Table of Contents

[**Purpose of this document:** 2](#_Toc14617077)

[**Problem Statement Question 3:** 2](#_Toc14617078)

[**Expected Result:** 2](#_Toc14617079)

[**Overview of the System:** 2](#_Toc14617080)

[**Features:** 2](#_Toc14617081)

[**Assumptions:** 2](#_Toc14617082)

[**Improvements:** 2](#_Toc14617083)

[**Tools Used:** 2](#_Toc14617084)

[**Architecture and Flow:** 3](#_Toc14617085)

[**Steps followed for Building Application:** 4](#_Toc14617086)

[**Source Code:** 8](#_Toc14617087)

[**Steps to run the application:** 8](#_Toc14617088)

[**Testing the application:** 9](#_Toc14617089)

## **Purpose of this document:**

This document contains the high level design details of IP blocker. It includes the architecture of the system and use cases.

## **Problem Statement Question 1:**

Assume a blacklist IP monitoring system, where any user trying to access the system, is checked against a blacklist of IPs, and the connection is dropped if the IP was found to be blacklisted.

Given a list of IPs (independent of size, can be anywhere from 1 record, to 50,000,000 records and more), design a database / cache structure to store those IPs for fast lookup, based on an incoming users IP.

## **Expected Result:**

Data model, format of the keys/values, and description of the reasoning of your choices.

## **Overview of the System:**

IP Blocker system maintains the Block Listed IP address in the **Redis database**. When the user is asking for the access with the ip address as parameter. It checks against **Redis** database and provide the message.

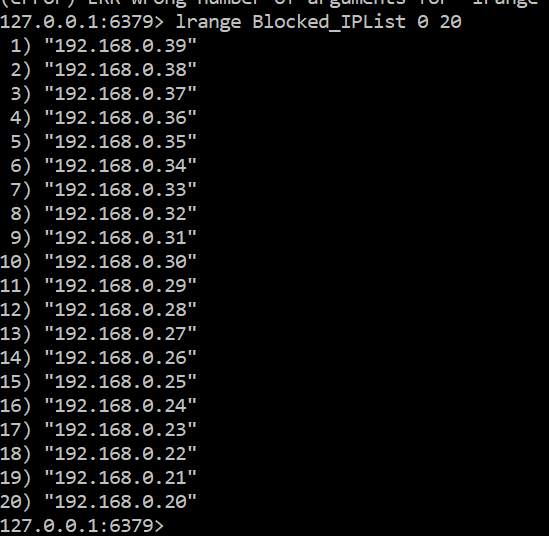
## **Features:**

This is a simple IP Blocker system. Which has only one end point which will be responsible for checking the ip address with ip in Redis databse and provide the information to the client system.

1. <http://localhost:9000?ip>=192.168.0.1 - GET

## **Assumptions:**

1. Blocked Listed Ips already present in the **Redis Database** with Key as **Blocked\_IPList**



## **Improvements:**

1. We can use the cache expiration features of play application to get the out of box features.

## **Tools Used:**

1. Java 1.8
2. Scala 2.12.8
3. IntelliJ IDEA with Scala plugin
4. **Redis 3.2**
5. Play Framework 2.7.3 (MVC framework)
6. SBT build tool.

## **Architecture and Flow:**

Ip blocker system is a simple application where we check the ip whether it is blocked for access or not. There are two parts in this application

1. Maintain the Blocked IP List
2. Simple Service to validate the given IP with Blocked IP List (It connects to Redis and check)

**Maintain the Blocked IP List:**

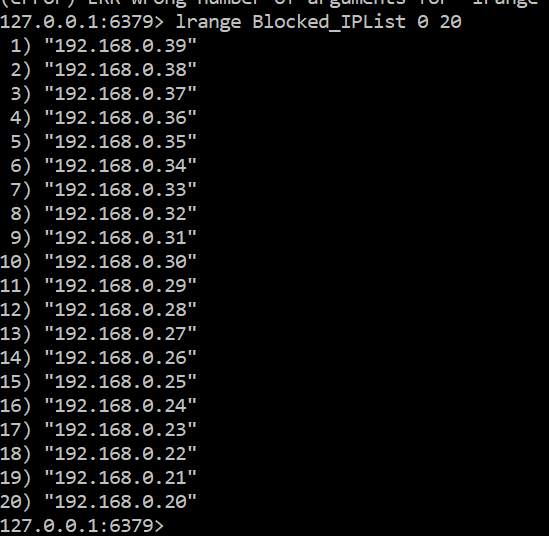
To store blocked list of IP address, I have chosen **Redis In-memory database**, Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Using this external in-memory Database we can access them from IP address list as the list will be in cache, so the accessing of the data will speed.

This main purpose of using Redis is, it will be external in-Memory Database, and the values will be in cache. We can access them very quickly and it will not be in internal server.

Assumption here is I have already stored the Blocked IP List in the Redis database.

To store Keys, Used Redis command line client to store the IP, following is the command to create and store IP

redis> LPUSH **IPList** "192.168.0.20"



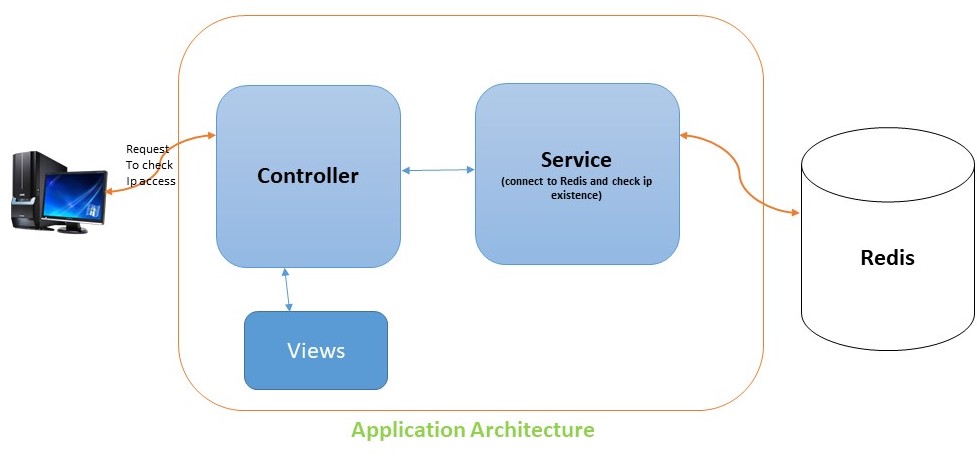
**Simple Service to validate the given IP with Blocked IP List:**

I have chosen **Play Framework 2.7.3** for MVC web framework to build a simple service which has only one end point to check the given IP address is blocked or not. Play is a high-productivity Java and Scala web application framework that integrates the components and APIs you need for modern web application development.

Play framework is integrating with **Redis seamlessly** with the help of **play-redis** module and provide the customized api for synchronous and asynchronous requests.

[Play framework 2](http://playframework.com/) is delivered with [SyncCacheApi and AsyncCacheApi](https://playframework.com/documentation/2.7.x/ScalaCache). This module provides **implementation of a cache over Redis** server, i.e., key/value storage.

Besides the compatibility with all Play's cache APIs, it introduces more evolved API providing lots of handful operations. Besides the basic methods such as **get, set and remove**, it provides more convenient methods such as **expire, exists, invalidate** and much more.

****

Ip blocker system contains below layers as show in the above architecture diagram.

1. Controller (Web layer)
2. Service (For complete business logic)

Play supports the Dependency Injection, and injected all the dependent objects using @Inject annotation. This avoids the creation of objects on our own using new operator instead server will create all the objects and injected.

Developed one end to validate the ip for access.

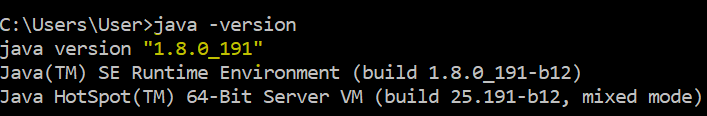
1. [http://localhost:9000/](http://localhost:9000/processCSV)ip=192.168.0.1 - GET - Responsible for validating the IP accessibility.

## **Steps followed for Building Application:**

**Step 1:**

Checked/installed for all the prerequisite software’s available on system. Prerequisite software’s are.

1. Java 1.8
2. SBT Build tool
3. Redis 3.2

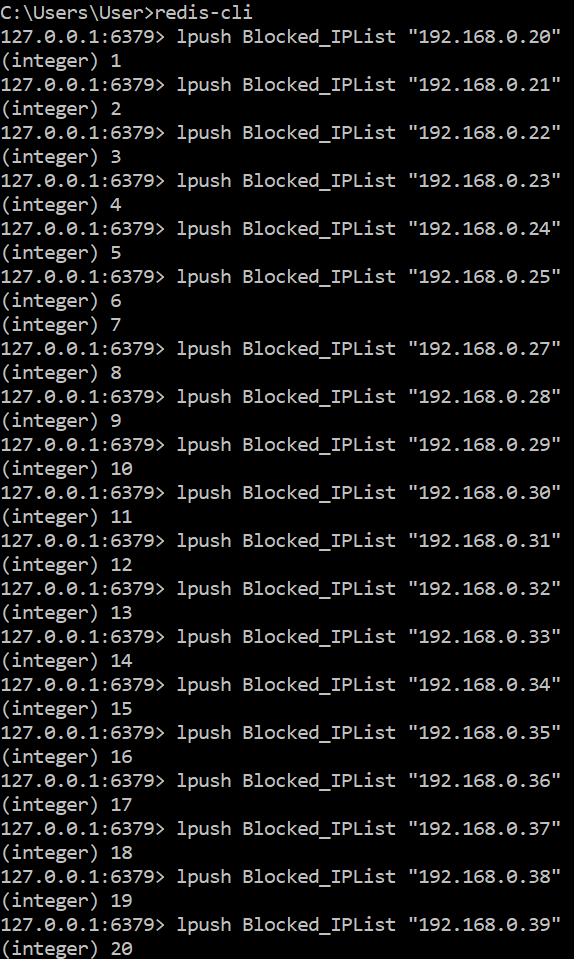


**Step 2:**

Installed Redis 3.2 on the sytem and connected to Redis by using command

redis-cli

And added all the Blocked Listed Ips using the below command



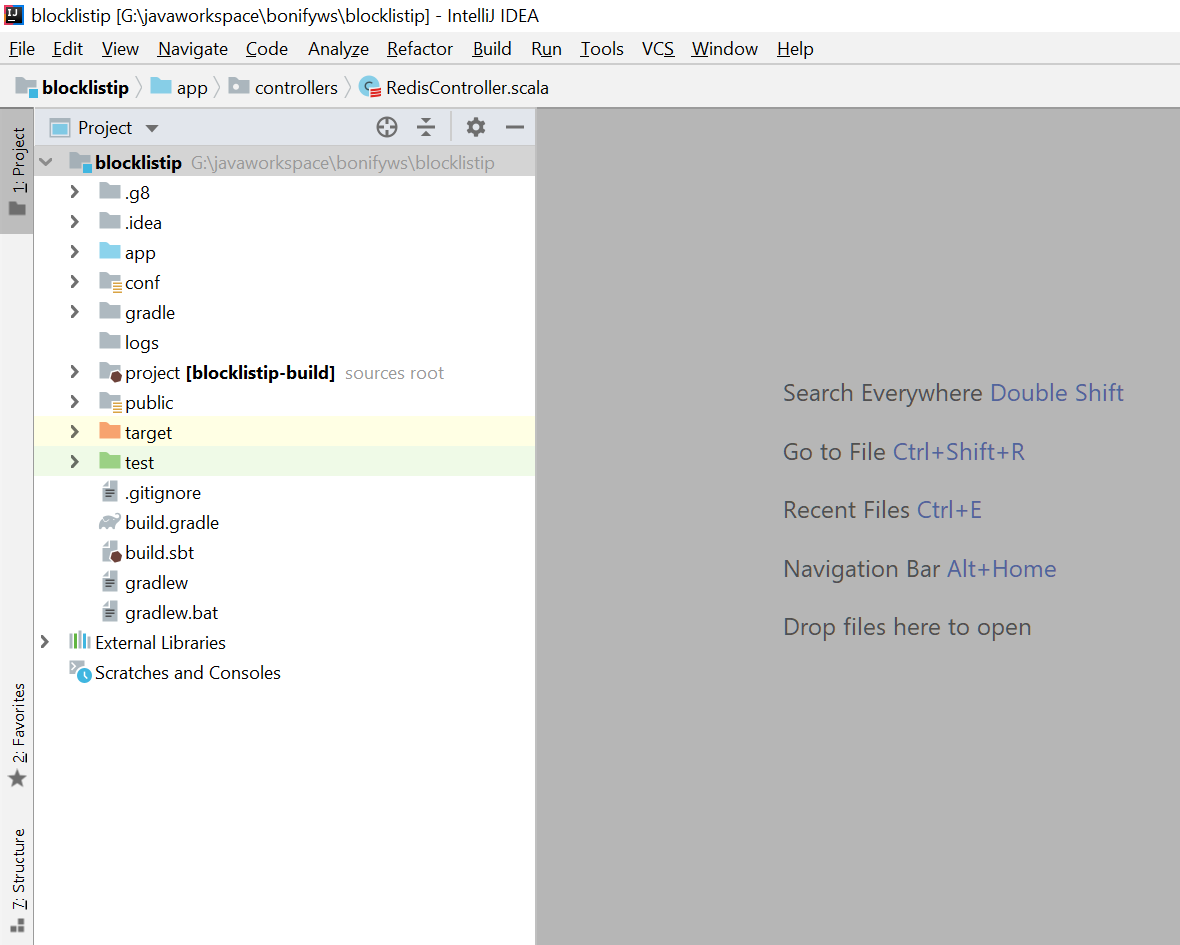
**Step 3:**

Create a play application using below command from console.

sbt new playframework/play-scala-seed.g8

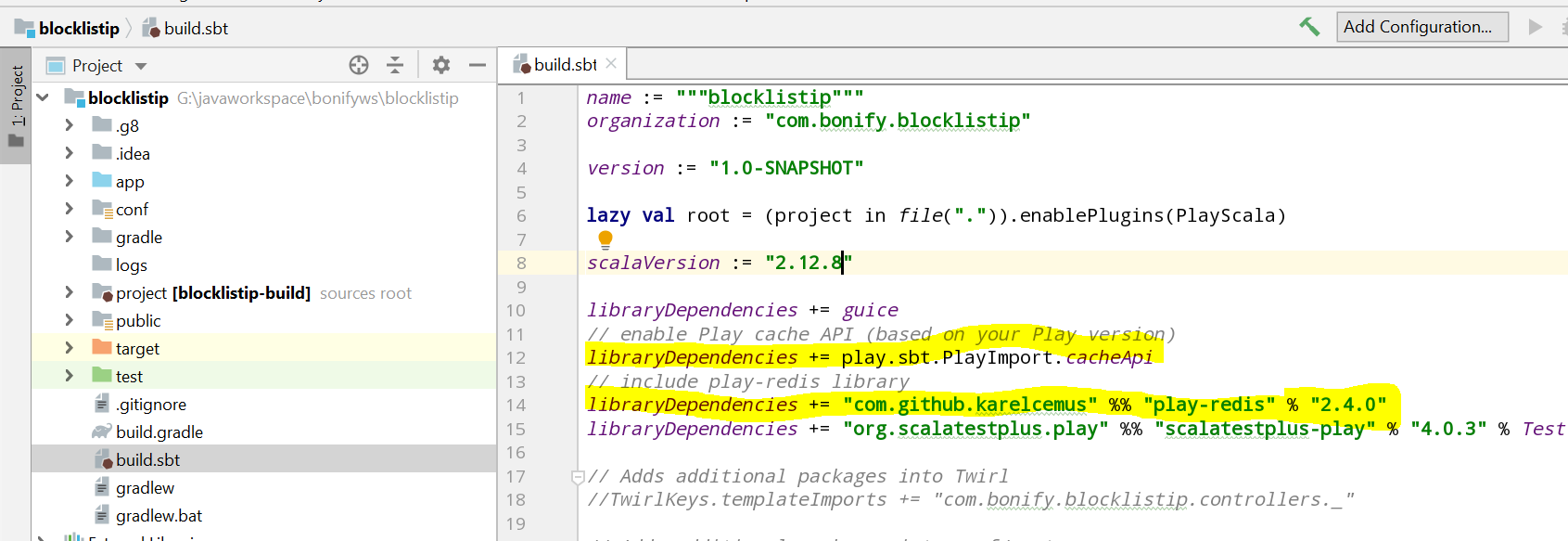
**Step 4:**

Import the created application into the IntelliJ IDE using import options while IntelliJ startup.



**Step 5:**

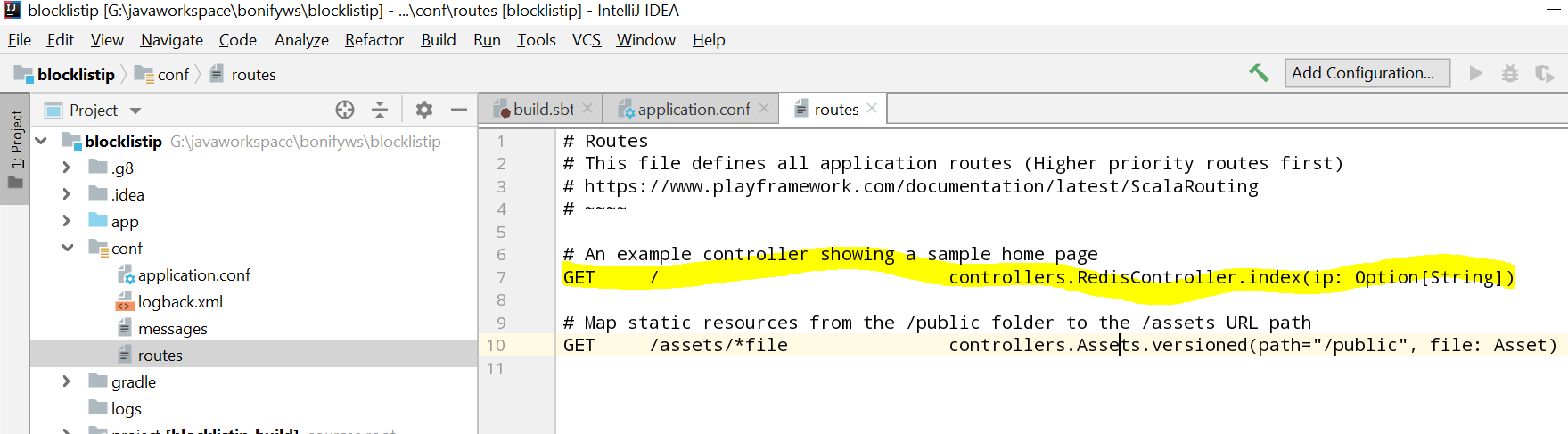
Open the build.sbt and add the redis module to the application.



Open the **application.conf** file and add the single line to configure play with redis. This configuration is enough to connect to Redis running on the localhost.

play.modules.enabled += play.api.cache.redis.RedisCacheModule

Open the **conf/routes** file to add the endpoint with one parameter.



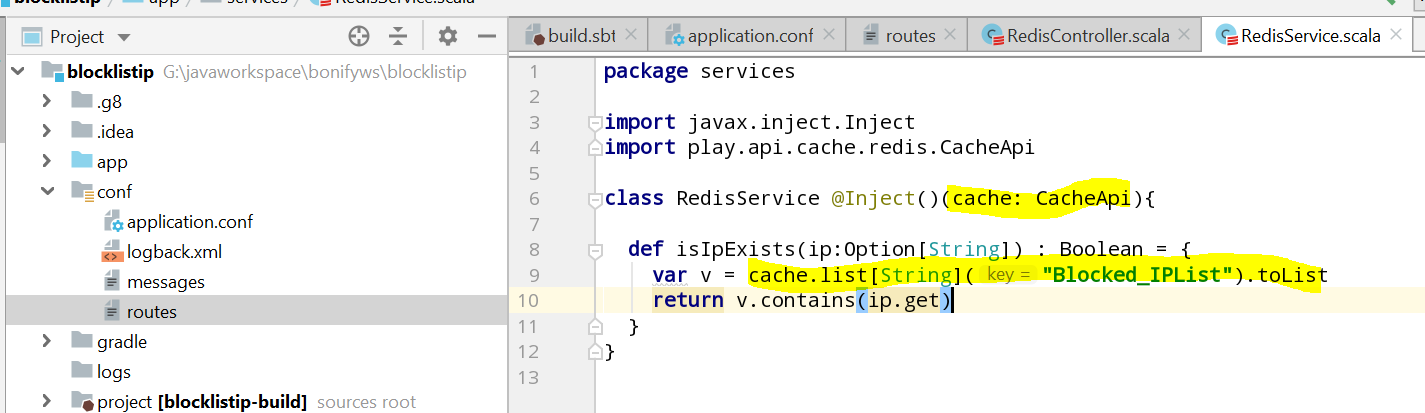
**Step 6:**

Created RedisController, RedisService.

**RedisController** – Receives the request from UI and delegates the request to service and result will be sent to the UI. RedisService is injected into the RedisController to do the business logic. Rediscontroller contains only one method which takes the ip address as parameter and validate it with IP list in Redis. It is in **app/controller** package.



**RedisService –** Receives the request from Controller and connect to the Redis and receives the IP list. And it validates provided IP with list returned from Redis. It is in **app/services** package.



**Step 7:**

Created the one view corresponding to end point to show the message of accessibility. It will be in **app/views** package.

1. index.scala.html – to show the message.

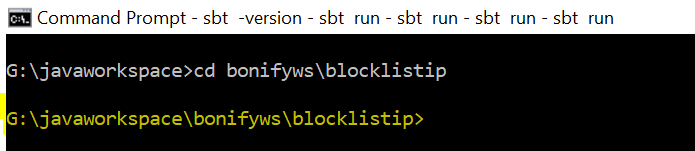
## **Source Code:**



## **Steps to run the application:**

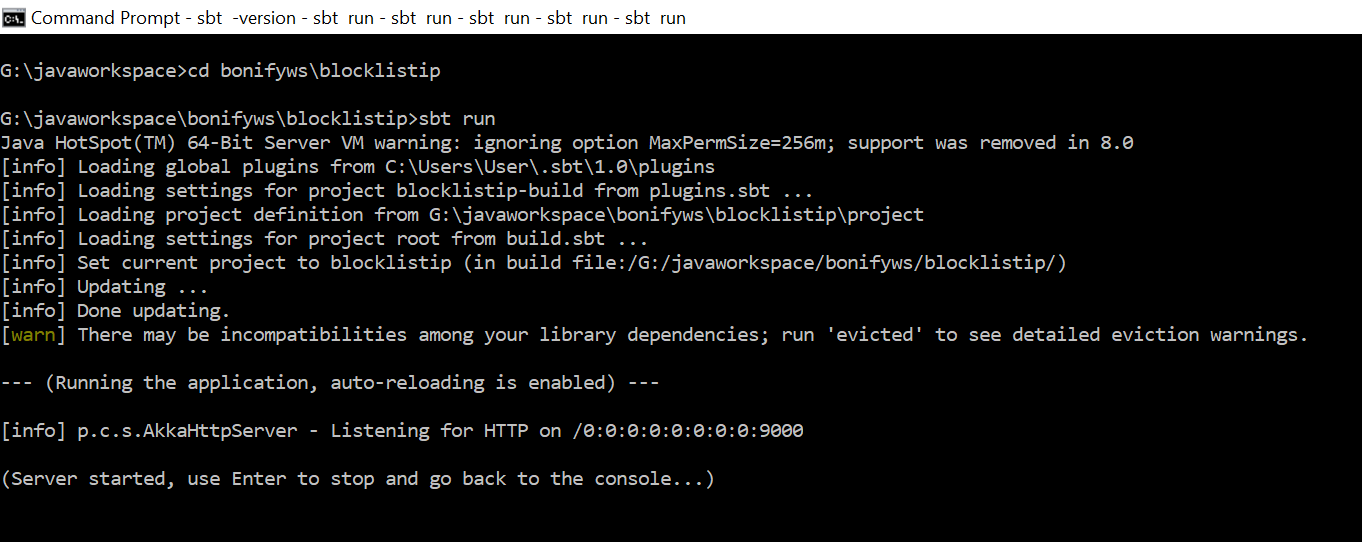
**Step 1:**

Make sure that all the prerequisites **(Java 1.8, SBT latest version, Redis 3.2 with all the Blocked Listed Ips)** are on system. Unzip the source code and move to that directory from command prompt using cd command.



**Step 2:**

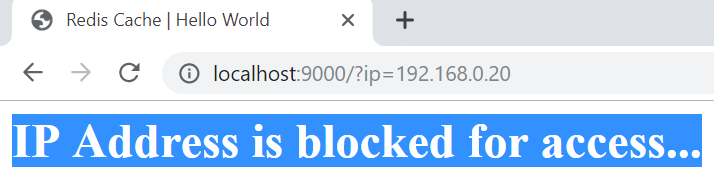
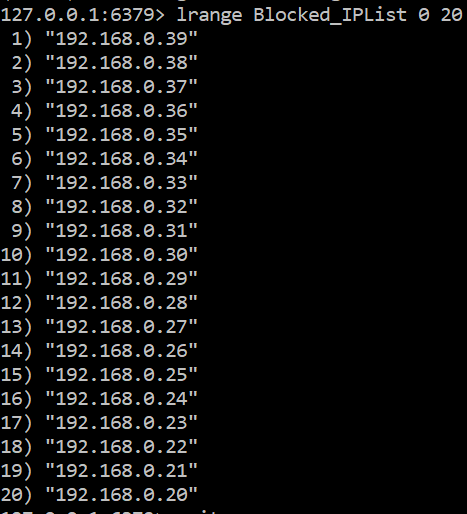
Run the **sbt run** command and wait for application to be started.



## **Testing the application:**

**Test for IP Access:**

Hit the <http://localhost:9000?ip=192.168.0.20> from browser. We should be able to see the message on the web page saying **“IP Address is blocked for access...”** as this Ip already in the Redis database in the Blocked list.

Hit the <http://localhost:9000?ip=192.168.0.210> from browser. We should be able to see the message on the web page saying **“IP Address is active for access...”** as this IP not in the Redis Blocked list.

