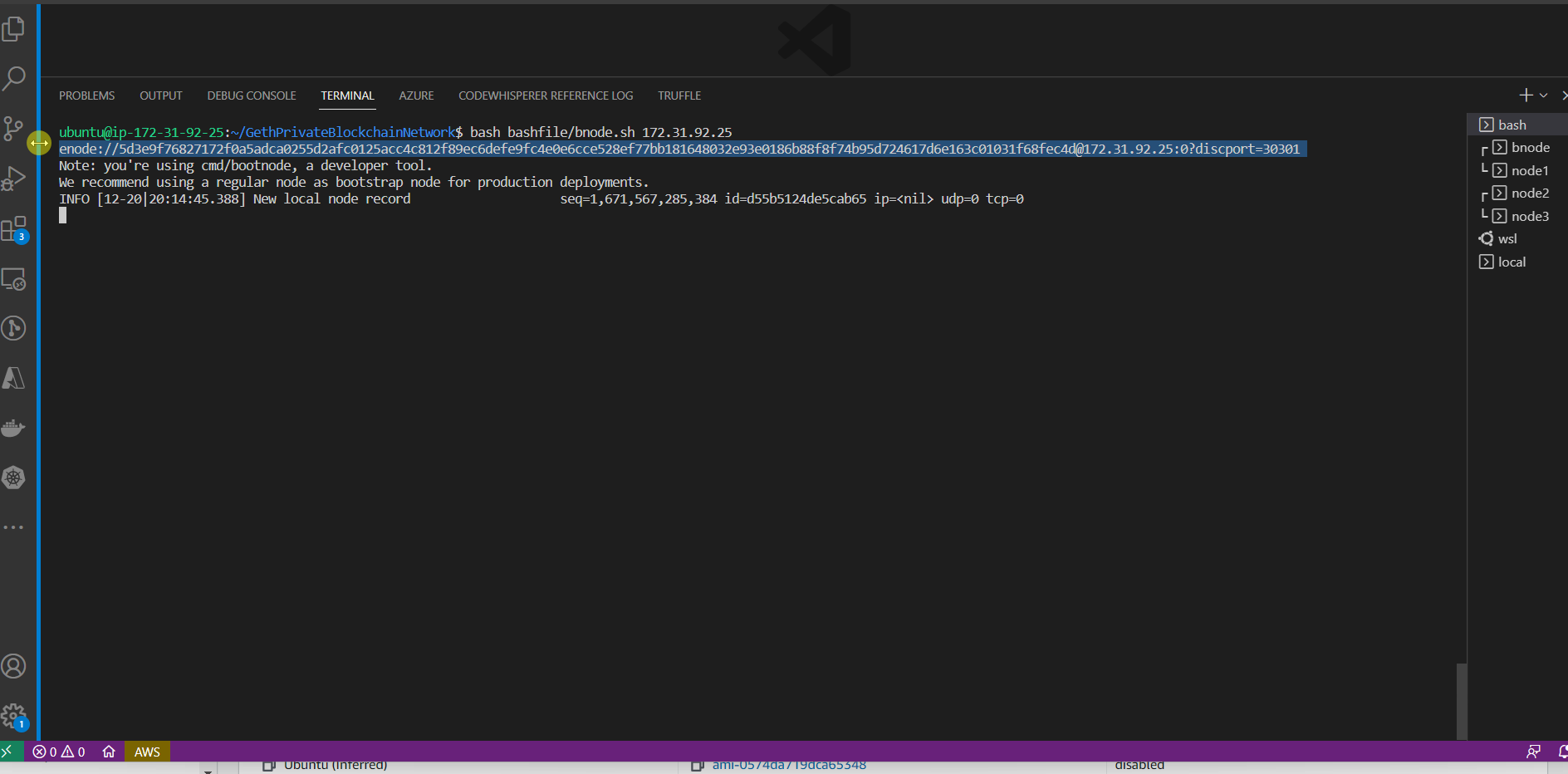
Text

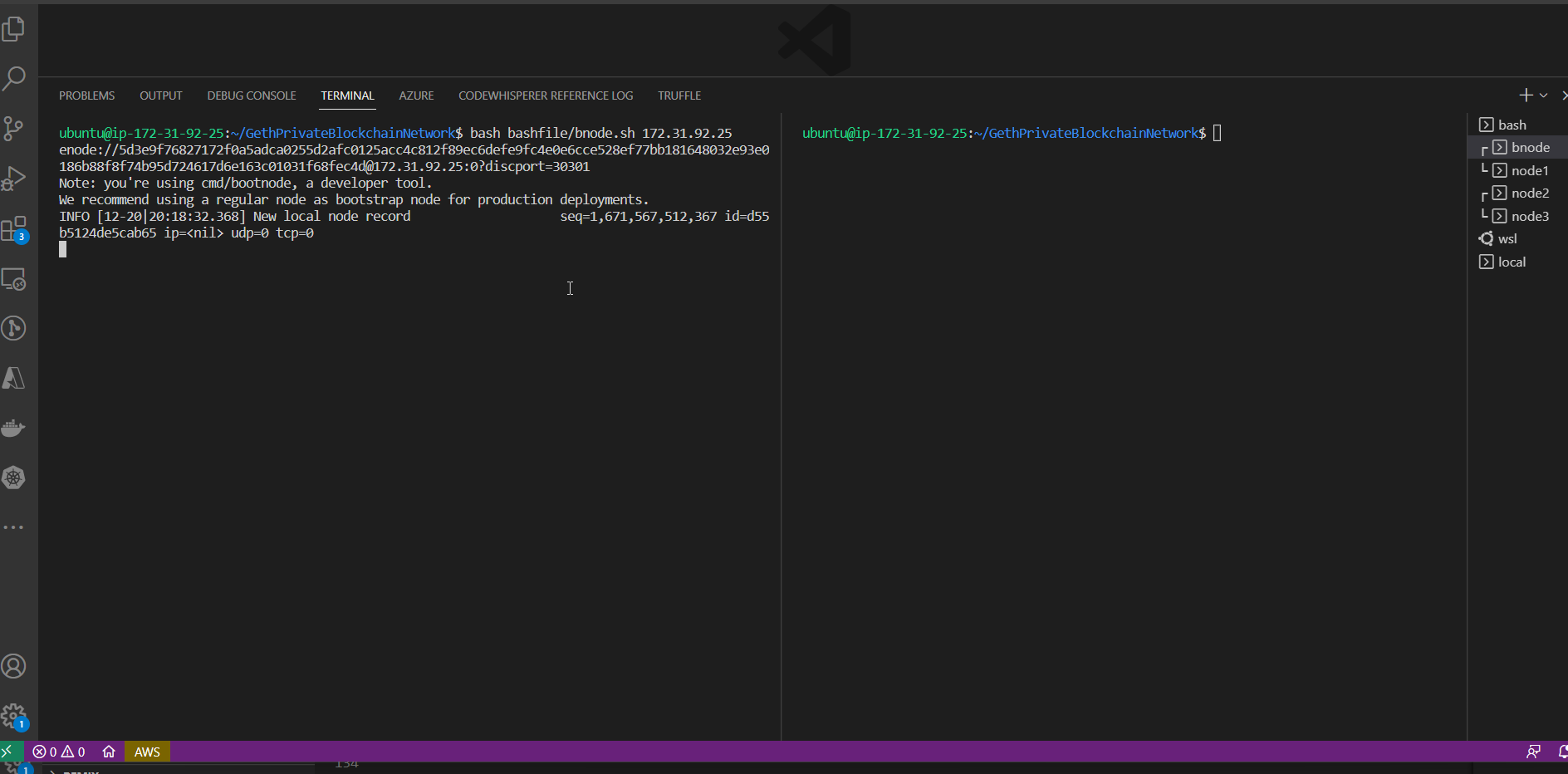
Description automatically generated with low confidence

## Start Bootnode using bash file

A screenshot of a computer

Description automatically generated with medium confidence

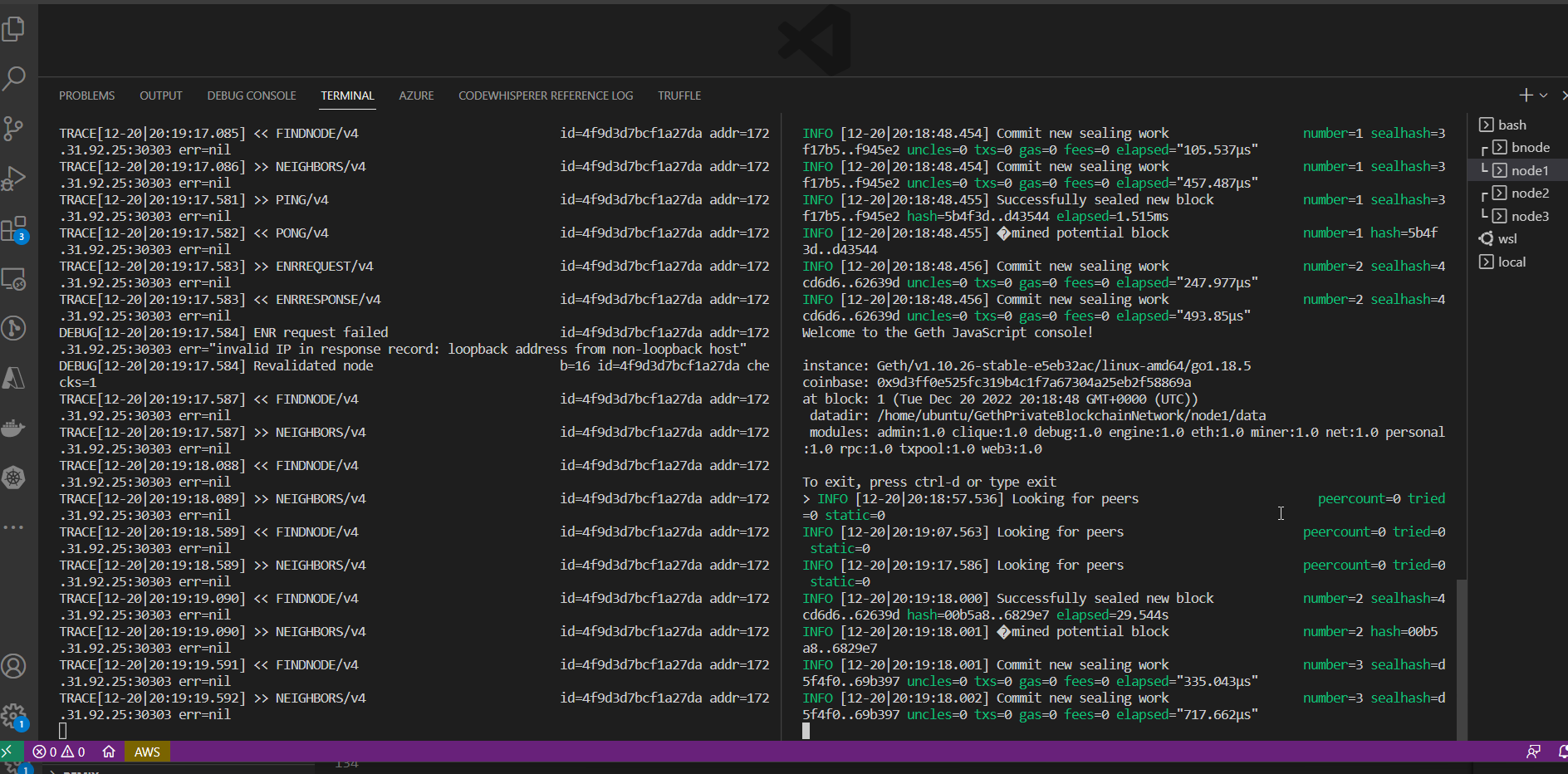




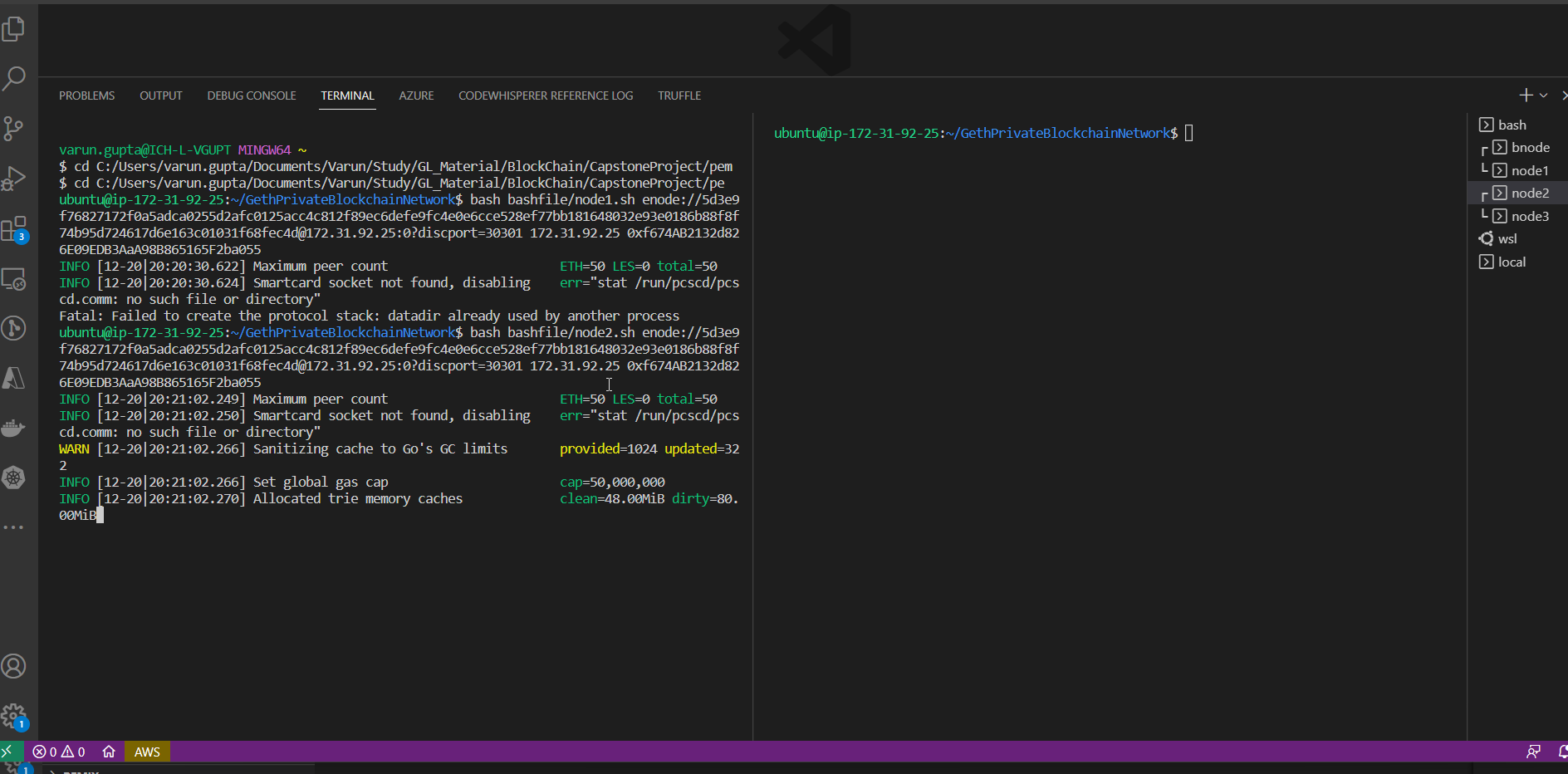
## Start Node1 using Bash file

A screenshot of a computer

Description automatically generated with medium confidence



## Start Node2 using bash file



Text

Description automatically generated

## Start Node3 using bash file

Graphical user interface, text

Description automatically generated

## Checking Peers of node1

Graphical user interface, text

Description automatically generated

## Peers count on node2 and node3

Graphical user interface, text

Description automatically generated

# Validation of Blockchain

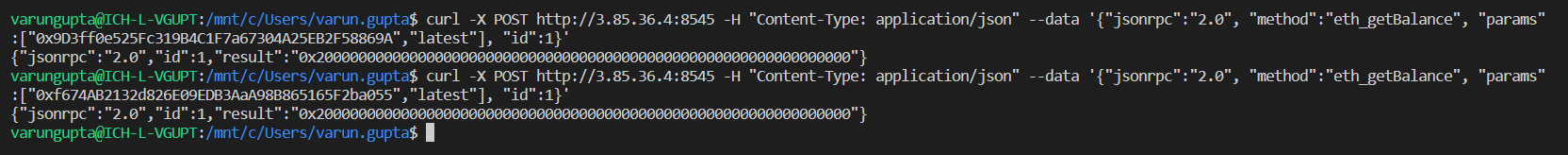
## Curl Operation on localhost

This exhibits the RPC connectivity with blockchain

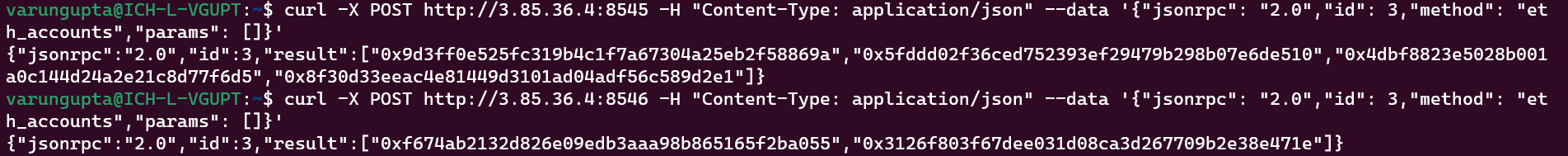


## Curl Operation from local System

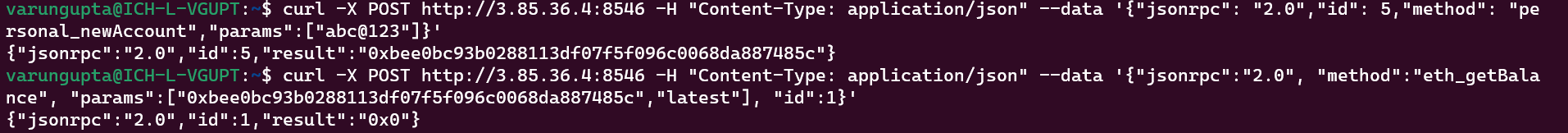
This exhibits the ability to connect to blockchain hosted remote networks.



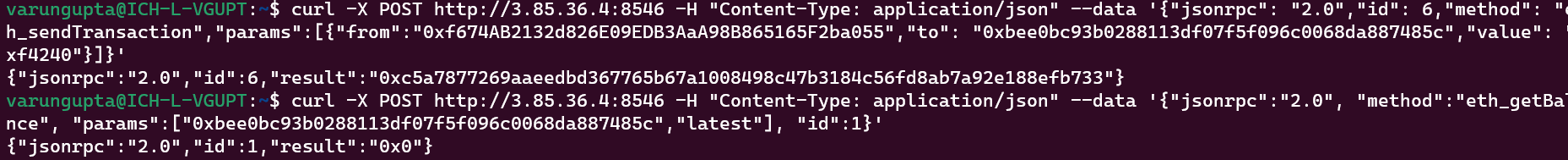
Fetching account details for node1 and node2



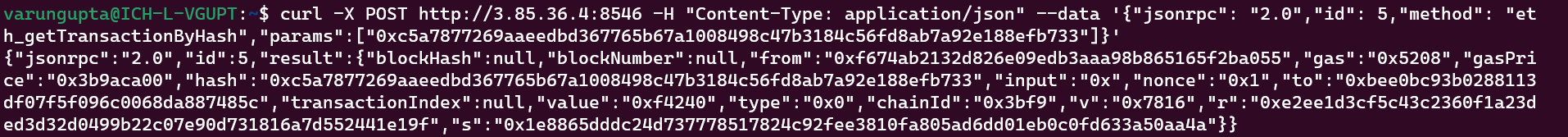
Creating a New Account from local system



Transferring funds from older account to newly created account



Transaction results



# Connection to local environment

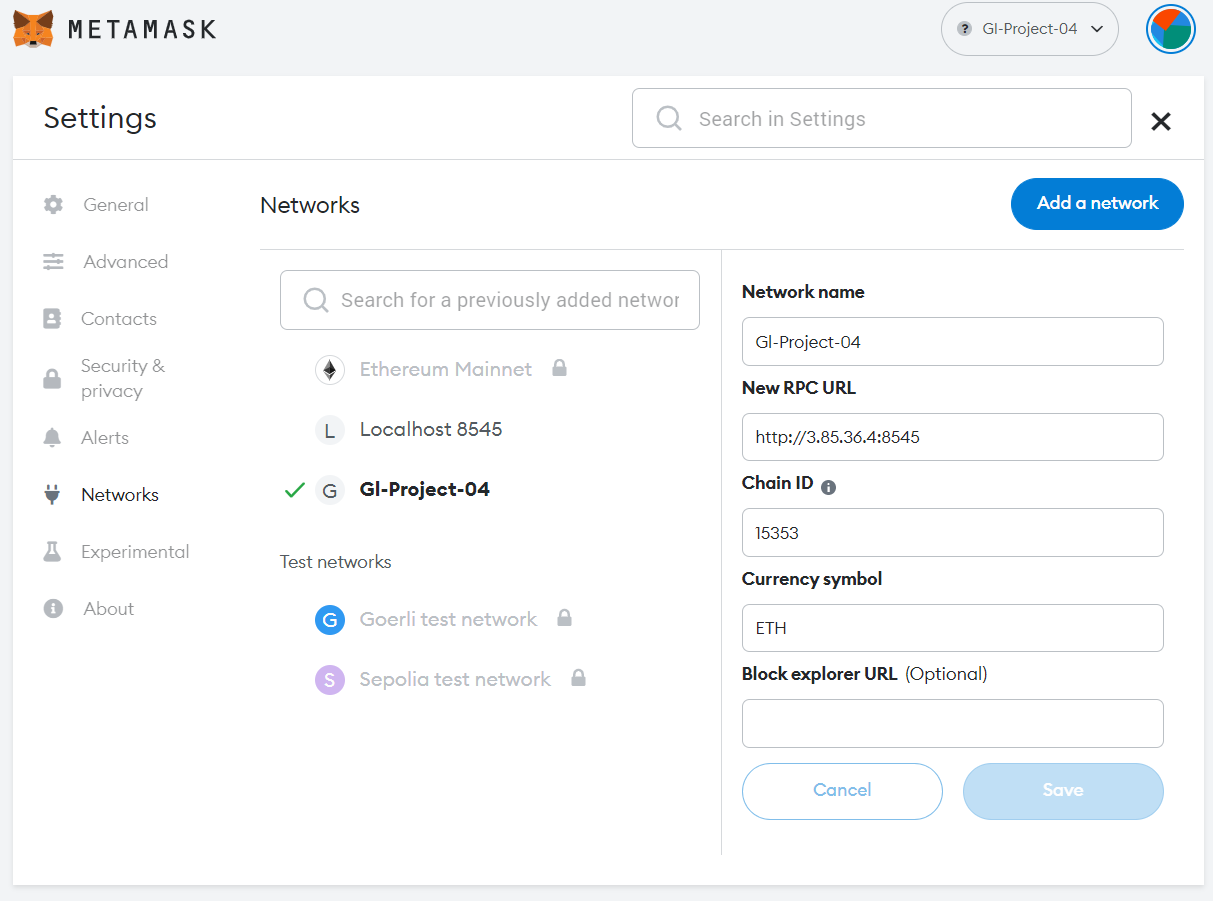
## Download keystore

We download the keystore files for each account, so that we could upload the same into Metamask

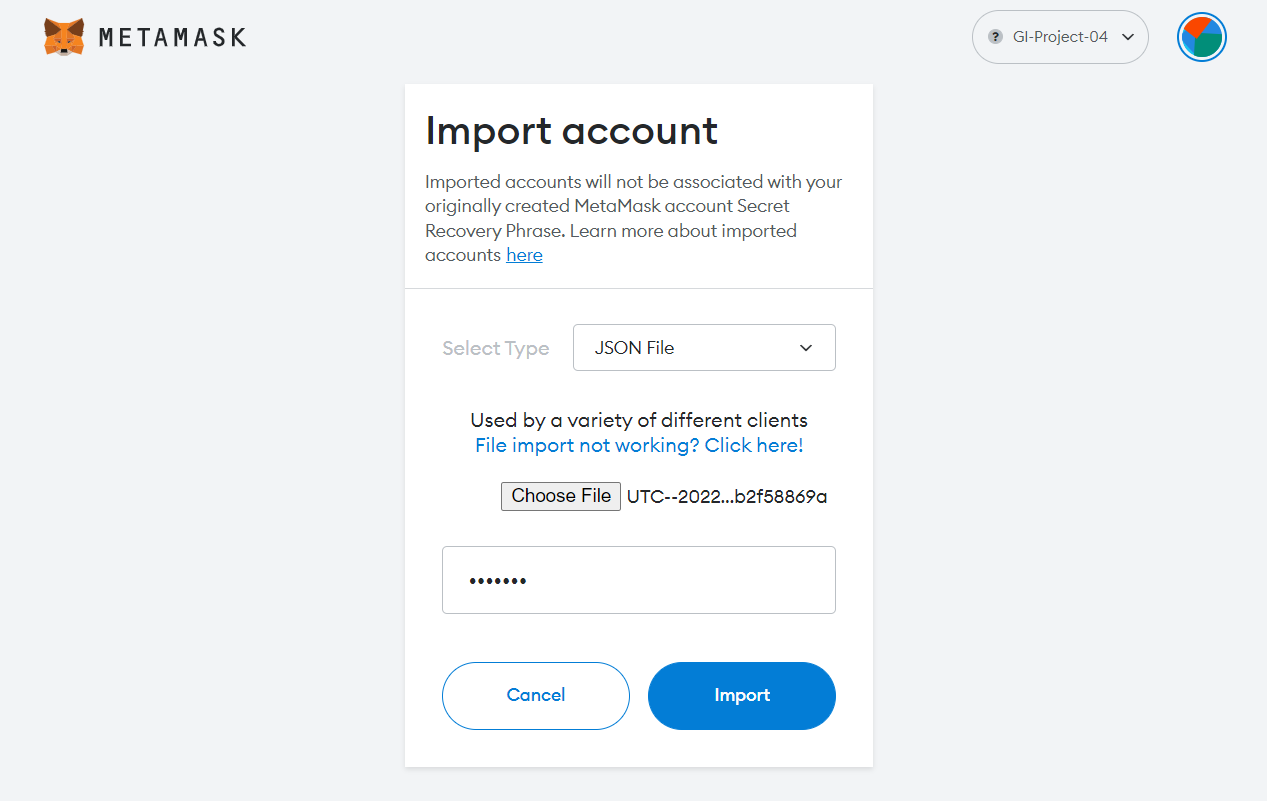
Text

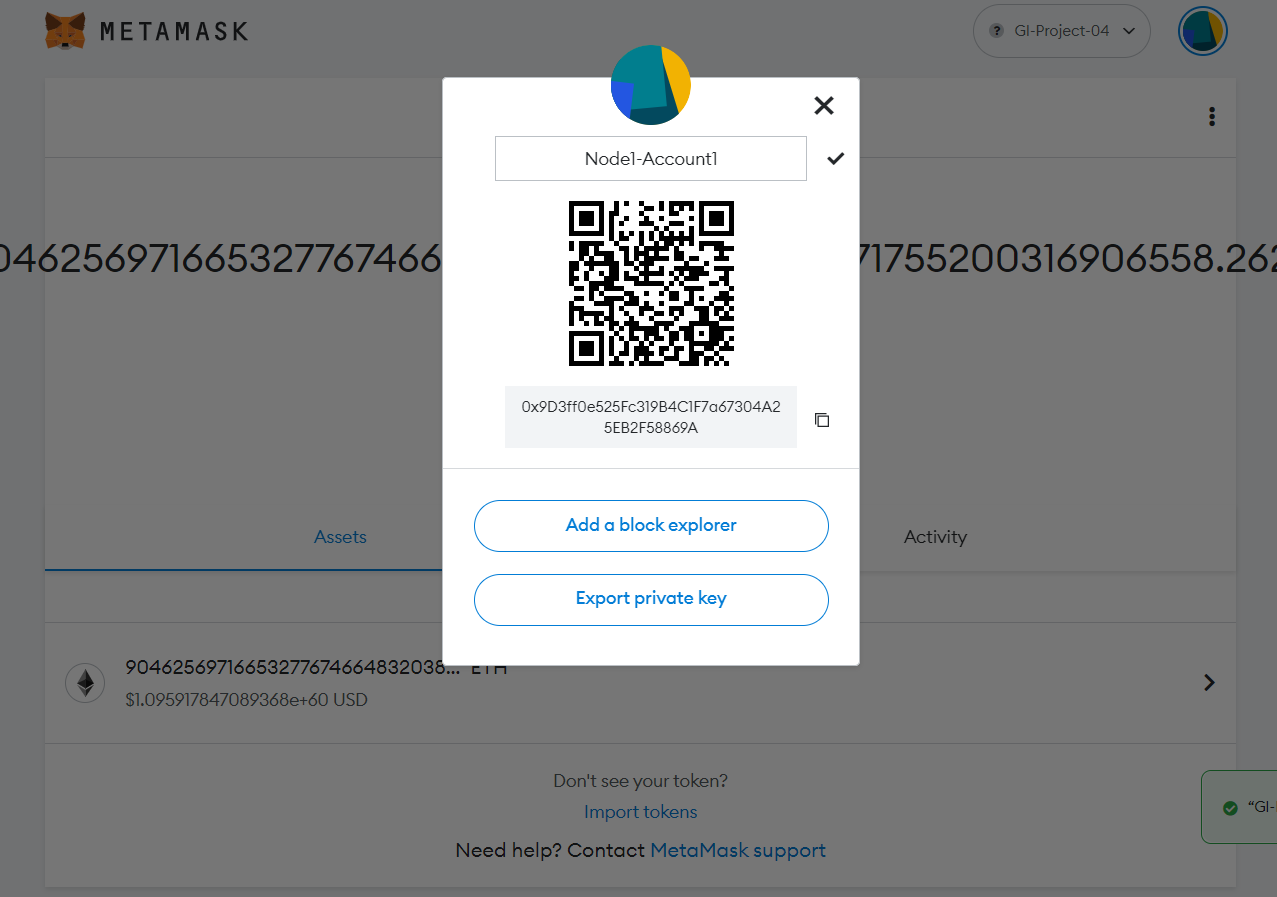
Description automatically generated

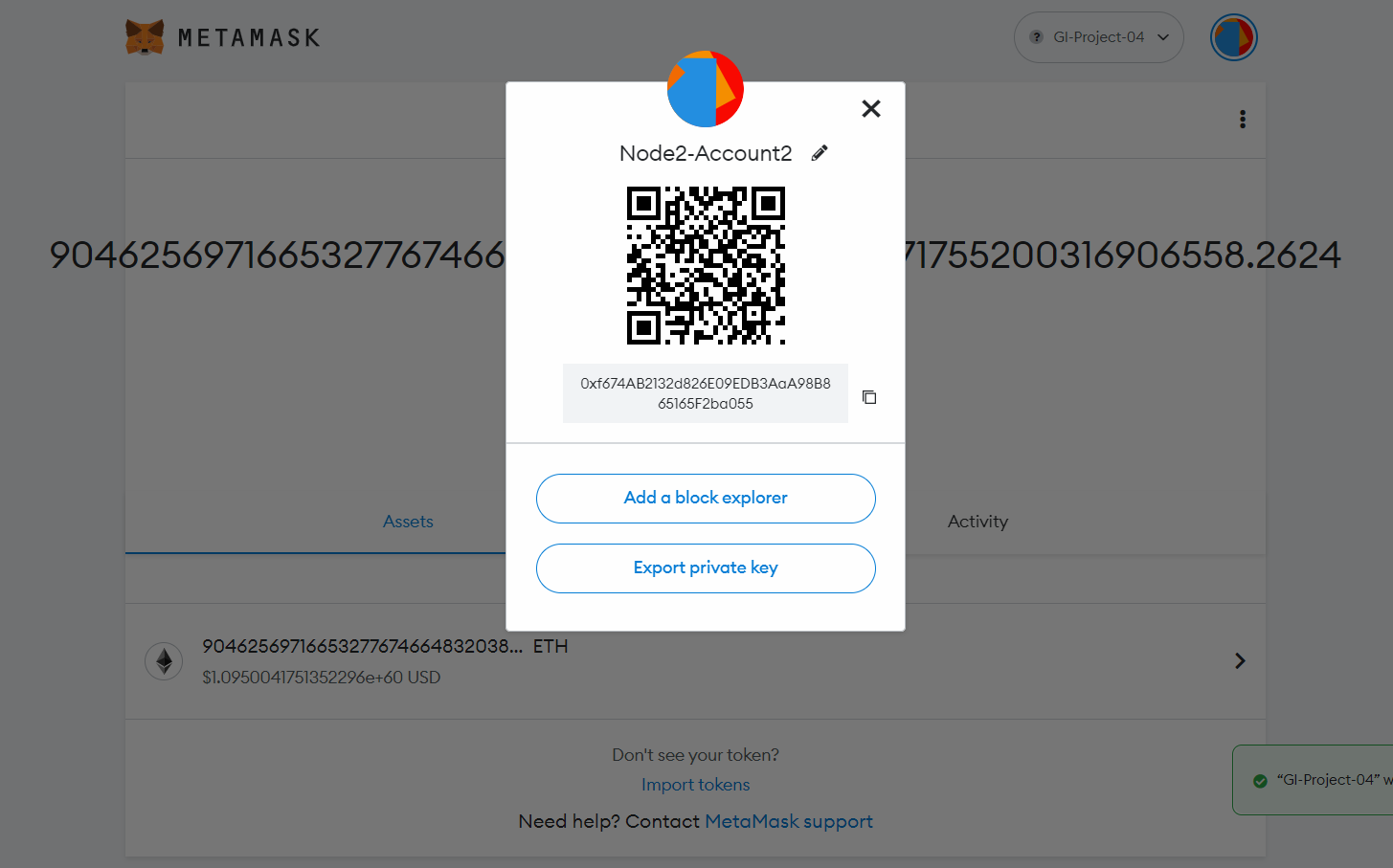
## Connecting Remote Blockchain to Metamask



## Importing accounts into Metamask using Keystore files







# Contracts

In order to resolve this issue, we have developed 3 contracts

1. FlightContract: This contract performs all operations related to one specific flight.
2. AirlineContract: This contract performs airline specific operation and creates instance of FlightContract and hold flights in mapping.
3. TransactionContract: This contract is for performing booking site related operation.

## Solution Architecture

Diagram

Description automatically generated

## Process Flow

Graphical user interface

Description automatically generated with medium confidence

## Flight Contracts

All operations pertaining to flight operation are handled by this contract. This maintains flight running and booking status. It also maintains guest list, price, availability and booked seat details.

### Running Status

**SCHEDULED**: At this phase the flight is only created and doesn’t allow booking. Flight details like Price, Number of seats are initialized at this stage

**DEPARTED**: At this stage flight is departed, and users are not allowed to perform any booking or cancellation operation

**LANDED**: At this stage we start calculation of concluded state, a window of 24 hours is provided to resolve any dispute.

**DELAYED**: Here we input delay time in Epoch time and users are compensated based on delay time

* + 2 hours delay: refund 10%
  + 6 hours delay: refund 20%
  + 12 hours delay: refund 40%
  + 24 hours delay: refund 80%
  + Beyond 24 hours: Cancel and refund 100%

**CANCELLED**: Here full refund is provided to all booked user

**CONCLUDED**: At this stage, the final amount is calculated, and the flight is deleted.

### Booking Status

**PRESALE**: At this stage users are not allowed to book tickets

**SALE**: At this stage, users are allowed to book ticket for specific flight

**READY**: This is 24 hours before departure. At this stage price of ticket is increased by 20%

**CLOSED**: At this stage, users are not allowed to perform any booking or cancellation operation.

**FINALISED**: This is landed stage, where users are provided 24 hours to raise dispute.

**CANCELLED**: At this stage refunds are provided to users.

## Airline Contracts

This contract can be treated as interface for airline operation.

**CreateAirline**: In this we create an airline with an airline address. Multiple airlines can be created, and this is maintained in mapping

**ScheduleFlight**: Here we created a flight, initializing flight with flight number, Schedule date and schedule time. Date and time are maintained as Epoch time.

**EnableFlight:** Here we enable flight and open it for user to book ticket for a flight. Airline provided details such as price and number of seats

**FlightOperation**: There are multiple functions created for airlines to perform flight operation like ready flight, delay flight, cancel flight.

### Transaction Contracts

This contract can be treated as booking sites.

**Book**: Allows user to book ticket to enabled flight

**Cancel**: Allows user to cancel their booked ticket

**GetPrice**: Allows users to get price of specific flight

**GetAvailabilty**: Allowed users to check available seats on a flight

**GetGuestByUuid**: fetch user seat number by unique id

**GetGuestBySeatNo**: this function fetches users uuid based on user seat number.