CLOUD FORMATION

An IAAC tool for AWS

Agenda

1. What is AWS Cloud formation?
2. What are the AWS resources that we can create with ACF
3. Analogies
    a. VPC,
    b. Subnet
    c. Security Groups
4. Use Cases
    a. Github Repo with YAMLs

Rajesh
Rajesh.thiyyagura@gwu.edu

AWS CLOUD FORMATION

AWS CloudFormation is a service that helps you model and provision AWS resources in a repeatable and automated way, using templates. With CloudFormation, you can describe the infrastructure and resources you need in a declarative JSON or YAML template, and CloudFormation takes care of **provisioning and configuring those resources** for you.

This allows for greater automation, repeatability, and consistency in managing infrastructure

CloudFormation is an IAC tool specifically designed for managing AWS infrastructure

Using CloudFormation as an IAC tool can help you reduce errors and manual effort, speed up infrastructure deployment, and increase infrastructure agility and resilience.

AWS Resources

AWS CloudFormation supports a wide range of AWS resources that can be created and managed using CloudFormation templates. Here are some examples of AWS resources that can be created with CloudFormation:

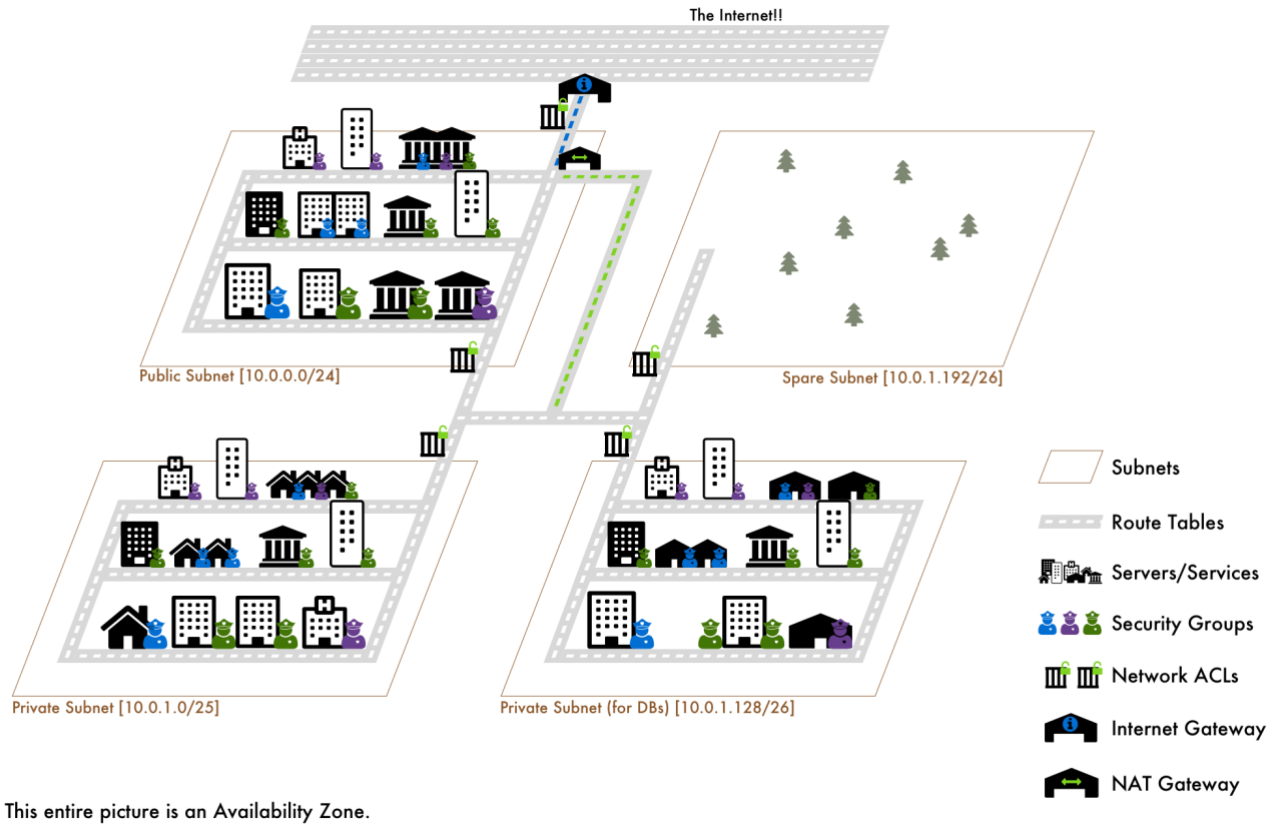
1. Amazon EC2 instances
2. Amazon RDS databases
3. Amazon S3 buckets
4. Amazon VPCs and subnets
5. Amazon API Gateway APIs
6. Amazon Route 53 hosted zones and records
7. AWS Lambda functions
8. AWS IAM users, groups, and roles
9. AWS CloudFront distributions
10. Amazon DynamoDB tables
11. Amazon SQS queues
12. Amazon SNS topics
13. AWS CloudTrail trails
14. AWS CodePipeline pipelines

This is just a partial list of the resources that can be managed with CloudFormation. The full list of supported resources can be found in the AWS documentation.

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html

Analogies

We shall discuss about the following core resources of AWS and understand them analogically
1.VPC
2. Subnet
3. Security Group
4. Route tables
5. Internet Gateway
6. NAT Gateway
5. NACL



VPC - A Virtual Private Cloud (VPC) is a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define. You can think of a VPC as a virtual data center in the cloud, with its own IP address range, subnets, and routing tables.

Subnet - A subnet is a range of IP addresses in your VPC that you can use to launch AWS resources. You can think of a subnet as a subdivision within your virtual data center, with its own IP address range that is a subset of the VPC IP address range. Each subnet is associated with a specific availability zone within an AWS region.

Security Group - A security group is a virtual firewall that controls inbound and outbound traffic to AWS resources in your VPC. You can think of a security group as a security gate that determines which traffic is allowed to enter or exit your virtual data center. Each security group can have rules that allow or deny traffic based on protocol, port number, and IP address range.

Route Table - A route table is a set of rules that determines where network traffic is directed within your VPC. You can think of a route table as a set of directions that tells network traffic where to go when it enters or exits your virtual data center. Each subnet is associated with a specific route table that determines where traffic is routed.

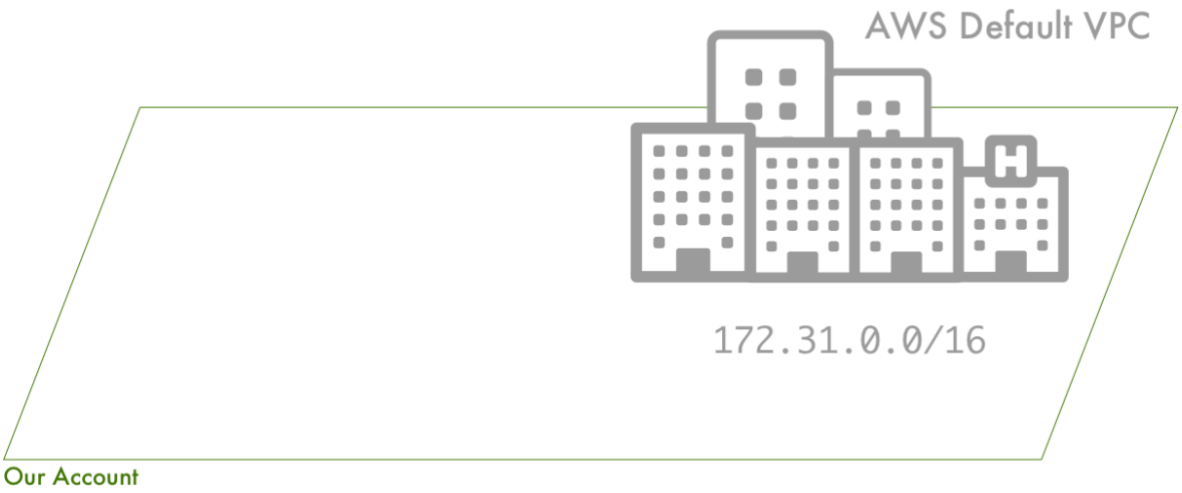
Internet Gateway - An internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between instances in your VPC and the Internet. You can think of an internet gateway as a gateway between your virtual data center and the Internet. It enables instances in your VPC to communicate with the Internet, and vice versa.

NAT Gateway - A NAT Gateway is a service that allows resources within a private subnet of a VPC to connect to the Internet or other AWS services, while remaining hidden behind a public IP address. You can think of a NAT Gateway as a translator that translates private IP addresses to public IP addresses, allowing traffic to flow between private resources and the Internet or other AWS services.

NACL - A Network Access Control List (NACL) is a feature of Amazon Virtual Private Cloud (Amazon VPC) that acts as a firewall for controlling traffic at the subnet level. You can think of an NACL as a bouncer at the entrance of a nightclub, deciding which traffic is allowed to enter and which traffic is denied. NACLs can have inbound and outbound rules that define which traffic is allowed to enter or exit a subnet.

OUR AWS ACCOUNT

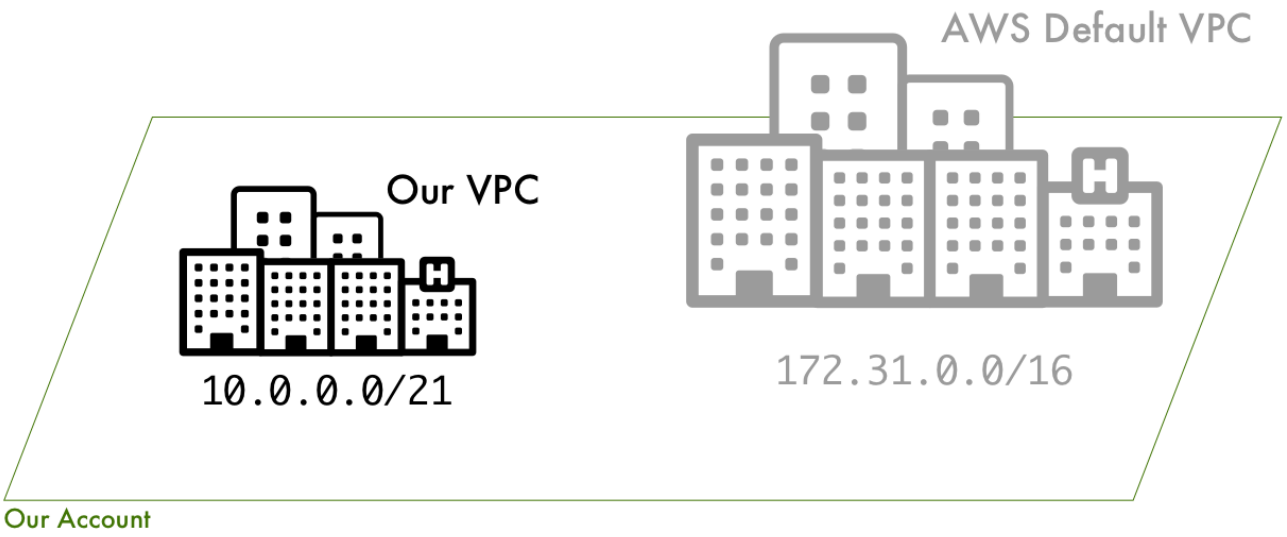
With respect to the analogy we're building, the account is our wide open landmass. The only defined area of this land is the default AWS VPC at 172.31.0.0/16.



VPC (CITY):

Given our wide open landmass(AWS Account), we now want to define a portion of it that will be the boundaries, or city limits, for our new city. How can we do that? Well let's use a range of postal codes like 98101 through 98191. That postal code will represent a defined amount of land. No other cities can be here.

For VPCs, instead of a range of postal codes we define a range of IP addresses.
We do so with CIDR block notation.
Right now 172.31.0.0 through 172.31.255.255 (/16 specifies that range) are taken up by the default AWS VPC, so we can't use those.
Also, we should stick to one of the RFC1918 ranges, so let's roll with 10.0.0.0/x.



Let's make our VPC small and go with `10.0.0.0/21` this will give us everything from `10.0.0.0` through `10.0.7.255`, or 2048 IPs. This will leave `10.0.8.0 - 10.255.255.255` open for future VPCs in this range (not to mention the 172 and 192 blocks).

The Subnets(postal codes):

Our city limits are defined through a range of postal codes, or segmented geographical areas. However, it's a good idea to split this space up even more so rather than start building willy-nilly. Therefore, we'd next seek to divide up city's range of postal codes into individual postal codes

Our city limits are defined through a range of postal codes, or segmented geographical areas. However, it's a good idea to split this space up even more so rather than start building willy-nilly. Therefore, we'd next seek to divide up city's range of postal codes into individual postal codes.

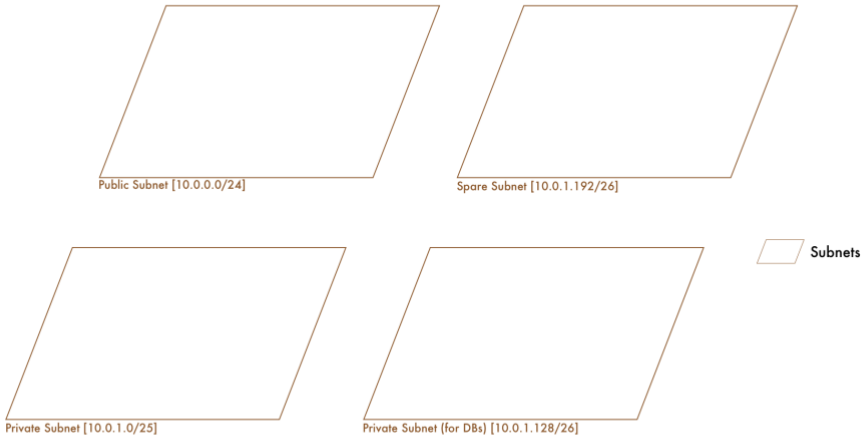
For example, even though our city might include postal codes 98101 through 98191, we don't want to treat that all as just one big space. While putting the airport on the same street as residential could be funny, we probably shouldn't do it. So, let's say we split up the first postal codes like so:

10.0.0.0/24 - Public Subnet for Web Servers (10.0.0.0 - 10.0.0.255 256 IPs)

10.0.1.0/25 - Private Subnet for API Servers (10.0.1.0 - 10.0.1.127 128 IPs)

10.0.1.128/26 - Database Subnet for DBs (10.0.1.128 - 10.0.1.191 64 IPs)

10.0.1.192/26 - Spare Subnet (10.0.1.192 - 10.0.1.255 64 IPs)



Subnet calculator link→ https://www.ipaddressguide.com/cidr

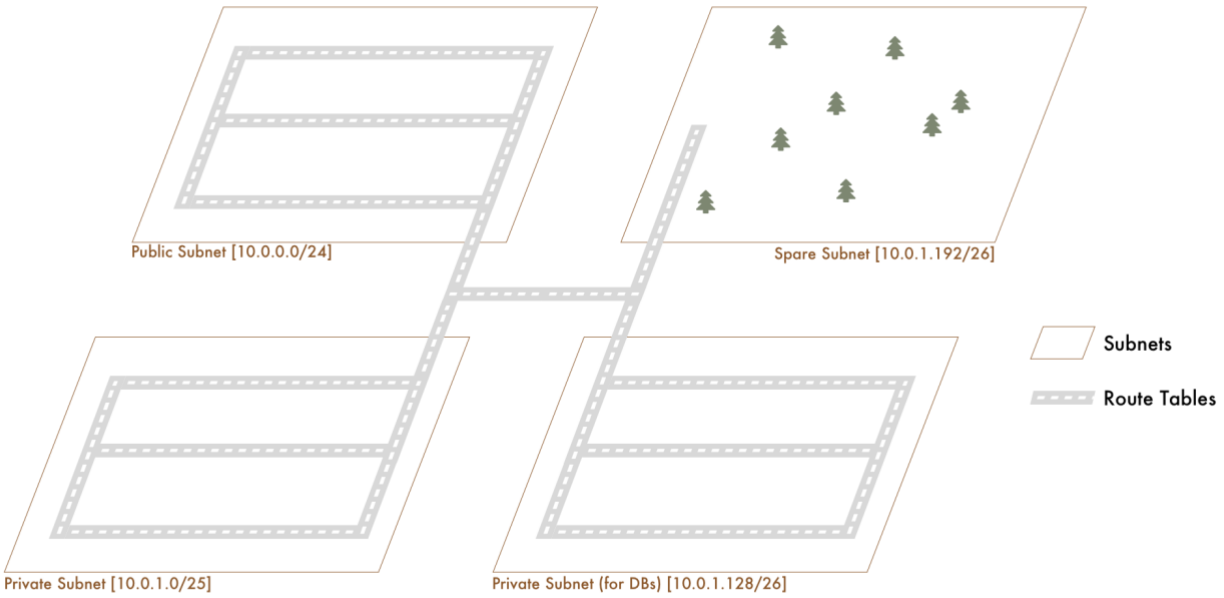
Route table (Roads):

We've defined our separate postal codes within our city. Everything is split into geographical regions. Now we need to give "traffic" a way to move in and out of these different areas. What do we do? We build roads. So a postal code has a series of roads that allow traffic to move in and out of it.

In VPCs, even though we have these different subnets, we need to allow traffic to flow through them. We do this with Route Tables. A Route Table is just a list of CIDR blocks (IP ranges) that our traffic can leave and come from. By default, newly created Route Tables will have the CIDR of our VPC defined. This means that traffic from anywhere within our VPC is allowed.

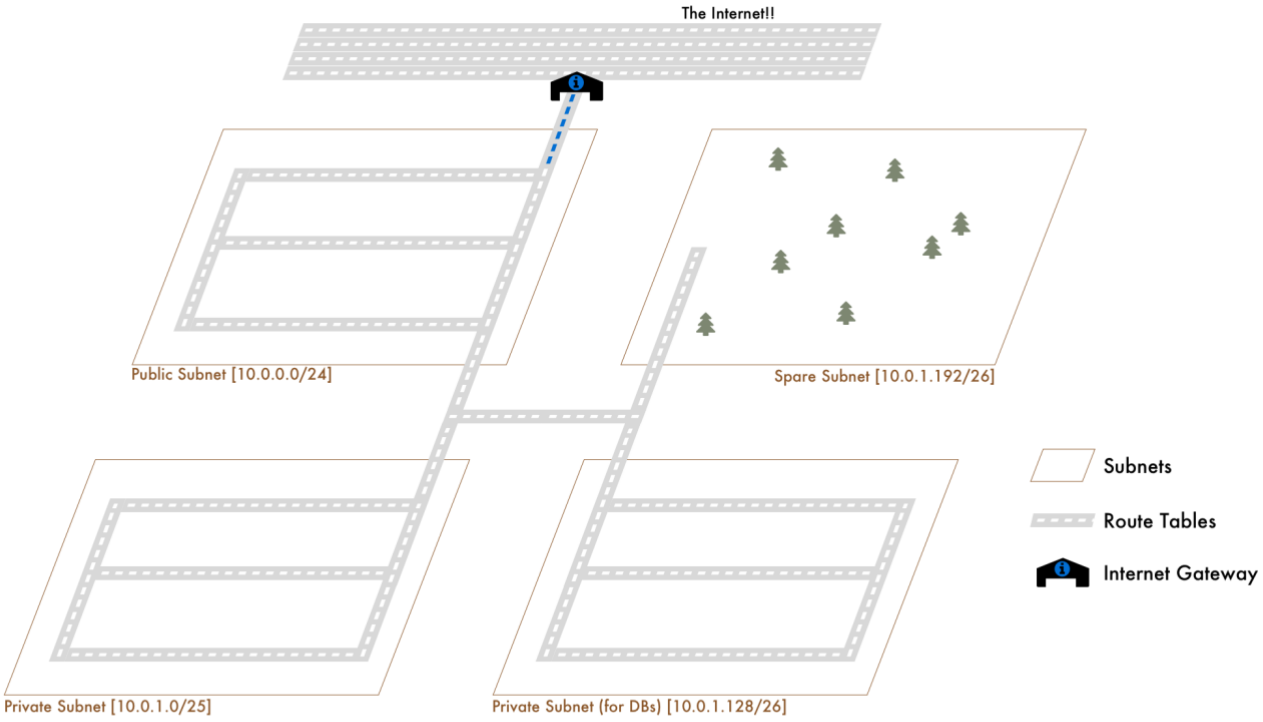
In addition to a list of IP ranges that our Route Table connect traffic between, it also has Subnet Associations. Simply put, these are "which subnets use this route table." For our city analogy it'd be "which postal codes are connected to these roads." A Route Table can have many subnets, but a subnet can belong to only one Route Table.

All comparisons aside, it can be summed up as - A subnet is associated with a Route Table and the Route Table dictates what traffic can enter and leave the subnet.



Public Subnets & Internet Gateway:

A subnet that's associated with a Route Table that's connected to an internet gateway is public. A subnet with a Route Table that's not connected to an internet gateway is private



Example configurations

To cross back over to Subnets, our Route Tables might look like:

Private Route Table

10.0.0.0/21

Public Route Table

10.0.0.0/21

0.0.0.0/0 (internet)

And then our subnets

10.0.0.0/24 - Public (Public Route Table)

10.0.1.0/25 - Private (Private Route Table)

10.0.1.128/26 - Database (Private Route Table)

10.0.1.192/26 - Spare (Private Route Table)

NAT GATEWAY:

When our Subnets connected to the Private Route Table need access to the internet, we set up a NAT Gateway in the public Subnet. We then add a rule to our Private Route Table saying that all traffic looking to go to the internet should point to the NAT Gateway. The tables might look like:

Private Route Table

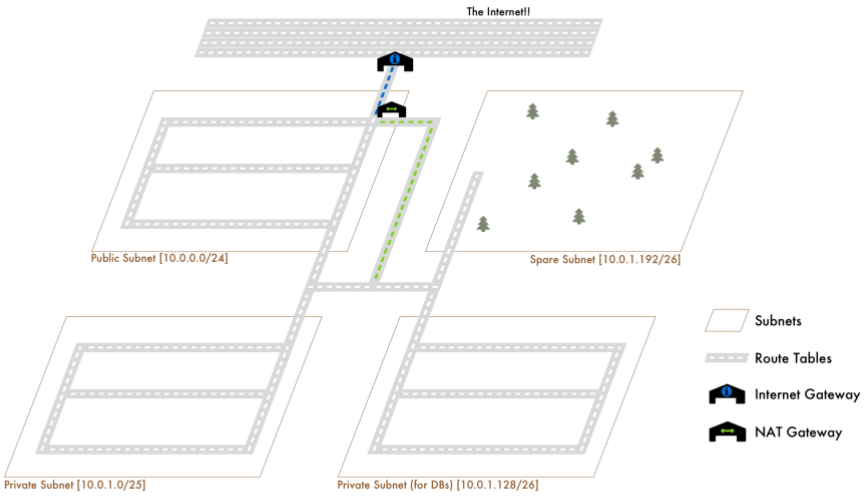
10.0.0.0/21

0.0.0.0/0 (nat gateway)

Public Route Table

10.0.0.0/21

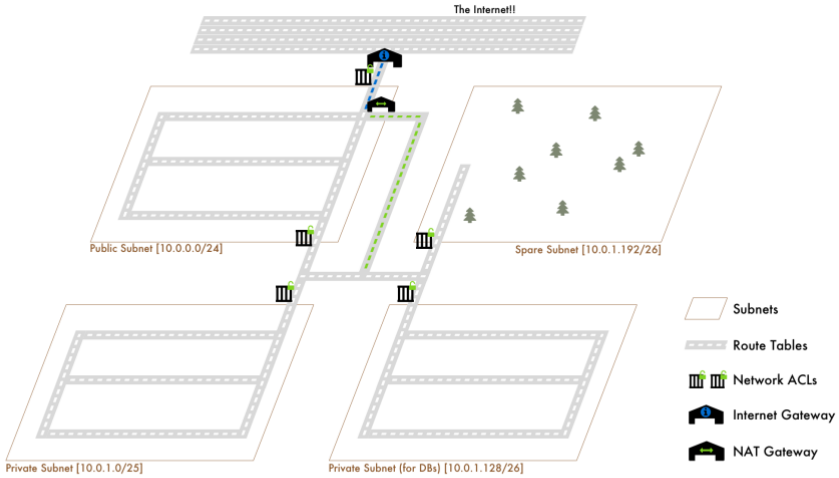
0.0.0.0/0 (internet)



NETWORK ACLS:

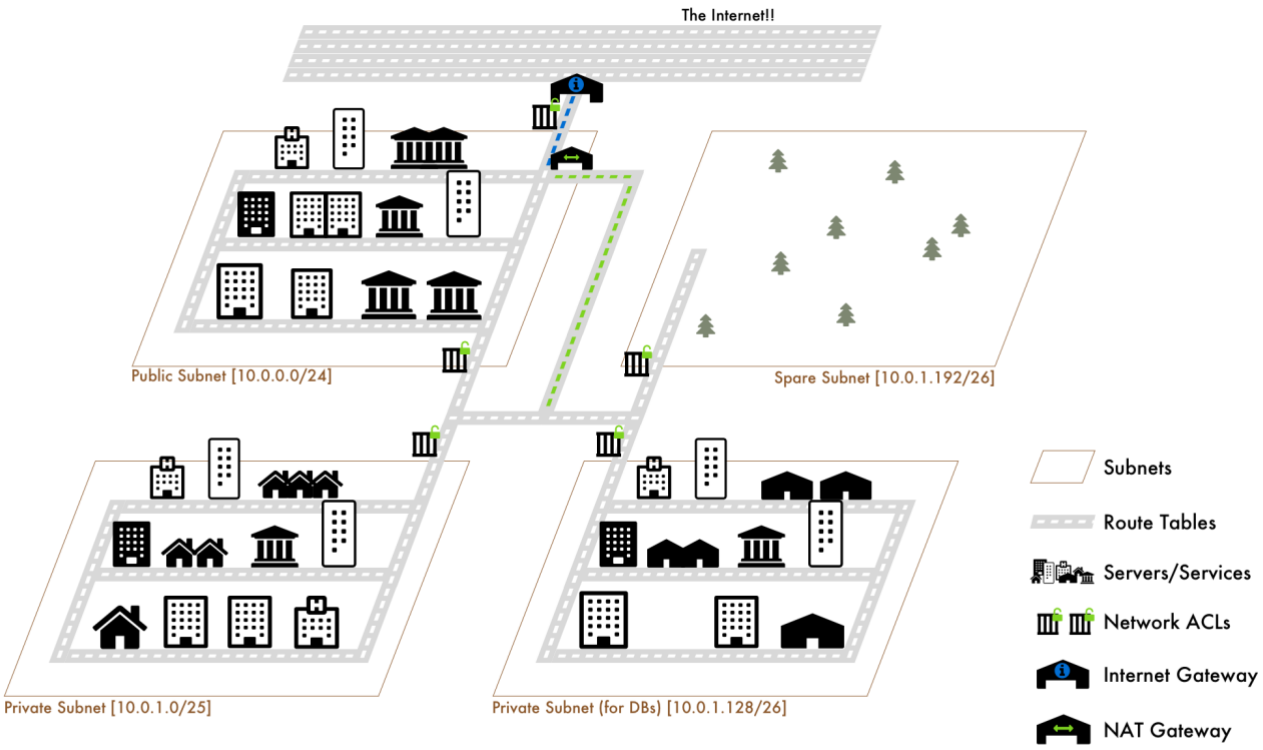
It turns out we've built our city in a very dark time where not everyone can be allowed to just come and go as they please. We need security gates and perimeters outside of each of our postal code areas to ensure only the *right* traffic is entering and leaving. That means we'll check traffic as it comes in to see if it's permitted AND we'll check as it leaves to make sure it can exit.

In our VPC, these are the Network ACLs. They dictate what traffic is allowed to enter and leave the subnets they're associated with.



Webserver and Services (BUILDINGS)

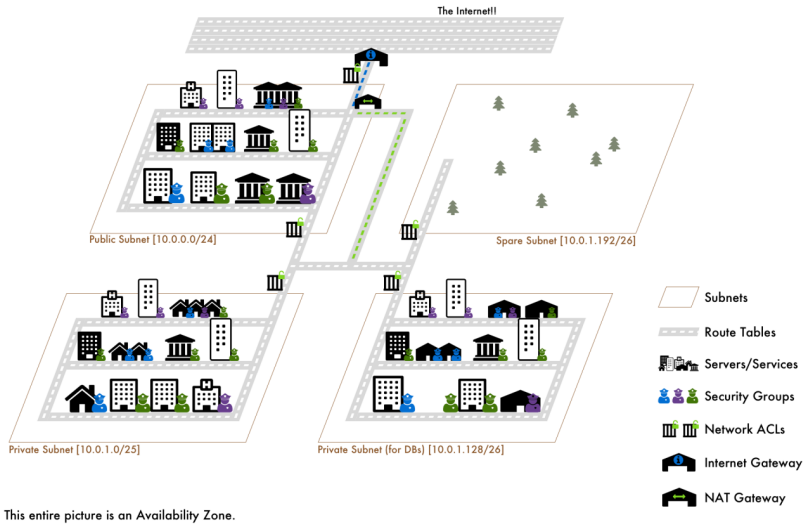
Servers and Services launched into our VPC are the buildings of our city. They receive a "private IP address" when created. We can also setup our subnets to assign "public IP addresses" as well. These are required if we'd like our server/service launched to be able to communicate with the internet via Internet Gateway



Security Groups(Security Guards of buildings):

It turns out that our city isn't just in a dark time, it's practically organized chaos. Because of this we have security guards outside of every building. These guards concern themselves with what traffic is allowed to enter and leave the building. They're not concerned with what traffic is DENIED to enter or leave.

In our VPC, these are our Security Groups. They protect our servers/services at the resource level instead of at a subnet level.



Instead of picking individual security guards, our buildings pick a security company to guard them. The benefit is that buildings that belong to the same security company share the same set of rules.

This means we can define a set of security rules on one Security Group, and have it used on multiple servers and services. You can also tell Security Groups to allow traffic from other Security Groups, which in our analogy would be like saying, "any buildings that belong to Security Corp BRAVO can enter buildings that Belong to Security Corp ALFA"

Availability Zones: Parallel Dimensions

A number of parallel dimensions exist in our world. We rebuild our city in each of those dimensions. If one goes to flames, we can send our inter-dimensional traffic to a different one.

Our VPC can connect to other Availability Zones (AZs). Continue segmenting the VPC IP range into subnets and create the same setup in each AZ. Make the same services available in each AZ as well. Reuse Network ACLs, Security Groups and Route Tables where appropriate. This way if one AZ goes down, our traffic can go to the setup in another one.

Regions: Alternate Realities

A whole new existence with a whole new set of parallel dimensions.

Although you can reuse some things in between regions (like IAM), for the most part, you'll rebuild everything. Even if that means using something like CloudFormation or an automation script.

This wonderful analogy I got this from this blog : https://start.jcolemorrison.com/aws-vpc-core-concepts-analogy-guide/#the-subnets

Use cases

I have created a set of examples in the Github repo:
https://github.com/rajeshreddy-T/IAC_AWS_CloudFormation