Building a simple IRC with rust

I was recently invited as tech speaker to sahyadri engineering college to conduct a workshop on rustlang and to make use of AWS if possible.
And I then began to think of right use case that can show the capablity of rustlang along with its suitability with AWS. Initial thoughts are to make use of AWS lambda and trigger a rustlang function, but felt that it is not enough and useful to the students straigtaway. Suddenly struck to my mind, is to build an IRC server using rocket web framework for both client and the server.

Why IRC?

Unlike stackoverflow, an IRC(Internet Relay Chat) is a realtime **group chat platform** where anyone could pose a question or start a discussion and instantly get helped by the community. On the bright side, It is one kind of knowledge sharing platform where people can collaborate and learn from each other.

There are two parts for an IRC,
one is the server which holds metadata of the clients registered to it(kind of nameserver) and also takes responsibility for broadcasting the messages.
Second is the client side, where it stores the session id and also message history.

Infra setup:

- An AWS account with some credits, as we will be using a non-free tier centos VM.
- Postman client
- SSH client (putty for windows)

APIs implemented

Server side APIs:

```
=> GET /
=> GET /register/<name>/<ip>
=> POST /broadcast
=> GET /logout/<id>/<name>/<ip>
```

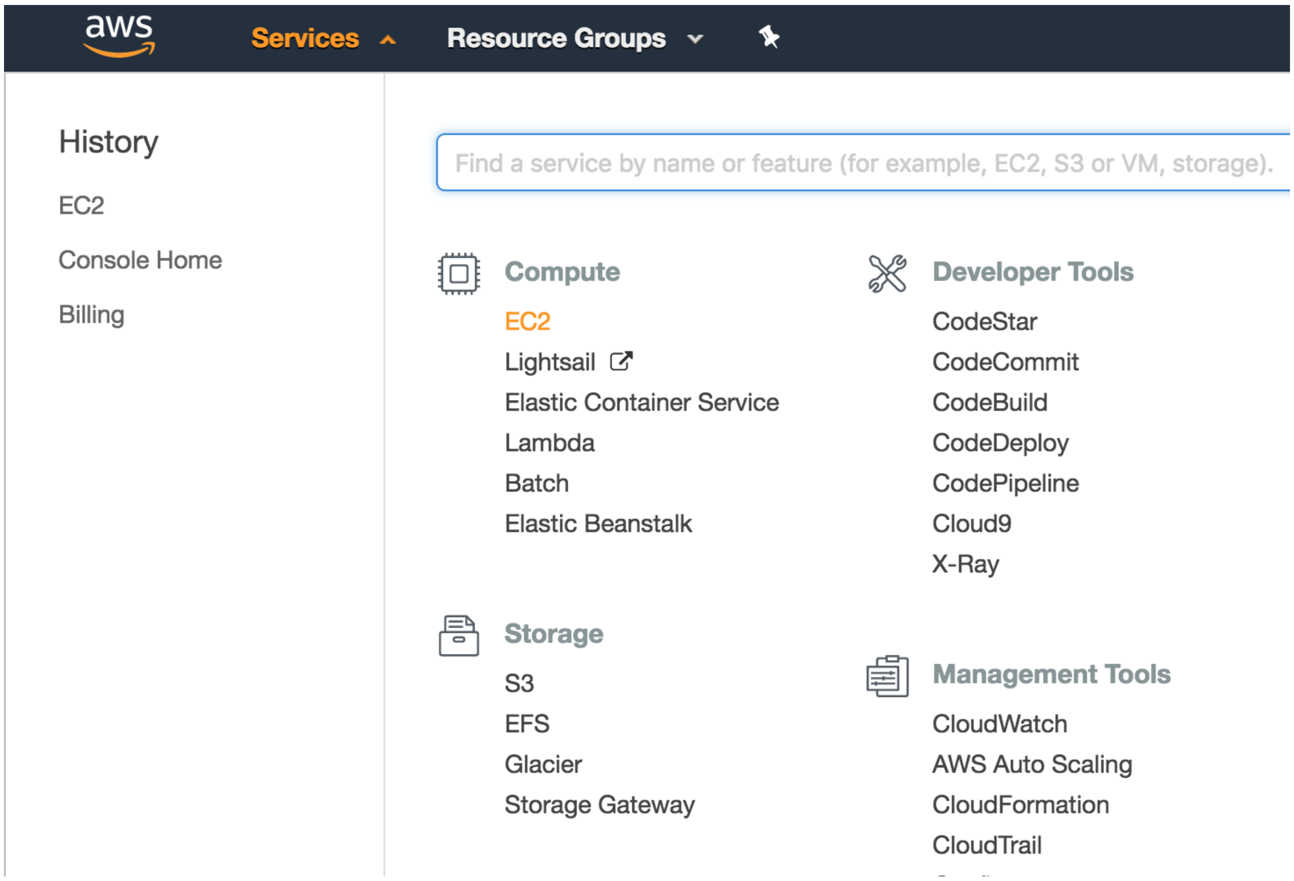
Client side APIs:

```
=> GET /
=> GET /<user>
=> GET /register/<name>
=> GET /send/<message>
=> GET /receive/<user_name>/<message>/<time>
=> GET /get/messages/<count>
=> GET /logout
```

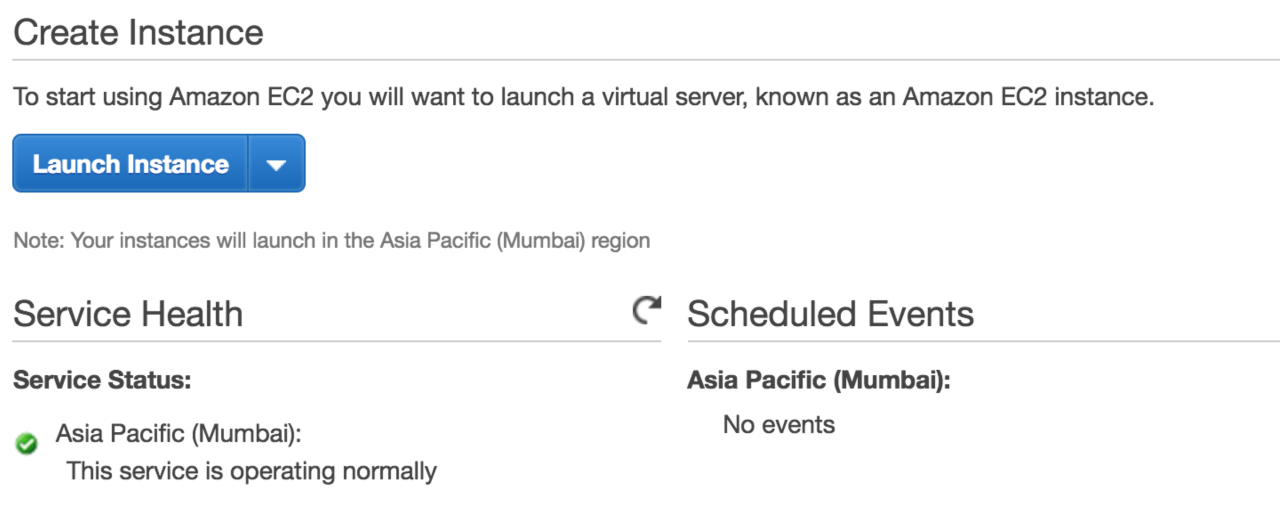
STEPS

1. Create a VM in AWS EC2 console

- Sign in to AWS console at https://aws.amazon.com/console/.
- Select EC2 after clicking on services tab on the top



- Click on launch instance



Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Note: Your instances will launch in the Asia Pacific (Mumbai) region

Service Health

Service Status:

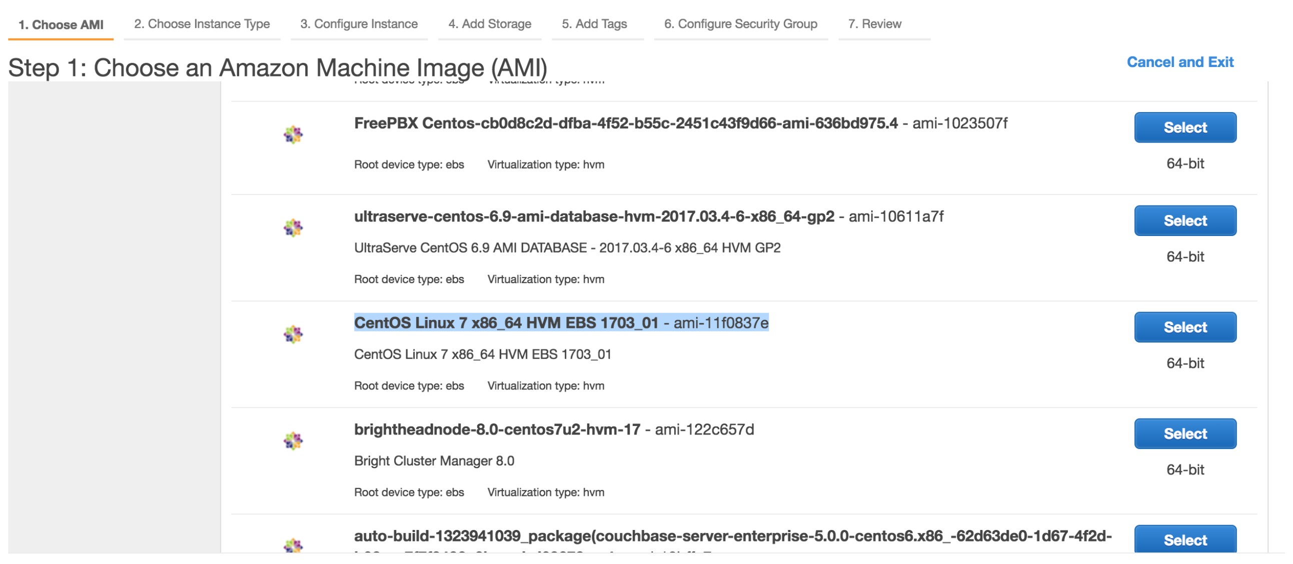
Asia Pacific (Mumbai):
This service is operating normally

Scheduled Events

Asia Pacific (Mumbai):

No events

- Choose an AMI after selecting Community AMIs tab and selecting centos from the operating systems list populated on the left side tab.
- Select the CentOS Linux 7 x86_64 AMI



Step 1: Choose an Amazon Machine Image (AMI)

FreePBX Centos-cb0d8c2d-dfba-4f52-b55c-2451c43f9d66-ami-636bd975.4 - ami-1023507f
Root device type: ebs Virtualization type: hvm
64-bit

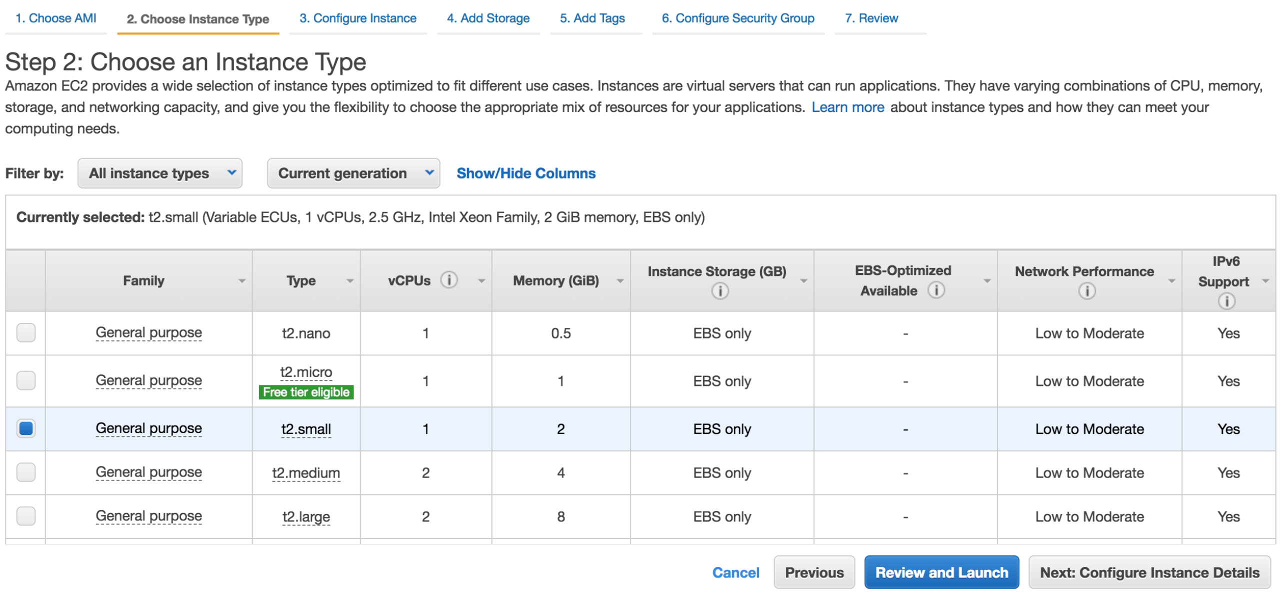
ultraserve-centos-6.9-ami-database-hvm-2017.03.4-6-x86_64-gp2 - ami-10611a7f
UltraServe CentOS 6.9 AMI DATABASE - 2017.03.4-6 x86_64 HVM GP2
Root device type: ebs Virtualization type: hvm
64-bit

CentOS Linux 7 x86_64 HVM EBS 1703_01 - ami-11f0837e
CentOS Linux 7 x86_64 HVM EBS 1703_01
Root device type: ebs Virtualization type: hvm
64-bit

brightheadnode-8.0-centos7u2-hvm-17 - ami-122c657d
Bright Cluster Manager 8.0
Root device type: ebs Virtualization type: hvm
64-bit

auto-build-1323941039_package(couchbase-server-enterprise-5.0.0-centos6.x86_-62d63de0-1d67-4f2d-

- Choose a t2.small instance type



Step 2: Choose an Instance Type

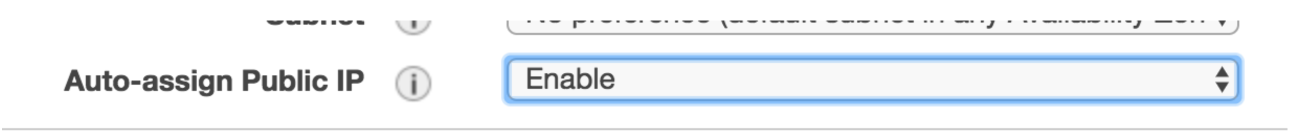
Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources 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.small (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 2 GiB memory, EBS only)

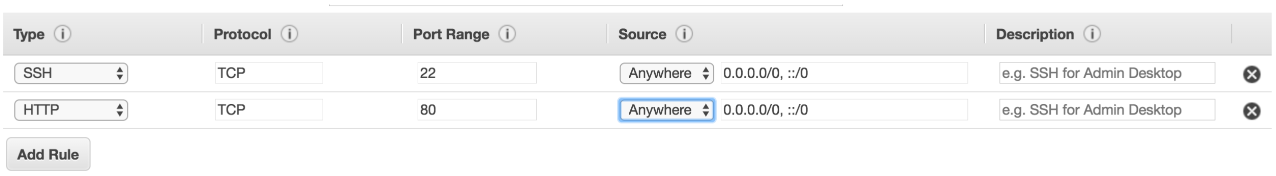
	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
✓	General purpose	t2.small	1	2	EBS only	-	Low to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

- Click on next: Configure instance details button available in the bottom
- On the next screen, select “enable” option for “Auto assign public IP” shown below



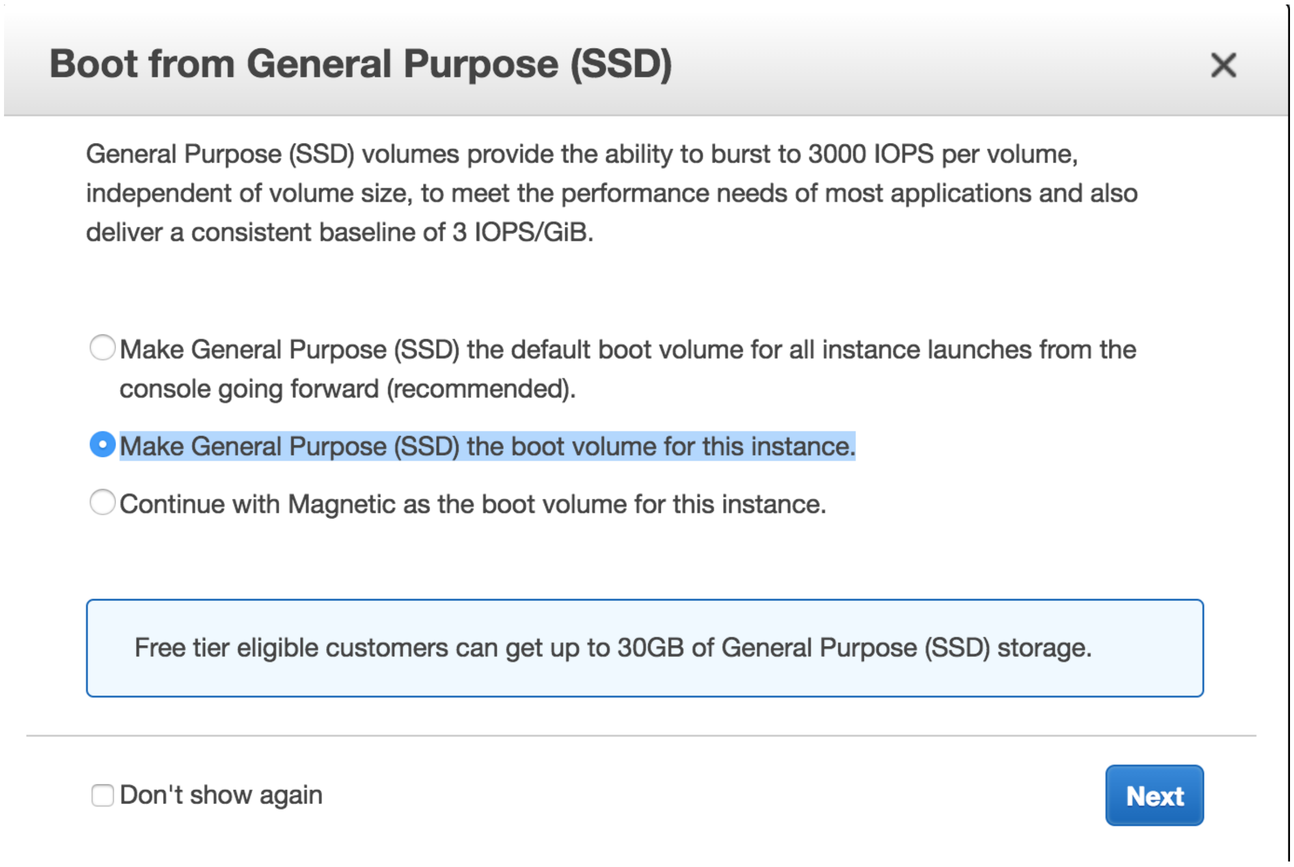
Auto-assign Public IP: Enable

- Now directly skip to configure security groups tab and a security rule for allowing http traffic, for now make the source as anywhere (You could whitelist a specific static/public there, if you have one)



Type	Protocol	Port Range	Source		Description
SSH	TCP	22	Anywhere	0.0.0.0/0, ::/0	e.g. SSH for Admin Desktop
HTTP	TCP	80	Anywhere	0.0.0.0/0, ::/0	e.g. SSH for Admin Desktop

- Click on review and launch



Boot from General Purpose (SSD)

General Purpose (SSD) volumes provide the ability to burst to 3000 IOPS per volume, independent of volume size, to meet the performance needs of most applications and also deliver a consistent baseline of 3 IOPS/GiB.

○ Make General Purpose (SSD) the default boot volume for all instance launches from the console going forward (recommended).

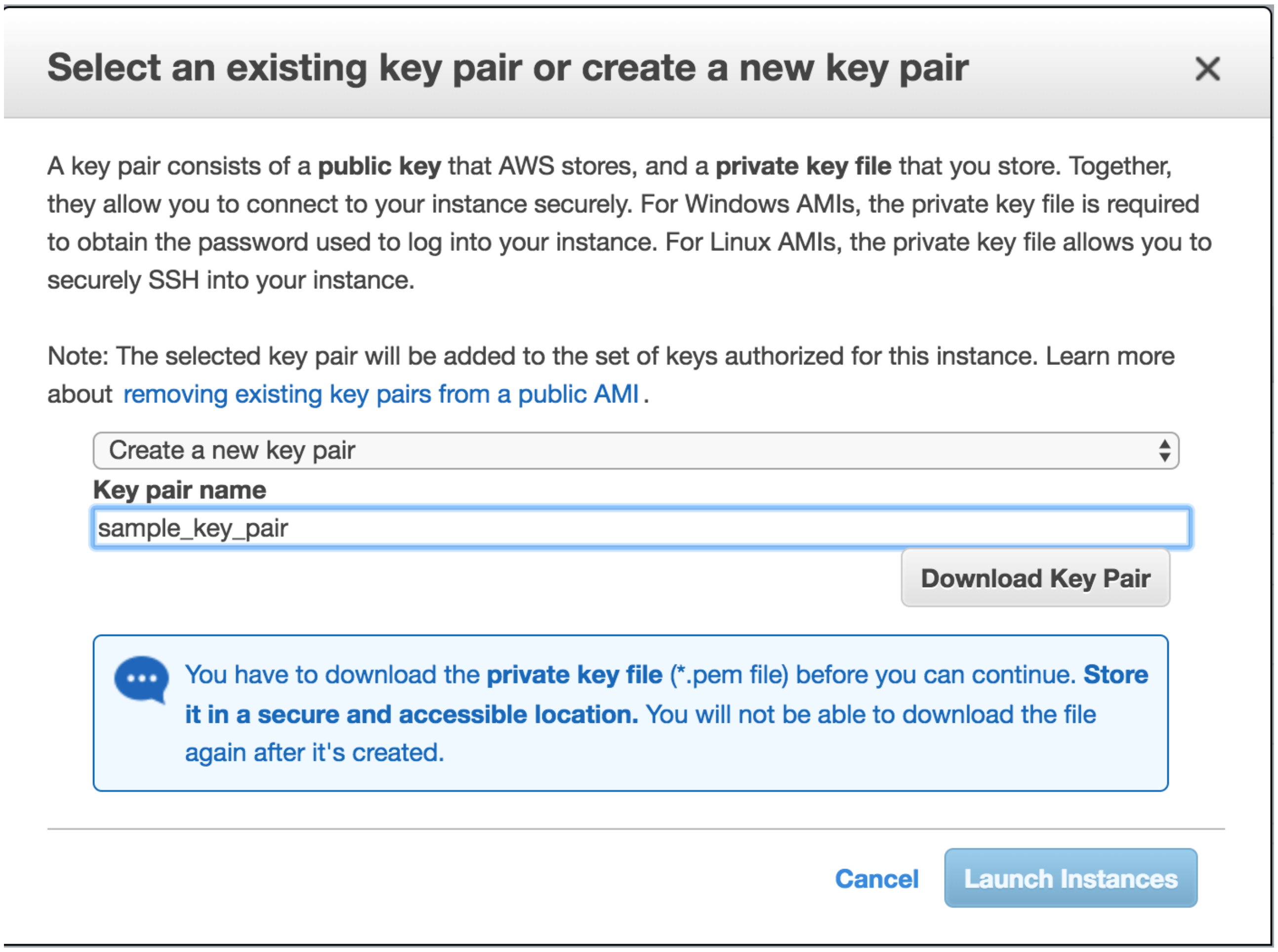
● Make General Purpose (SSD) the boot volume for this instance.

○ Continue with Magnetic as the boot volume for this instance.

Free tier eligible customers can get up to 30GB of General Purpose (SSD) storage.

☐ Don't show again

- Click on launch and you see the screen to create a key pair and give a name to it and click on download for the key pair. You can find the key pair in your download location of your browser.



Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI .

Create a new key pair

Key pair name

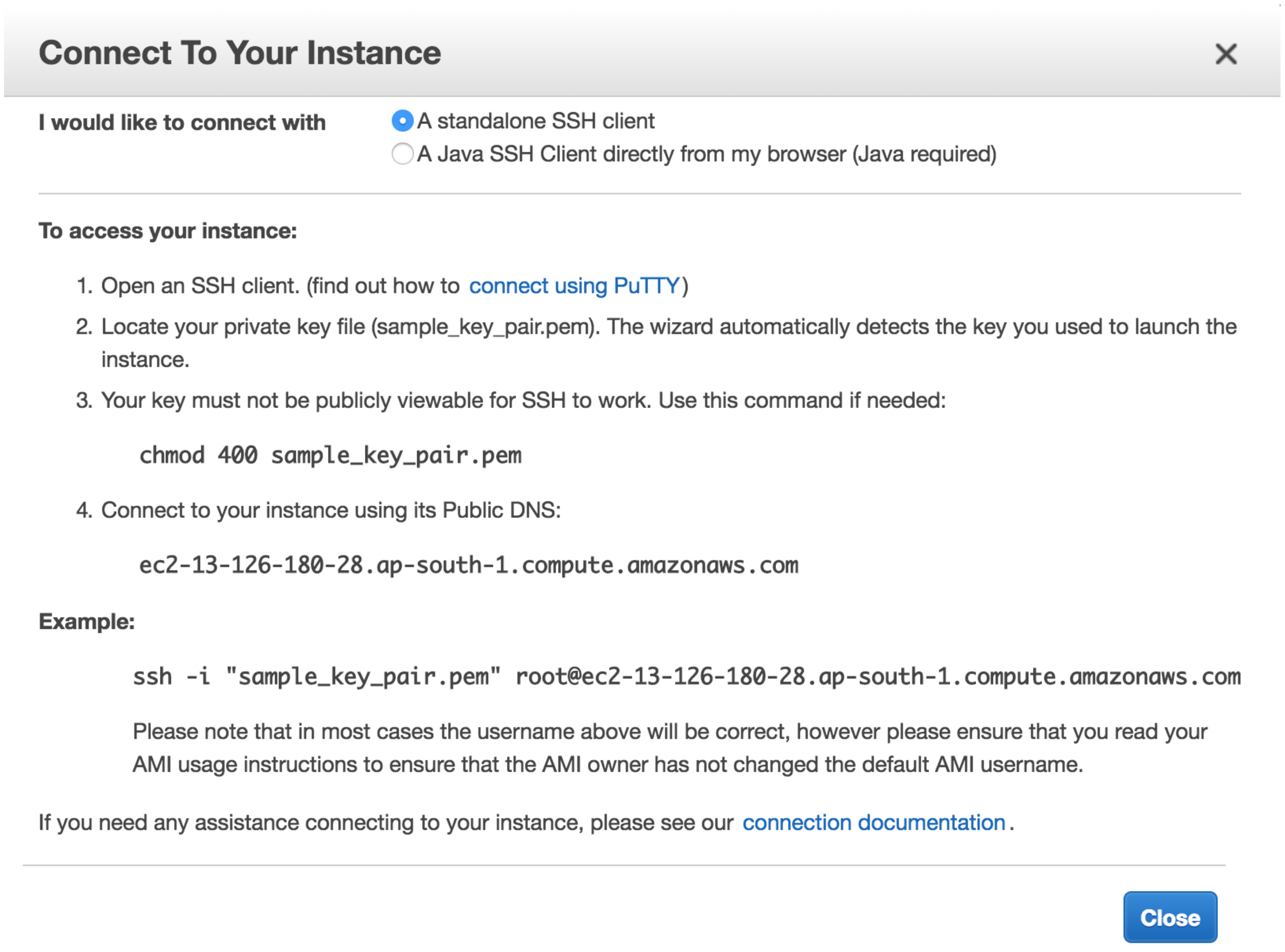
sample_key_pair

You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

- Launch instances and you will be back to ec2 dashboard page where you can find your VM. It will take time to actually be ready, wait until the instance state column says “running”.

Connecting to the VM

- Once the instance is running, select the instance and click on “connect” button above.


Connect To Your Instance

I would like to connect with [✓ A standalone SSH client] A Java SSH Client directly from my browser (Java required)

To access your instance:

1. Open an SSH client. (find out how to connect using PuTTY)
2. Locate your private key file (sample_key_pair.pem). The wizard automatically detects the key you used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command if needed:

```
chmod 400 sample_key_pair.pem
```

4. Connect to your instance using its Public DNS:

```
ec2-13-126-180-28.ap-south-1.compute.amazonaws.com
```

Example:

```
ssh -i "sample_key_pair.pem" root@ec2-13-126-180-28.ap-south-1.compute.amazonaws.com
```

Please note that in most cases the username above will be correct, however please ensure that you read your AMI usage instructions to ensure that the AMI owner has not changed the default AMI username.

If you need any assistance connecting to your instance, please see our connection documentation.

- In the above screenshot, step3 talks about changing the permissions of the key pair previously downloaded and the command mentioned in example section can be used to connect to your VM instance.
- In the download folder where your previously downloaded `sample_key_pair.pem` is located, run the following command from terminal to reduce the permissions.

`chmod 400 sample_key_pair.pem`

- At the same location, run the command to connect to VM instance through SSH.
  `ssh -i "sample_key_pair.pem" root@ec2-13-126-180-28.ap-south-1.compute.amazonaws.com`
  If this command doesn’t work for you, try this:

`ssh -i "sample_key_pair.pem" centos@ec2-13-126-180-28.ap-south-1.compute.amazonaws.com`

- From the above SSH command, you can figure out public IP of the instance as 13.126.180.28, make a note of it.

2. Installing essential packages

- After login, change to root to install some packages typing `sudo su && cd`
- Install packages `yum -y install git docker vim`
- Start the docker daemon with `service docker start`
- Clone the repo `git clone https://github.com/krishnakumar4a4/rust-irc.git`
- A folder rust-irc will be created locally which has both client and server.

3. Running the irc server on the machine

`cd rust-irc/rust-irc-server/`

```
docker build -t rust-irc-server:1.0 .
docker run -it --name rust-irc-server -p 80:80 -d rust-irc-server:1.0
```

- Check the container by running `docker ps`
- Check the logs using `docker logs -f rust-irc-server`
  Note: You will see huge log printing the terms downloading and compiling, please wait all of them finishes and give the message:
  `🚀 Rocket has launched from http://0.0.0.0:80`
- Now the server is ready.
- Make a note of the server public IP address mentioned in previous step 1

Repeat the steps 1 & 2 to create one more VM for the Client

4. Running the irc client on the machine

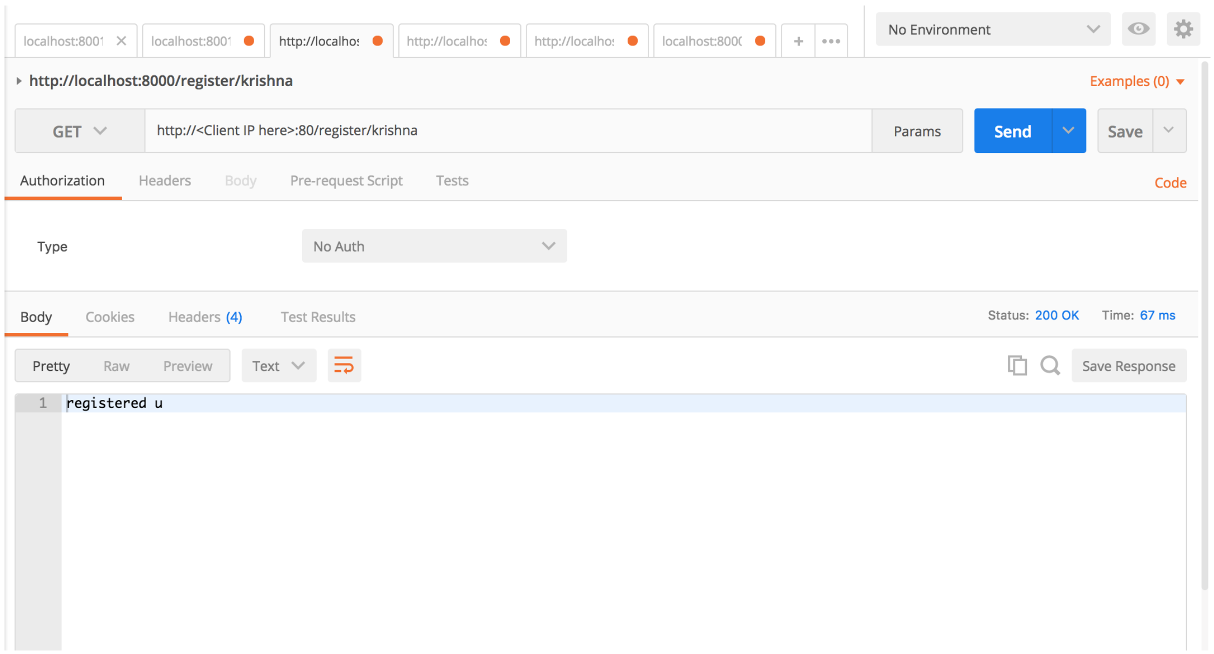
`cd rust-irc/rust-irc/`

- Edit the conf.ini file with server public IP address and client public IP address as you have identified at the end of step 1.
  `docker build -t rust-irc:1.0 .`

```
docker run -it --name rust-irc -p 80:80 -d rust-irc:1.0
```

- Check the container by running `docker ps`
- Check the logs using `docker logs -f rust-irc`
  Note: You will see huge log printing the terms downloading and compiling, please wait all of them finishes and give the message:
  `🚀 Rocket has launched from http://0.0.0.0:80`
- Now the server is ready.
- Make a note of the client public IP address mentioned in previous step 1. This IP address should be used for postman.

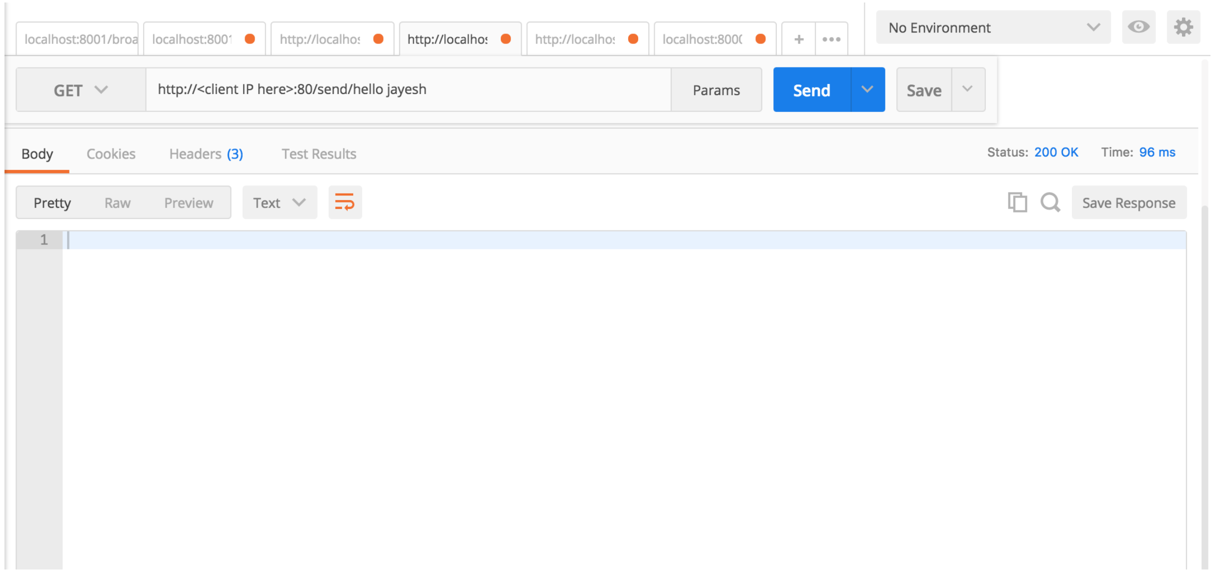
5. Checking with postman client

- Register the client with a name and you should see "registered u" as response indicating successful registration.
- 

  GET http://<Client IP here>:80/register/krishna

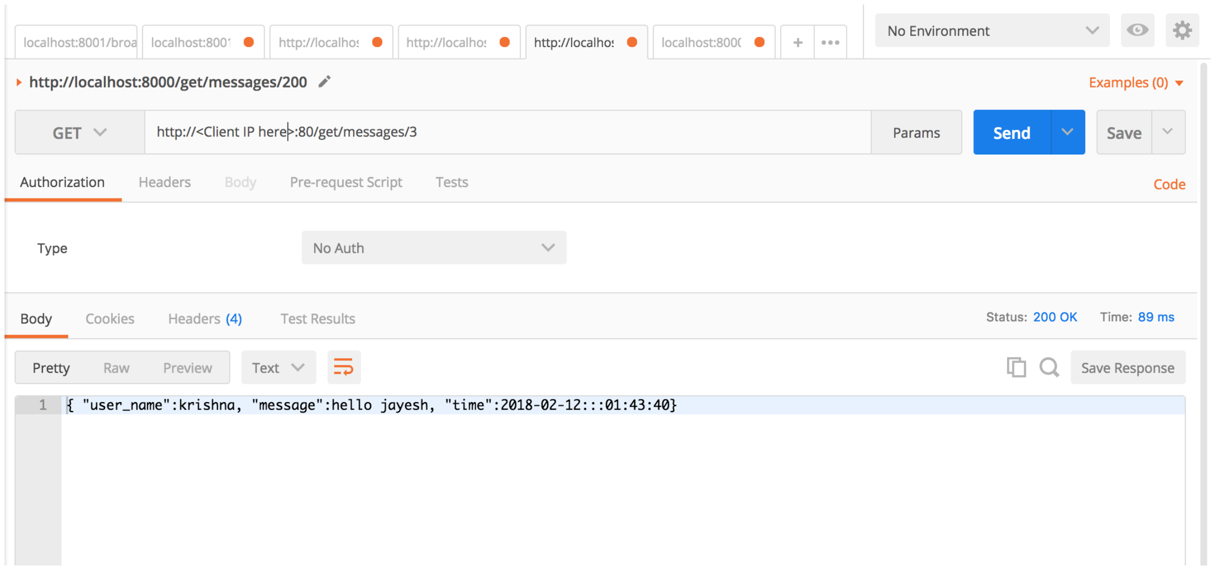
  Authorization — Type: No Auth

  Status: 200 OK Time: 67 ms

  ```
  registered u
  ```
- Send message, it is asynchronous and hence you see no response if successful
- 

  GET http://<client IP here>:80/send/hello jayesh

  Status: 200 OK Time: 96 ms

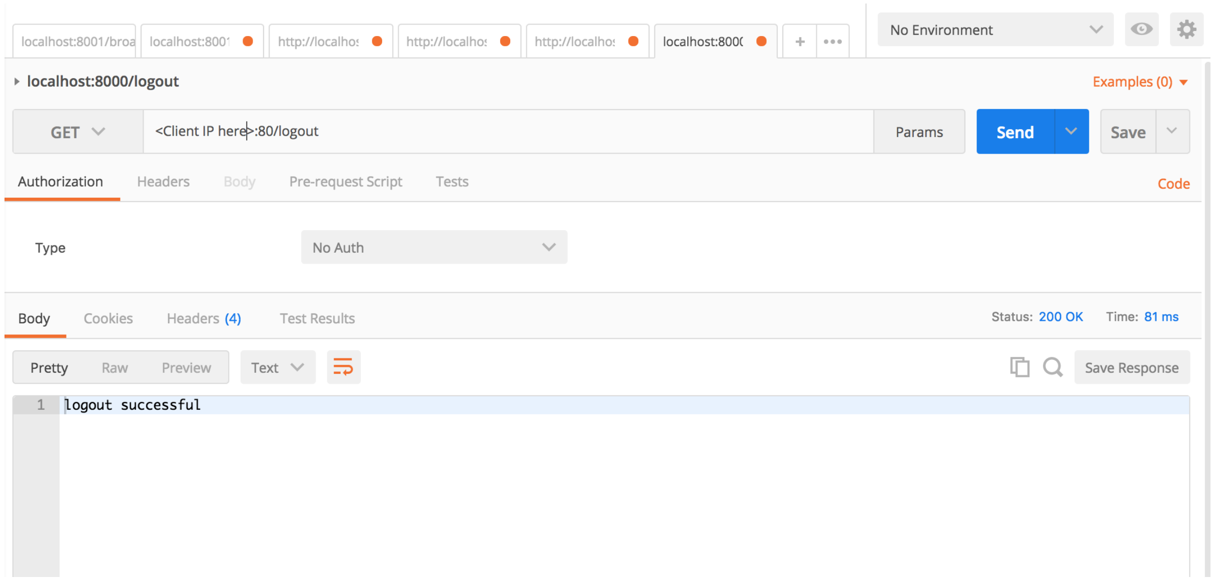
- Client has to poll for messages from the server, Here is how you get the last message
- 

GET http://<Client IP here>:80/get/messages/3

Type: No Auth

Status: 200 OK Time: 89 ms

```
{ "user_name":krishna, "message":hello jayesh, "time":2018-02-12:::01:43:40}
```

- Since you are the only client connected as of now, you see your own message, If you publish your server public IP address to your friends and ask them to use this in their client configuration, you will see their messages too.
- Finally, once you register with a name, you will not be able to register with the same name, until you logout.
- 

GET <Client IP here>:80/logout

Type: No Auth

Status: 200 OK Time: 81 ms

```
logout successful
```

Next steps

- Building a nice web UI replacing postman client.
- Use websockets for client to server connection for real time status, connection and session management.
- Use cache for session management on the server side instead of hashmap.
- Enhance the existing client and server APIs, some of them can be rewritten to POST methods, instead of GET.
- Build more APIs as required for the UI.

Code

- Available on github at https://github.com/krishnakumar4a4/rust-irc
- Star it if you like it. Open to contributions all the time.
- Reach me out at @KrishnaKumarT36 on twitter or on linkedIn .

Disclaimer

Project doesn’t implement the real IRC protocol, it simply uses REST to emulate this behavior.