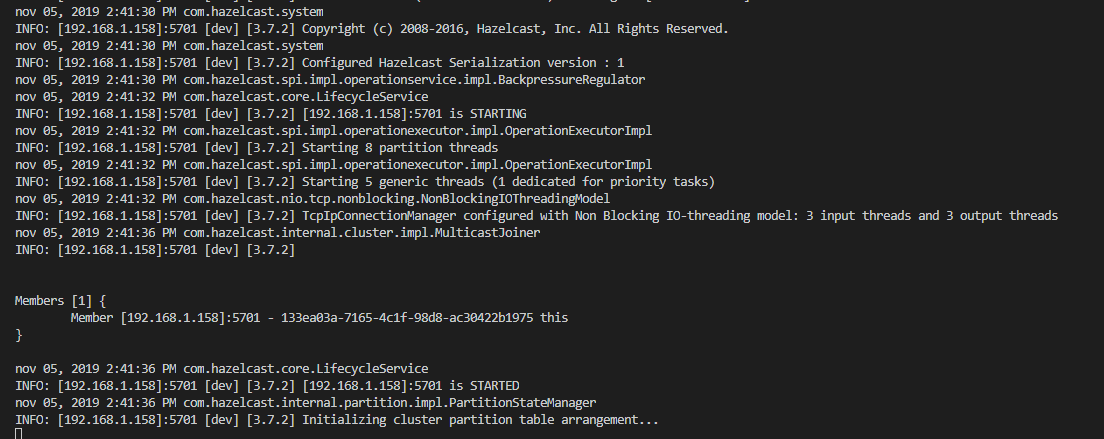
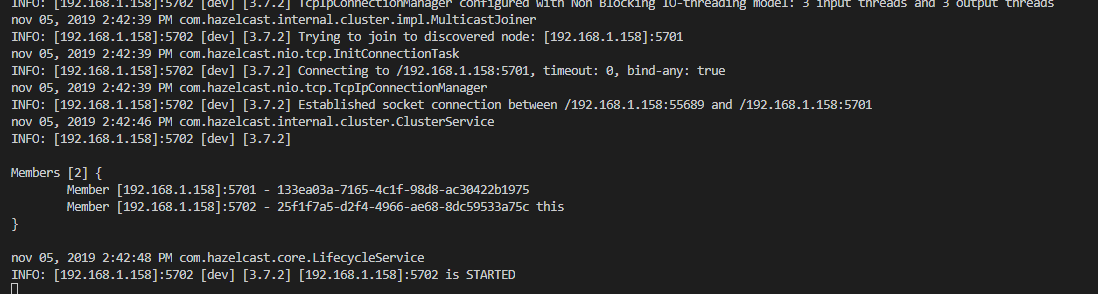
# Practica HazelCast

## Parte 1

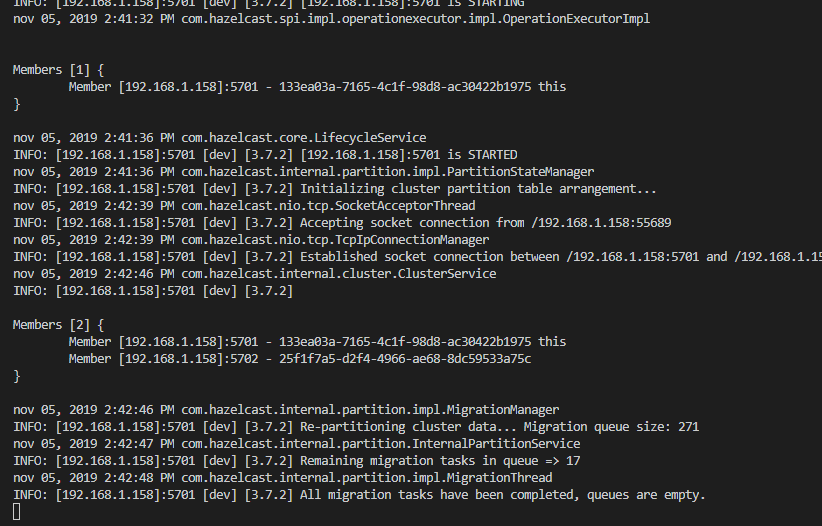
Se levanta el primer nodo



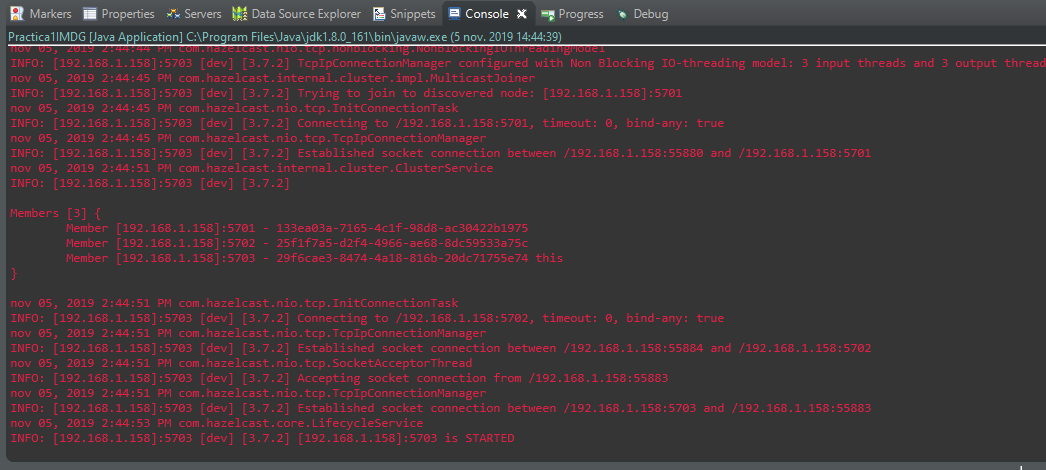
Se levanta el segundo nodo



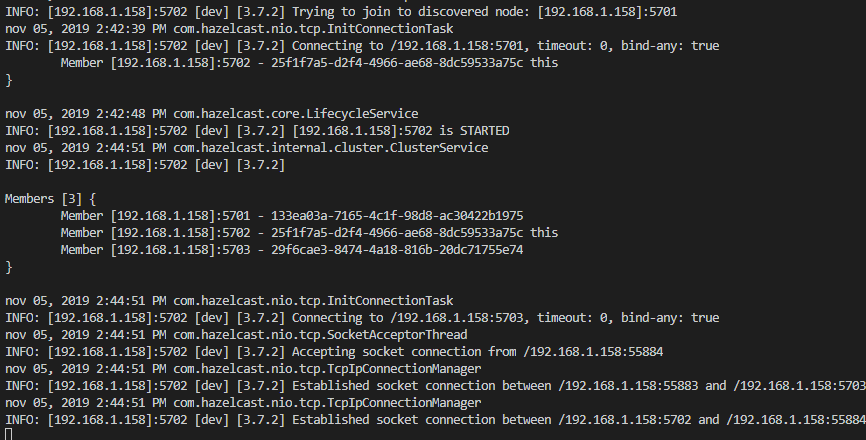
Desde la terminal donde se visualiza la ejecución del primer nodo, se puede ver que el segundo se conecta



Se levanta el tercer nodo



Desde la terminal donde se visualiza la ejecución del segundo nodo, se puede ver que el tercero se conecta



## Parte 2: Uso básico de cache distribuida

En el siguiente código se expresa la implementación de los 3 nodos los cuales usan contadores distribuidos de hazelCast (countdownlatch)

public static void main(String[] args) throws InterruptedException {

        // Instanciar hazelcast y crear una cache

        // Insertar un dato y arrancar 3 veces el main,

        // Leer el output de consola y ver como hazelcast va encontrando "miembros"

        // Comprobar que se conectan (en el output deberian verse 3 miembros en la consola) y capturarlo

        Config config = new Config();

        config.getNetworkConfig().getJoin().getTcpIpConfig().addMember("localhost").setEnabled(true);

        config.getNetworkConfig().getJoin().getMulticastConfig().setEnabled(false);

        Person p = new Person("Carlos", 28053, "", "");

        DataGridNode node = new DataGridNode();

        ICountDownLatch latch = node.getHzInstance().getCountDownLatch( "countDownLatch" );

        // tokens por numero de nodos

        latch.trySetCount(3);

        System.out.println( "Waiting" );

        //Process to be executed----------------------

        node.addToCache(p);

        //--------------------------------------------

        //-----Restar un token------------------------

        latch.countDown();

        System.out.println("Faltan " + latch.getCount() + " nodos");

        //--------------------------------------------

        latch.await( 100, TimeUnit.SECONDS );

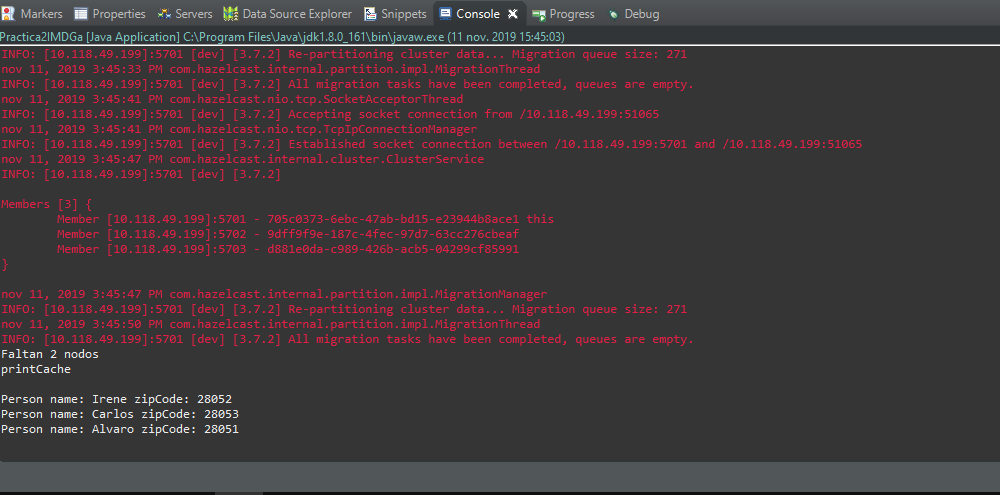
        node.printCache();

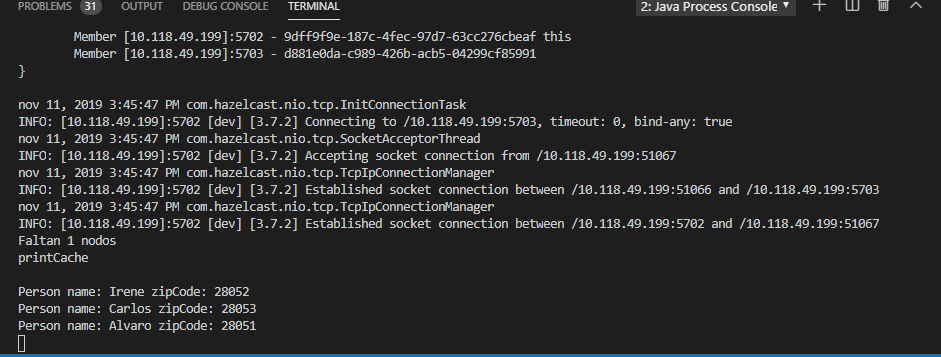
        //-------------------------------------------

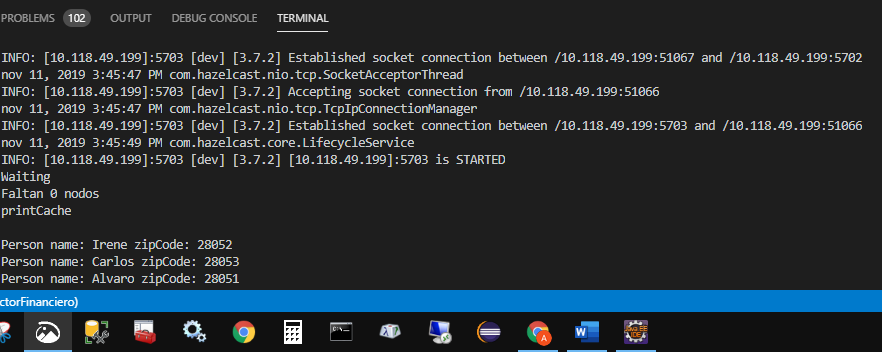
        latch.destroy();

    }

Como resultado de la ejecución los nodos instanciados esperan la terminación del proceso de todos los nodos para hacer un printCache:







# Parte 3: Utilizando nuestro propio servidor

Para esta parte se levanta un servidor genérico el cual se puede conectar cualquier cliente con la ip correspondiente (localhost en este caso).

El siguiente es la implementación de un cliente el cual hace print de cache inicialmente y luego agrega un objeto al cache, el cual queda persistido en el servidor

public static void main(String[] args) {

        // Instanciar hazelcast Cliente y crear una cache

        ClientConfig config = new ClientConfig();

        ArrayList<String> ips=new ArrayList();

        ips.add("127.0.0.1");

        config.getNetworkConfig().setAddresses(ips);

        HazelcastInstance client = HazelcastClient.newHazelcastClient( config );

        //print cache

        printClientCache(client);

        //Vuestro código va aqui

        IMap<Long, Object> cacheNode = client.getMap("data");

        Person p = new Person("Alvaro", 28005, "", "");

        //put to cache

        IdGenerator idGenerator = client.getIdGenerator("newid");

        cacheNode.put(idGenerator.newId(), p);

        //print cache

        printClientCache(client);

        client.shutdown();

    }

    private static void printClientCache(HazelcastInstance client){

        System.out.println( "printCache\n" );

        IMap<Long, Person> map = client.getMap("data");

        for (Entry<Long, Person> entry : map.entrySet()) {

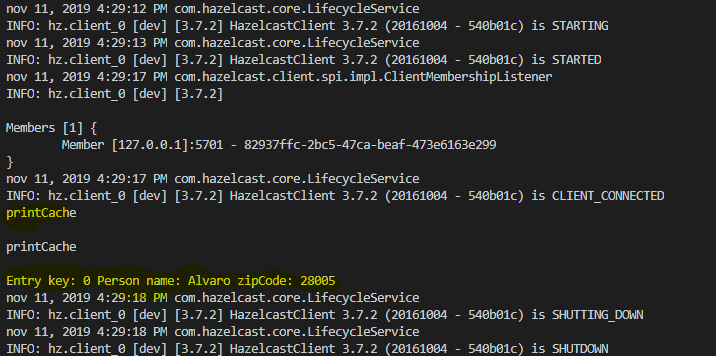
            System.out.println("Entry key: "+ entry.getKey() + " Person name: " + entry.getValue().getName() +

            " zipCode: " + entry.getValue().getZipCode());

        }

    }

En la primera ejecución el cache esta vacío



Posterior se re ejecuta el cliente y se visualiza que la cache esta previamente poblada:

