Design Decisions

Les Foster, Jun 3, 2020

# Background

The project is based on the “readme.md” file within the repository. Initial text repeated below.

|  |
| --- |
| IP Address Management REST API Create a simple IP Address Management REST API using Spring Framework on top of any data store. It will include the ability to add IP Addresses by CIDR block and then either acquire or release IP addresses individually. Each IP address will have a status associated with it that is either “available” or “acquired”.  The REST API must support four endpoint:   * Create IP addresses - take in a CIDR block (e.g. 10.0.0.1/24) and add all IP addresses within that block to the data store with status “available” * List IP addresses - return all IP addresses in the system with their current status * Acquire an IP - set the status of a certain IP to “acquired” * Release an IP - set the status of a certain IP to “available” |

# Restating

IP addresses are often distributed among VPNs in AWS (and very likely other cloud vendors of whom I've heard do similar things). At least from the AWS side, some management of these IP addresses is required for things like adding EC2 instances in response to auto scaling, or perhaps for adding multi-IP consumers such as some of the serverless APIs.

For whatever purpose, this task is to use “block” allocation (CIDR block to be precise) and individual “checkout” of IP addresses. Rather than calling them “locked” we call them “acquired”. Rather than unlocked, they are “available”.

## Questions

I asked these questions of the lead at Trillion:

 First, supporting both IPV6 and V4?  Second, are we limiting or distributing CIDR blocks?  I can foresee very large reports coming back.  Also, when it says "a certain IP" is acquired, does that imply an IP is passed in to acquire it, or that the caller just needs any available IP address?

I received these answers:

Supporting V4 is good.

Limit to one CIDR block. Needs to take in one CIDR block and store the corresponding IP range as list of IPs.

And,an IP is passed to acquire it and needs to be checked if it is available to acquire or not.

## Terminology

* To Create is really to let the system know it can manage a set of IP addresses.
* To List is to dump IP vs status
* To Acquire is to set an IP address into a state such that it cannot be acquired, and at the same time return it to the caller, with the understanding the caller can use the IP address.
* To Release is to inform the system that the IP address may be acquired, with the understanding (presumably) that the previous acquisition caller gives up rights to the IP address.

# REST API

CREATE – equivalent to POST, in that a new resources is being created. Here, a CIDR block will be posted, and the known IP address list should be increased. This action will change what can be acquired, and listed. But nothing within this range should have been acquired previously.

LIST – equivalent to GET, in that nothing new is created. Only a response with data is given. Should consider compressing the result.

ACQUIRE – equivalent to PUT, in that a resource is being changed. However, no caller should ever expect to push the entire state of the content. Only one such IP address. Acquire will transform the backend state (persistent state) by marking one IP address as “acquired”. This should return a value (the IP address).

RELEASE – equivalent to PUT, in that a resource is to be changed. Not changing entire set—only a single IP address. Should not return any value except success.

All of the above may be taken as Use Cases.

# Scenarios

Time permitting, these will be addressed. However, priority will be given to getting the “happy path” completed.

* Attempt is made to Create IP addresses in some range that overlaps in whole or in part, existing IP addresses.
  + Simplest response is to reject that with an appropriate error (TBD).
  + More complex would be to allow all IP addresses possible.
  + Otherwise could give reject response with list of subranges that might have been possible.
* Attempt is made to acquire and no IP addresses remain. [Priority 3]
  + Error : nothing left. REST response error TBD
* Attempt is made to add nonsense CIDR block range. [Priority 2]
  + Prevent this happening with proper structure of inputs.
  + Return error.
* Attempt is made to release an IP address that has not been acquired. [Priority 4]
  + Return a warning – caller should know what has been allocated.
  + This may be a sign of a problem with IP management itself.
  + Best to make certain the system cannot lose track of what has been acquired!
* DOS – Making many requests.

# NFRs – Non Functional Requirements (Assumptions)

This system should

* Have a reasonable level of security given time constraints. At least there must be credential secured in some way.
* Reliable (probably covered by choice of Spring framework and data source backing).
* Standardized (no need for homegrown databases, etc.)
* Be cloud-Friendly – true use of RESTful state. Should be configurable as possible given time constraints.
* Have good code design.
* Be time and space complexity efficient.
* “On top of any data store” - may imply that the system work at a high standard level (JPA perhaps?) and the persistent store may be swappable by the end user.

# CIDR Block Rules: A Review

CIDR stands for Classless Inter-Domain Routing. CIDR blocks are notated with a base address plus a prefix bit-length indicator. The bit-length is for the prefix, or the “routing mask”. Here are examples.

10.0.0.0/24 - 24 bits used up for routing mask. 8 bits left for ids. 256 possible addresses.

172.1.9.1/8 - 8 bits for routing mask. 24 bits for ids. 16M possible addresses.

92.168.100.14/16 – 16 bits for routing mask. 16 left for ids. 64K possible addresses.

Aside: IPv6 also supports CIDR blocks.

## Implications

We are supporting only IPv4. This limits maximum to four bytes worth of IP's. That's 2^32 or 4,294,967,296 possible IP addresses. What if that is divided by 8? 536,870,912 bytes (8 bits each).

Still quite large.

Limiting to /16. That's around 64K different things to keep track of.

# Storing All This

How to back this? 16Mb is not huge by modern standards, but in a database it could get cumbersome. However we only need a T|F value. That can be represented in a single bit. We therefore would need swaths of up to 16M-bits. That amounts to 2 million bytes. That's still a lot of simplistic rows.

It might also be possible to simply store only what's been acquired. That strategy pays off for sparse use, but not for heavy use. Then the opposite might be more practical. But read on...

# Operations

Before settling on how to persist these, it may be best to consider how they must be fetched or returned. We have operations for create. Barring the rest, that could be accomplished just keeping a list of strings.

Acquiring one? Releasing one? Both require that an address be found, and either marked off limits or returned to the pool.

List? Requires being able to show everything. Let's put that aside for the moment.

When we acquire, we can return any IP address. It need not be first, last, etc. However, it can be done serially. Acquiring must skip other previously acquired IP addresses. Hmmm. We read above “a certain” IP.

Releasing must return a specific address to the pool.

A/R operations can fragment the IP addresses.

Finding a free one could get time-consuming.