Setting up a private network on AWS using geth

Introduction

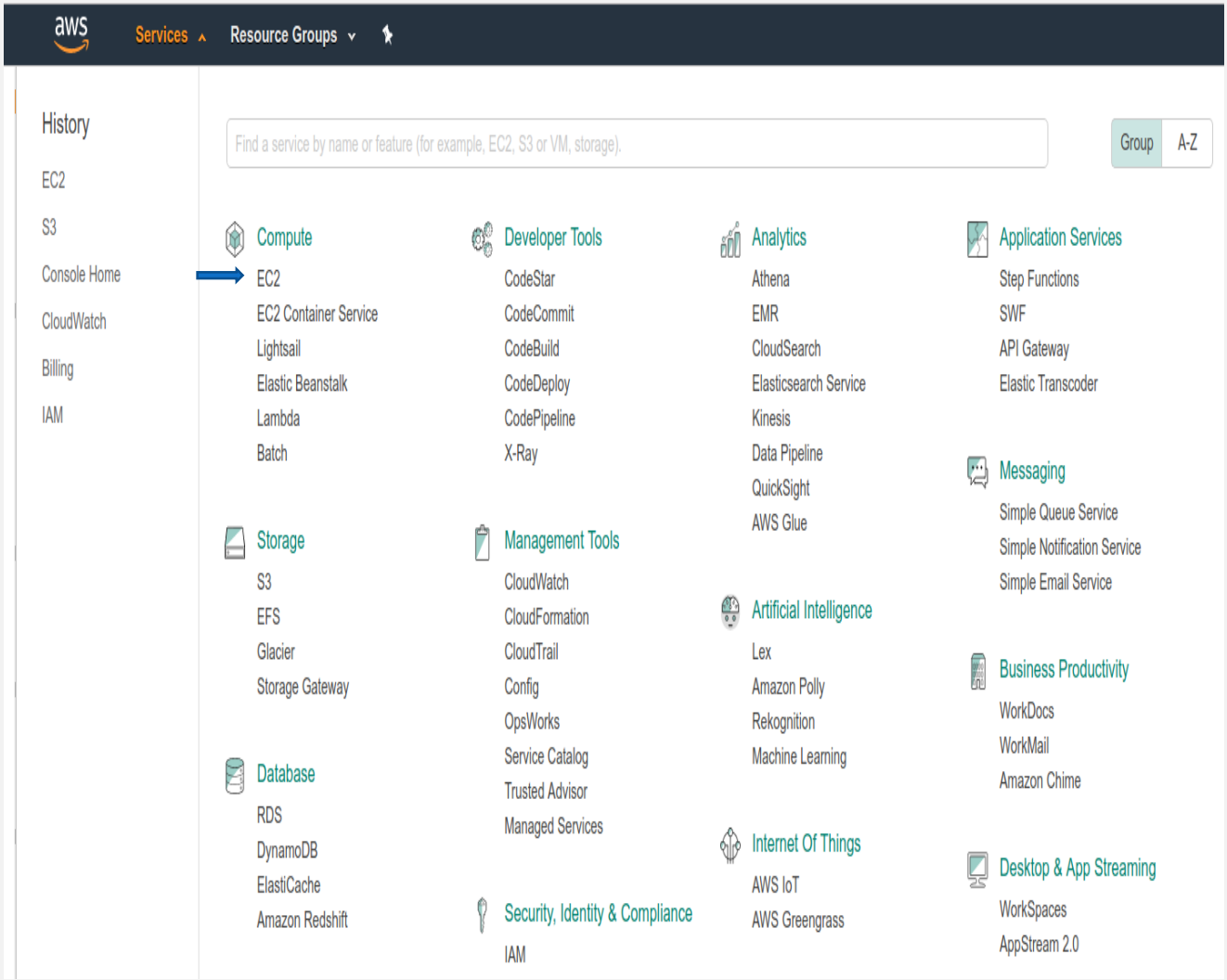
This is a step by step guide to setup a private Ethereum network on AWS using geth. This document is useful for anyone trying to create a private chain/testnet using geth

This document is divided into two parts:

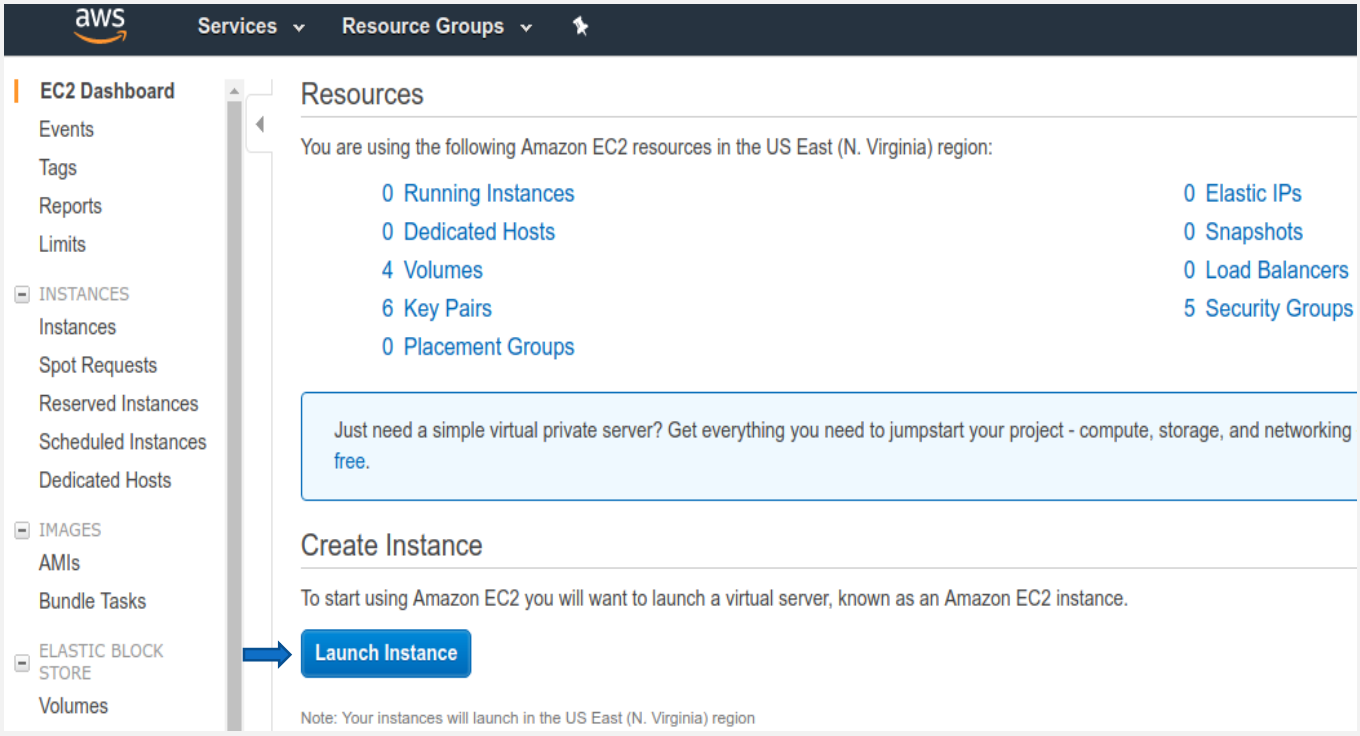
- Creating EC2 instance on AWS and installing geth
- Synchronizing this node with other nodes

1. Launching EC2 instance and installing geth

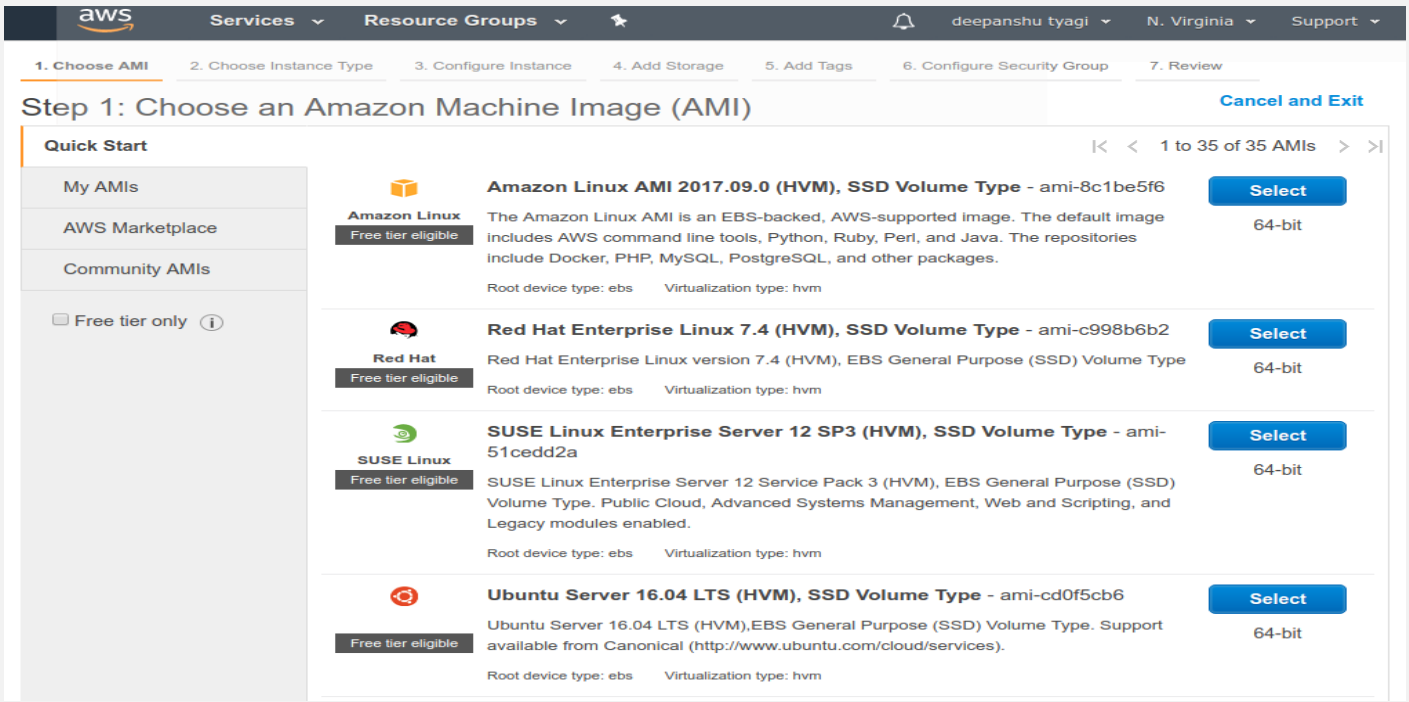
After logging into AWS console, select the region on the right-hand side, click on down arrow under services on the left-hand side, then select the EC2 under compute



Select launch instance under create instance which will take you to the OS selection screen



Select any of the amazon machine images depending on your OS requirements, in our case we choose ubuntu server



Choose instance type, we recommend using t2.medium (2 vCPU, 4 GB RAM) with default 8G SSD,we chose a medium server to have more memory and not crash during testing or development, but you can choose any size depending on your expected usage

aws Services Resource Groups deepanshu tyagi N. Virginia Support

1. Choose AMI | 2. Choose Instance Type | 3. Configure Instance | 4. Add Storage | 5. Add Tags | 6. Configure Security Group | 7. Review

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance types | Current generation | Show/Hide Columns

Currently selected: t2.medium (Variable ECUs, 2 vCPUs, 2.5 GHz, Intel Xeon Family, 4 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
■	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
	General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
	General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes

Click Next: Configure instance details in the bottom right hand corner, select the number of instances you want in our case we launch one instance first, select the availability zone under subnet and enable the accidental termination protection and cloudwatch detailed monitoring

aws Services Resource Groups deepanshu tyagi N. Virginia Support

1. Choose AMI | 2. Choose Instance Type | 3. Configure Instance | 4. Add Storage | 5. Add Tags | 6. Configure Security Group | 7. Review

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances 1 Launch into Auto Scaling Group

Purchasing option Request Spot instances

Network vpc-4c310335 (default) Create new VPC

Subnet No preference (default subnet in any Availability Zone Create new subnet

Auto-assign Public IP Use subnet setting (Enable)

IAM role None Create new IAM role

Shutdown behavior Stop

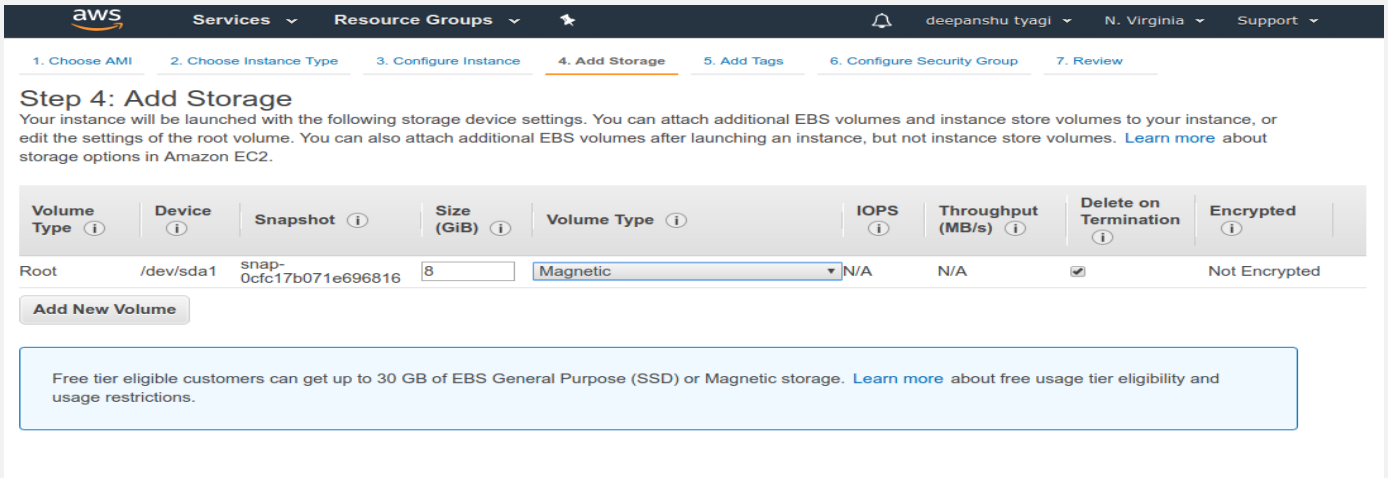
Enable termination protection Protect against accidental termination

Monitoring Enable CloudWatch detailed monitoring
Additional charges apply.

Tenancy Shared - Run a shared hardware instance
Additional charges will apply for dedicated tenancy.

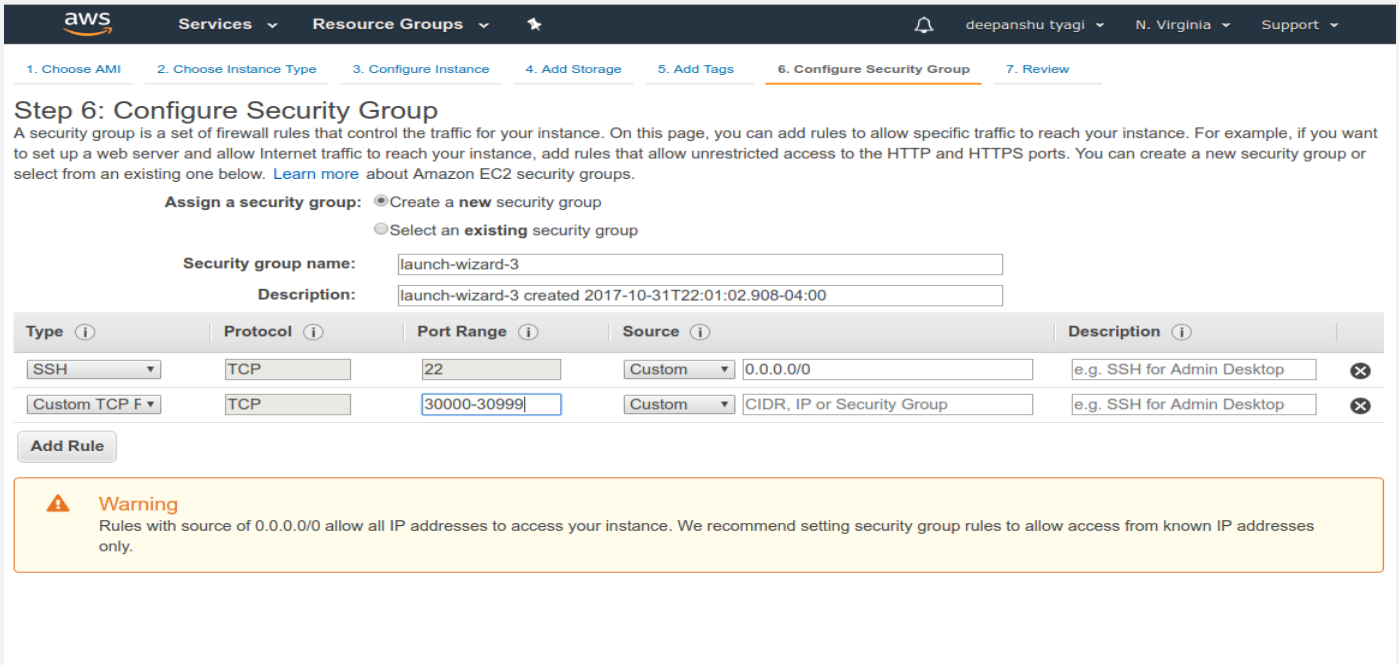
Advanced Details

The next step is to add storage we used 8GB standard and volume type as magnetic. You can either check the box to delete on termination or not. If it is checked it means that if you delete an EC2 instance, it will also delete the data on it. We recommend this option to avoid storing extra data in the cloud



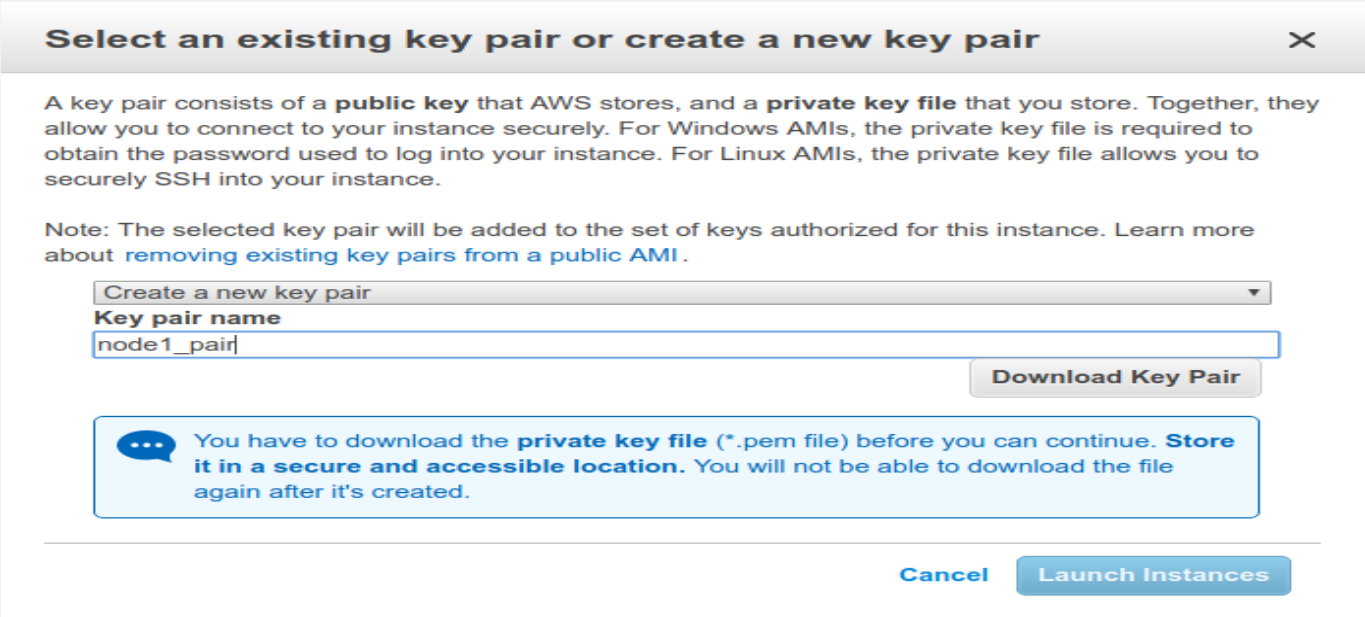
1.1 Setup Security Groups

Setting up a security group on an Amazon EC2 instance is done to let the server know which ports to accept inbound or outbound data from. First you need to name your security group and include a description. Make sure that both instances are under same security groups which allows Custom TCP (30000-30999) as in our case. Use source to restrict access to custom IPs, in our case we use anywhere with default IP as 0.0.0.0/0, ::/0.



1.2 Setup Key Pair

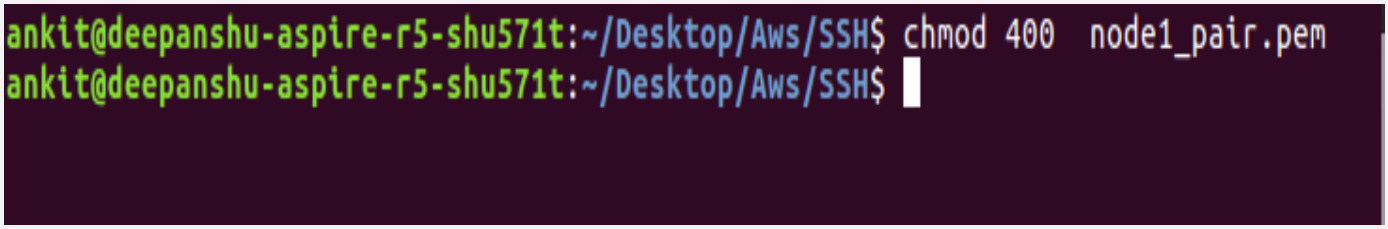
After reviewing all the information, as you launch your EC2 instance, you will see an alert show up to either select an existing key pair or to create a new key pair. Setting up your key pair is to enable you to securely SSH into your instance, while ensuring that others cannot. As shown in the image, your key pair consists of a public key which AWS stores and a private key which you store on your computer. Make sure to download your private key to SSH into your instance later when needed



Now that you have completed activating your cloud server and you have downloaded your key pair you should be shown a screen to confirm your Instance has been setup properly and an ID for your Instance. It might take a few minutes to complete loading, but be patient as you need it to complete for it to return the public IP that enables you to connect to your cloud instance. Once it is completed loading and it has returned your public IP you are able to SSH into your EC2 from your terminal, using the key pair that you just created

1.3 SSH into you EC2 instance and install geth

In order SSH into your EC2 instance we use chmod 400 command to lock your private key and change permission on keypair file



From your terminal, change directory to the folder that has the key pair and input the command below. Type yes, then click enter, and your terminal will respond that you have permanently added your public IP to the list of known hosts, which will give access to your amazon machine image

```
ankit@deepanshu-aspire-r5-shu571t:~/Desktop/Aws/SSH$ ssh -i node1_pair.pem ubunt
u@54.161.85.43
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-1039-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

60 packages can be updated.
0 updates are security updates.


Last login: Wed Nov  1 00:32:36 2017 from 100.1.88.43
ubuntu@ip-172-31-90-212:~$
```

Now you are logged into your geth node1 on AWS, make a directory to store the private chain data and install geth using the the following commands. We named node1 as our chain directory

```
sudo apt-get install software-properties-common
sudo add-apt-repository -y ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install ethereum
```

You can check the installed geth version on you amazon machine image using following command

```
ubuntu@ip-172-31-90-212: ~/node1
ubuntu@ip-172-31-90-212:~/node1$ geth --version
Incorrect Usage. flag provided but not defined: -version

NAME:
   geth - the go-ethereum command line interface

   Copyright 2013-2017 The go-ethereum Authors

USAGE:
   geth [options] command [command options] [arguments...]

VERSION:
   1.7.2-stable-1db4ecdc

COMMANDS:
```

Every blockchain needs a genesis block. For private network you need to use a different genesis file. Initialize geth with genesis.json file in node1 data directory, the json file configuration should be edited as per the private network requirements. Both the nodes on amazon machine must be initialized with the genesis file having similar configurations. This Genesis.json file was used to initialize the private blockchain

```
{
  "config": {
    "chainId": 33,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
  },
  "nonce": "0x0000000000000033",
  "timestamp": "0x0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "gasLimit": "0x8000000",
  "difficulty": "0x100",
  "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x3333333333333333333333333333333333333333",
  "alloc": {}
}
```

We used aws s3 service to first upload the genesis.json file and then used wget to upload the genesis file to node1 directory on our AMI

```
ubuntu@ip-172-31-90-212:~/node1$ wget https://s3.amazonaws.com/geth1/genesis.json
--2017-10-31 22:39:27--  https://s3.amazonaws.com/geth1/genesis.json
Resolving s3.amazonaws.com (s3.amazonaws.com)... 54.231.114.250
Connecting to s3.amazonaws.com (s3.amazonaws.com)|54.231.114.250|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 460 [application/json]
Saving to: 'genesis.json'

genesis.json        100%[===================>]     460  --.-KB/s    in 0s

2017-10-31 22:39:27 (10.8 MB/s) - 'genesis.json' saved [460/460]
```

Initialize the private chain using geth in node1 data directory on amazon machine image as shown below using the following command

```
ubuntu@ip-172-31-90-212:~/node1$ geth init genesis.json
WARN [11-01|01:23:02] No etherbase set and no accounts found as default
INFO [11-01|01:23:02] Allocated cache and file handles         database=/home/ub
untu/.ethereum/geth/chaindata cache=16 handles=16
INFO [11-01|01:23:02] Successfully wrote genesis state         database=chaindat
a                                 hash=5704d0…9bc5b0
INFO [11-01|01:23:02] Allocated cache and file handles         database=/home/ub
untu/.ethereum/geth/lightchaindata cache=16 handles=16
INFO [11-01|01:23:02] Successfully wrote genesis state         database=lightcha
indata                            hash=5704d0…9bc5b0
```

Repeat the steps above to create a second ec2 instance and install geth on the same. We named our second instance on amazon machine as geth node 2 and chain data directory in second instance as node2

```
ubuntu@ip-172-31-92-159:~/node2$ geth init genesis.json
WARN [11-01|01:20:27] No etherbase set and no accounts found as default
INFO [11-01|01:20:27] Allocated cache and file handles         database=/home/ub
untu/.ethereum/geth/chaindata cache=16 handles=16
INFO [11-01|01:20:27] Successfully wrote genesis state         database=chaindat
a                              hash=5704d0…9bc5b0
INFO [11-01|01:20:27] Allocated cache and file handles         database=/home/ub
untu/.ethereum/geth/lightchaindata cache=16 handles=16
INFO [11-01|01:20:27] Successfully wrote genesis state         database=lightcha
indata                             hash=5704d0…9bc5b0
```

2. Make sure that both the nodes are synchronized

There are different ways of node synchronization on private network, I have used one of the many possible ways. After initializing the client with genesis file in node1 and follow the steps below to get the node information and to know the number of peers that are connected to this node. The command below opens the JavaScript console, geth provides us JavaScript runtime environment to build and deploy applications on top of it. Get the node information using the command below

```
ubuntu@ip-172-31-90-212:~/node1.1$ geth --nodiscover console 2>>eth.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.7.2-stable-1db4ecdc/linux-amd64/go1.9
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txp
ool:1.0 web3:1.0

> admin.peers
[]
> admin.nodeInfo.enode
"enode://ea17a1bd571d847e581d292b9897b96c8989311488df718bb8aa32081c563102f7ba580
a31fd1bc9cd64552a3d1ac55f5baba367d74d1e81409f7aba8433afdc@[::]:30303?discport=0"
>
>
```

There are no peers connected to the node1 currently, using enode info we can connect our geth node2 to geth node1. Replace [::] in enode info with the private IP address of geth node1 and using the command **admin.addpeer("enodeinfo"**), add node1 as a peer of node2. We can public use ip address but to avoid connecting to node1 every time we start and stop our instance we use private ip address

```
> admin.addPeer("enode://ea17a1bd571d847e581d292b9897b96c8989311488df718bb8aa320
81c563102f7ba580a31fd1bc9cd64552a3d1ac55f5baba367d74d1e81409f7aba8433afdc@172.31
.92.159:30303?discport=0")
true
```

Note that after admin.addPeer() on node2, two nodes are synchronized. We do not need to do the same thing on node1. After peering, we see Block synchronization has started