Team 7

Linux Networking (792-038)

Project Milestone 1, 2 + updated and 3

DNS as a Service on a Virtual Private Cloud

Arsalan Arsalan (aarsala)
Madhava Anuraag Dasu (mdasu)
Sai Manideep Chippa (schippa)
Pupul Mayank (pmayank)

Project Description

Introduction and Objective

The goal of this project is to build a multi-cloud virtual private cloud environment with DNS as an add-on service.

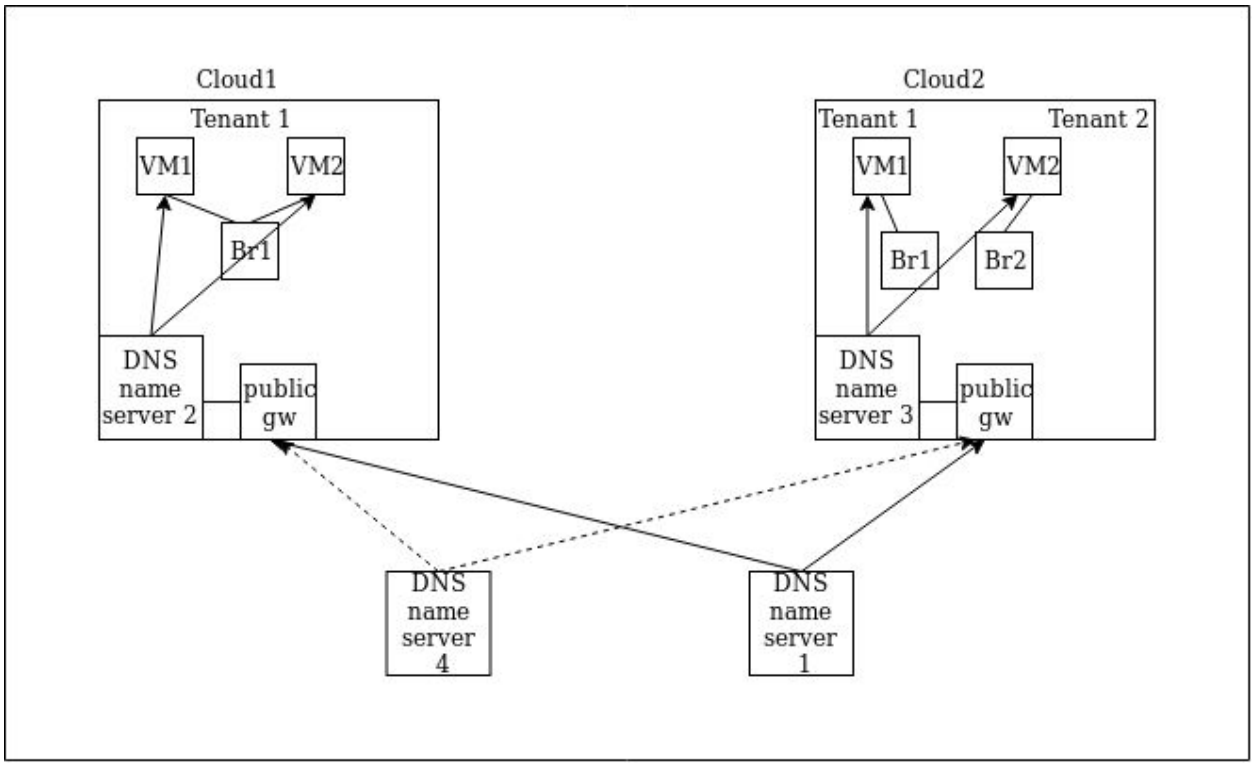
What is multi-cloud?
The idea is to build a cloud infrastructure that offers services to the tenants. Cloud Infrastructure supports tenant applications and this cloud offers CPU, memory, and other resources required by the tenant to run their applications. This is done by creating virtual networks, virtual machines, and other networking devices as per the tenant's requirement. The project aims to make this multi-cloud by making more than one cloud accessible to the tenant. Some declaration of existing work in this field which helped architect this idea are mentioned in the Related Work section.

What is VPC?
In the traditional cloud environment, the tenant does not have the power to control the infrastructure of the cloud. This was originally referred to as the public cloud. One new idea was to offer the customer control over the infrastructure of the cloud. This cloud is isolated from each customer making it a private cloud for the customer. Since the customer has ownership of the network, he can decide to use it as per his requirements as opposed to fixed resource availability. One more thing to note is that all the networks in the cloud are private, hence they are not routable on the public Internet, making it a truly private cloud.

Add-on service our cloud provides:
The next goal of this project is to provide an add-on service along with providing Infrastructure as a cloud service to make this solution more lucrative for the customers.
The service provided to the cloud users here is DNS.
Domain Name System (DNS) is a system which provides a translation of domain names to IP address. DNS system in the cloud should be designed in such a way that it provides the end-users a route to the applications deployed in the cloud by translating names into numeric IP addresses. Cloud DNS system should be highly available and is a relatively easy way in making the applications and services of the tenants of the cloud accessible to the end-users.
We propose to provide the DNS service as a hierarchical distributed database architecture. A single DNS server cannot handle the huge global DNS traffic. It also provides a way for easy to update DNS servers and easy maintenance.
In addition, we would add some functional and management features(which are described below) to our solution.

Architecture

The basic networking architecture of the project is presented below:



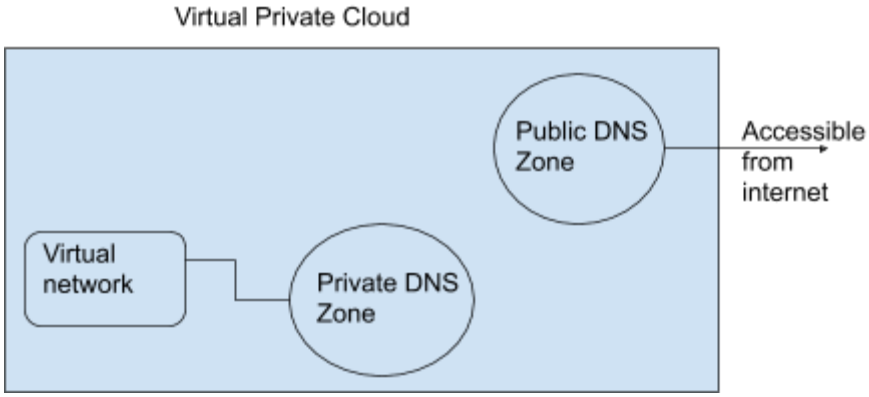
In this representation, there are two clouds connected via tunnels (not shown in the architecture), Cloud 1 has a single tenant which has two servers and Cloud 2 has two tenants with a server per tenant.
There is a DNS name server that resolves the name request it receives from the user to the Public gw of the second hierarchical name server which serves the requests within the cloud.
Here DNS name server4 is a redundant DNS name server for name server 1 which takes up the DNS operations if for some reason the primary name server fails.
Suppose the DNS name server gets a request to access the server of tenant 1, then the name server provides the IP address of the name server of the cloud which has tenant 1.
Now inside a cloud, for example, a cloud 1, we have two servers that belong to the same tenant. The DNS name server inside a cloud resolves the name request to the IP addresses of the servers.
We can extend the DNS name server itself to perform a NAT operation so that the cloud private networks are not routable in public internet.

The proposed solution will provide the following functional and management features:

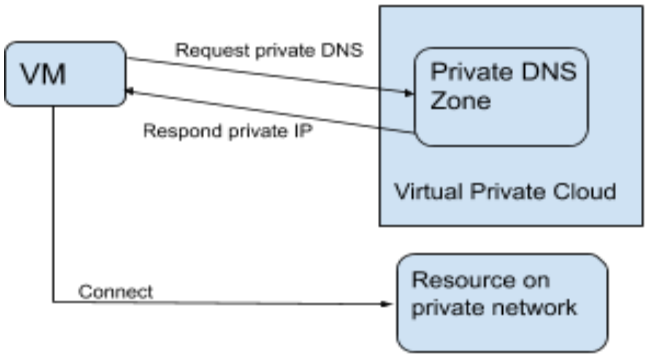
Functional features:

1. Public and Private Zones:

One of the functional features of the DNS as a service that will be provided as a part of our virtual private cloud will be the presence of private zones along with public zones. Public zones are the ones which are visible from the internet.
The public zones will be containers which will contain information as authoritative name servers and respond to queries, irrespective of their originating point. A tenant will have the flexibility to create DNS records such that they can be published directly on the internet. The name servers which are registered with domain name registration will need to point to this virtual private cloud's name servers.



Private zone/(s) will be containers of DNS records which cannot be accessed from the internet directly but be accessible only by resources which are present in the same project. A tenant will have to authorize the VPC networks that will have access to this zone. Instead of using the domain names provided by our VPC's DNS, the tenant will have the flexibility to provide their own custom names for virtual machines within one or multiple virtual networks. For the resolution to happen over private zones from the virtual network, it must be linked to the private zone. It would also support reverse DNS lookup for a private IP within the virtual network. Since this zone will be accessible only by a private network, hostile agents will not be able to get information about the internal network, thus adhering to the security requirements of an organisation.



2. Load Balancing:

The other functional feature which will be provided in our project is DNS load balancing.

DNS load balancing is the practice of configuring a domain in the DNS name server such that client requests to the domain are distributed across the group of the available server machines of the same tenant.
If the tenant application is hosted in different instances of the cloud. We can distribute the traffic across the instances by implementing a DNS load balancer.
Here the load is the number of requests received by the name server. This load is balanced by resolving the domain name to the IP addresses of the servers. This resolution can happen by predefined algorithms such as Round Robin and Weighted Round Robin.
We can implement this dynamically also where the DNS name server keeps checking the health of the servers and accordingly weights would be set to mitigate the load.
We propose to have two levels of DNS load balancing where the first name server serves the requests by providing the Public Gateway IP addresses of the different clouds. Our next name server in the hierarchy serves the requests by providing IP addresses of servers that belong to the same tenant.

3. DNS Forwarding:

DNS forwarding is the process by which particular sets of DNS queries are handled by a designated name server as opposed to the name server which was contacted initially. We can have a particular subset of queries being resolved a dedicated name server. So when other name servers receive a request for those subset, they would forward the request to the designated name server for those subset.
We propose to have a designated name server to handle external requests and separate name servers for internal name server.

Management features:

1. DNS Security:

Standard DNS queries, which are required for almost all web traffic, create opportunities for DNS exploits such as DNS hijacking and man-in-the-middle attacks. These attacks can redirect a website's inbound traffic to a fake copy of the site, collecting sensitive user information and exposing businesses to major liability. We propose the following solutions to overcome these security threats.

A DNS firewall is a tool that can provide a number of security and performance services for DNS servers. A DNS firewall sits between a user's recursive resolver and the authoritative

name-server of the website or service they are trying to reach. The firewall can provide rate-limiting services to shut down attackers trying to overwhelm the server.
We will also use iptables which is an application that allows users to configure specific rules that will be enforced by the kernel's netfilter framework. It acts as a packet filter and firewall that examines and directs traffic based on port, protocol and other criteria.

We also propose to implement the DNS auditing where we are logging the actions of specific events which provides the means to investigate an attack, recognize resource utilization or capacity thresholds, or to simply identify an improperly configured DNS system. If auditing is not comprehensive, it will not be useful for intrusion monitoring, security investigations, and forensic analysis. It is important, therefore, to log all possible data related to events so that they can be correlated and analyzed to determine the risk.
Data required to be captured includes: whether an event was successful or failed, the event type or category, timestamps for when the event occurred, where the event originated, who/what initiated the event, affect the event had on the DNS implementation and any processes associated with the event.

2. DNS Scalability:

Scalability is one of the important features in any network or cloud infrastructure. There are two ways in which scalability policy can be implemented - scale-out and scale-up. Scaling up is increasing the capacity of already existing resources to accommodate more load in the same Linux box. Scaling out is increasing the number of Linux boxes that can accommodate the load parallelly.
We propose to implement a scale-out policy in the project. We will be making the following decisions:
When to scale out?
How much to scale out?
Where to scale resources?

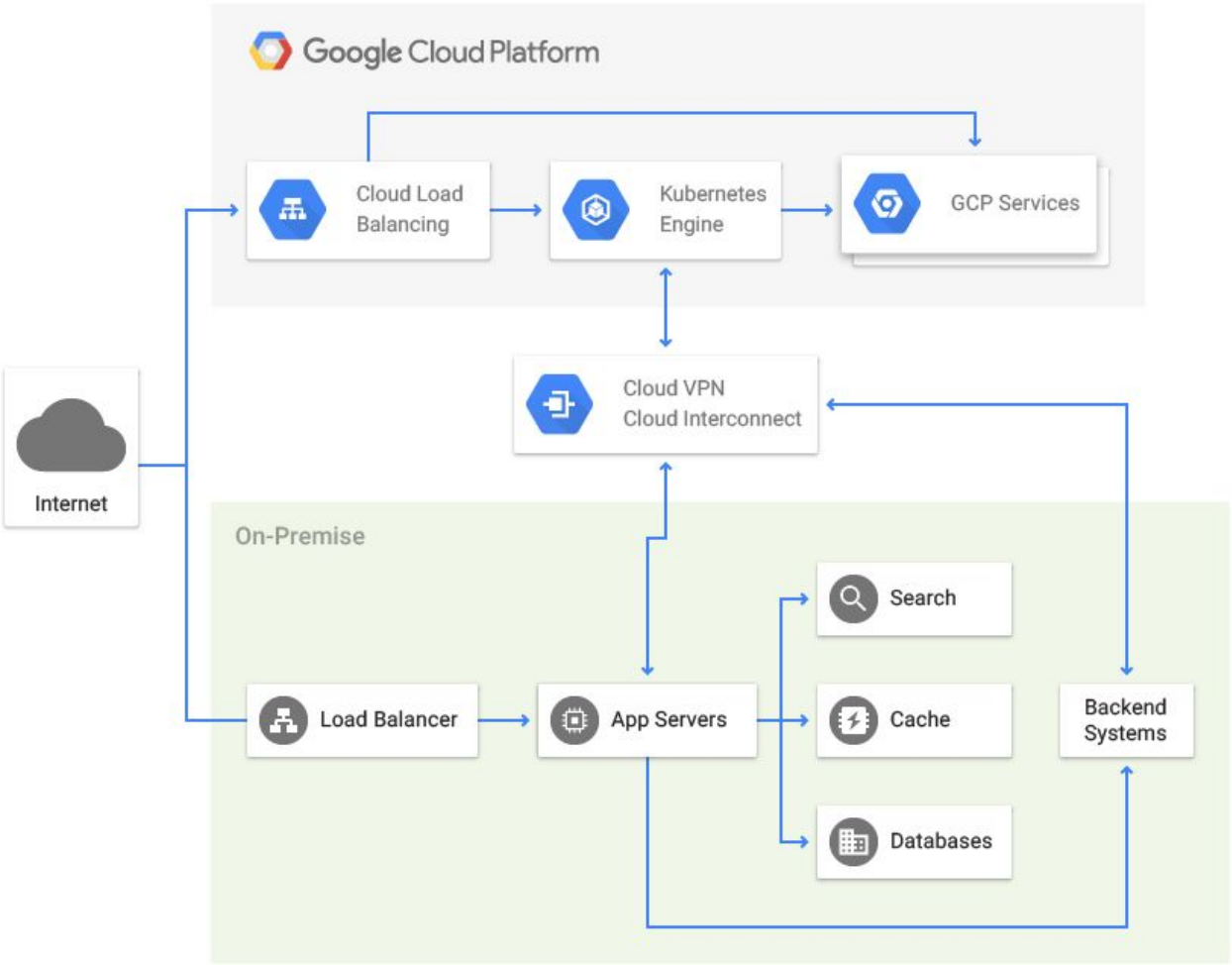
3. Fault-Tolerant:

DNS name servers are one of the major points of attack, as they are the primary communication point for any request. If the dns server becomes unavailable because of a fault or an attack(which the firewall did not detect), then the users cannot reach the tenant applications hosted on the servers in the cloud.
The main goal here is to avoid downtime and maintain full connectivity.
To tackle this problem we propose to implement the fault tolerant policy in the project by creating a new name server when a fault has been detected. The new name server would have the same database as the faulty name server before the fault was detected.The system would reroute the request to the new name server which serves the requests. A centralized database is required to implement this policy.

This entire implementation would be automated in the project.

Related Work

1. Google Cloud DNS:
   Google provides multiple types of cloud architecture depending on tenant requirements. The related work compared here is the hybrid cloud with on-premises data and applications as shown in the figure below:



   A google cloud platform console project is a container that provides services such as resources, a domain for access control, and the place where billing is configured and aggregated. Google Cloud DNS is claimed to be a reliable, scalable, and managed authoritative Domain Name System (DNS) service. It runs on the same infrastructure as Google. It allows tenants to have high availability, low latency and a cost-effective way to make their services and applications available to their users. Cloud DNS translates requests for domain names like www.google.com into IP addresses like 74.125.29.101 as shown in the figure below:

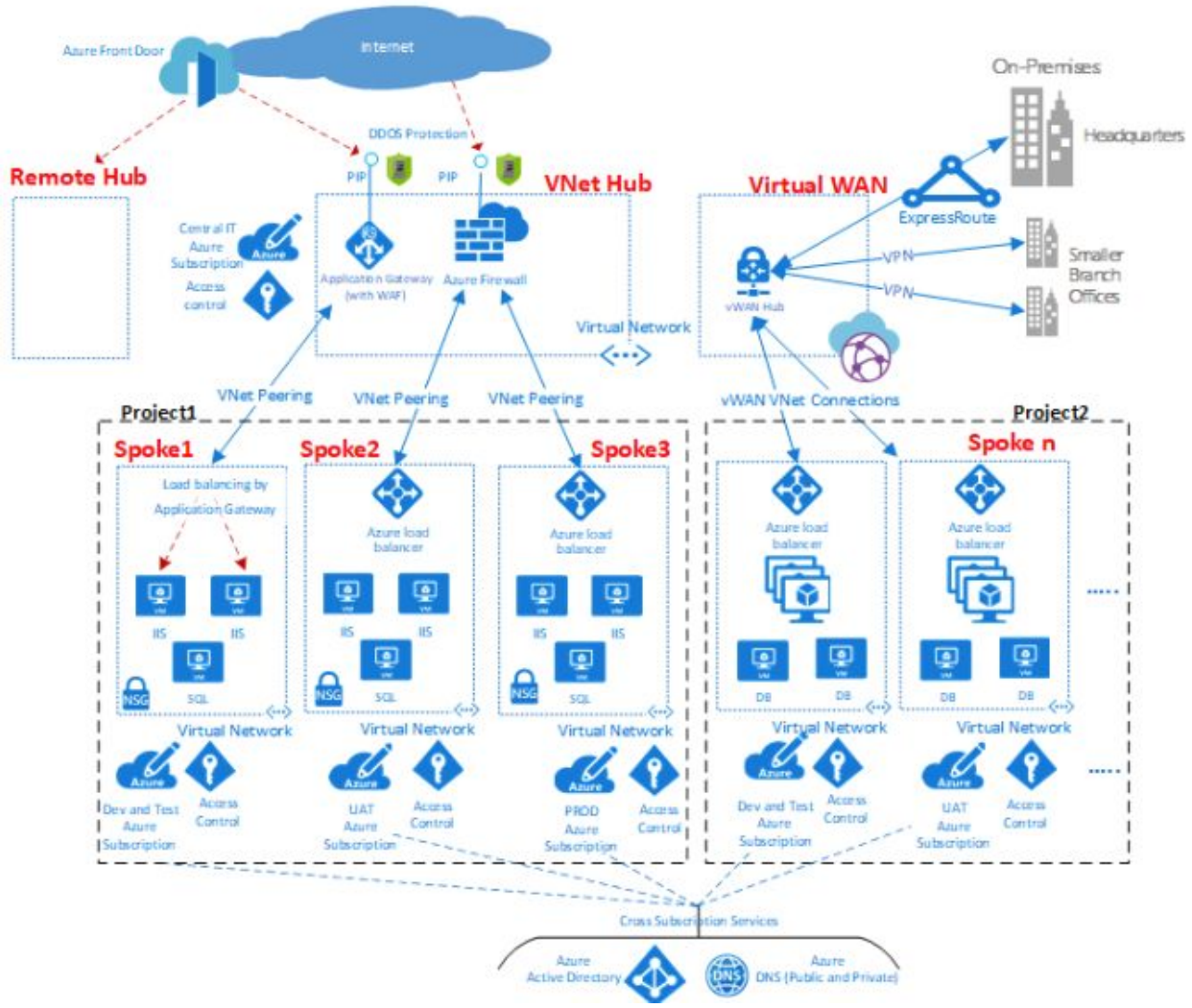
Every resource of this Cloud DNS lives within a project and every operation on this Cloud DNS must necessarily specify the project to work with.
Some of the features offered by google cloud DNS are:

- Managed Zones: Public, Private, Forwarding and Peering zones
- DNS Forwarding: It supports both inbound and outbound DNS forwarding using DNS server policies. DNS forwarding is set here between non-GCP name servers and GCP's internal name servers.
- Resource Record Changes: It supports changing in resource record sets in bulk.
- DNS Server Policy
- DNSSec

GCP also supports automatic scaling, where a tenant can create and update millions of records and handle large query volume without any intervention.

2. Azure Cloud DNS:

Microsoft Azure cloud also provides hosting service for DNS domains. It performs name resolution by using Microsoft Azure infrastructure. Tenants can host their domains in Azure and manage their DNS records using same credentials, APIs, tools, and billing as their other Azure services.
Hub and spoke is the model used by MS Azure Cloud as shown in the figure below..A hub is the central network zone that controls and inspects ingress or egress traffic between different zones: internet, on-premises, and the spokes. The role of each spoke can be to host different types of workloads.



Some of the features provided by Azure cloud DNS are:

- Role Based Access Controls
- Activity Logs
- Resource Locking
- DNSSec
- Alias Records
- Disaster Recovery using Azure DNS and Traffic Manager

The Microsoft global network of name servers has the scale and redundancy to give ultra-high availability for tenant's domains.

References:

1. Google cloud DNS: https://cloud.google.com/dns/docs/overview#shared-vpc
2. Azure cloud DNS: https://docs.microsoft.com/en-us/azure/dns/dns-overview
3. Route 53 DNS: https://aws.amazon.com/route53/

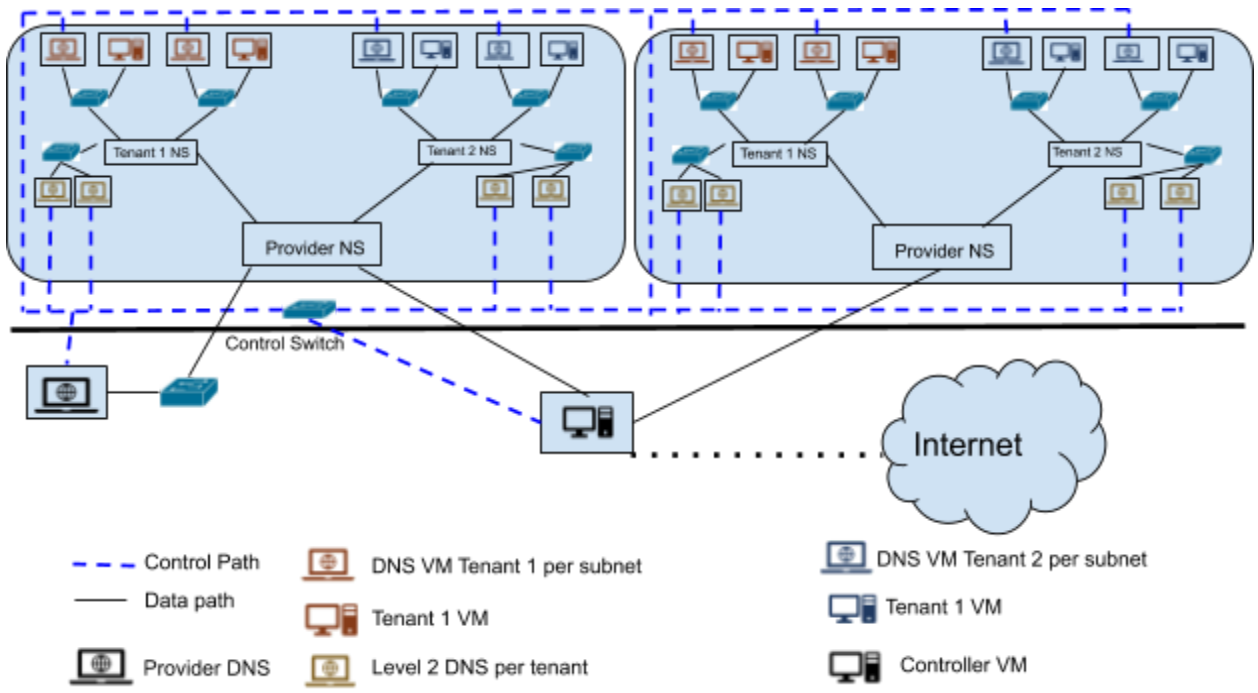
Milestone 1 feedback:

- What is the use of server 1 and 4?
- Architecture is missing multitenancy. DNS is in control plane, no need to tie it with datapath
- Need more explanation of service to the tenant
- Functional Features:
    - Content Based Routing
    - Load Balancing
- Management Features
    - Choose one between fault tolerance and scalability

  No mention of Route 53 DNS

Milestone 2

Updated Architecture:



Component Description:

1. **Provider Space:**
    a. **Provider Namespace:**
    The purpose of provider namespace is to connect the virtual private cloud to the internet. All the traffic from the internet to the VPC and vice versa goes through the provider namespace. This is done by running masquerade rules on provider namespace. It also prevents communication across multiple tenants.
    b. **Provider DNS:**
    The purpose of provider DNS is to forward DNS traffic to the Layer 2 DNS servers and get responses which need to be returned to the clients. It acts as the client-facing public DNS server. All the DNS queries to the VPC go through the provider DNS. This is the top level domain for the hierarchical DNS architecture provided by this service.
    c. **Controller Switch:**
    This is a L3 controller switch which is connected to all the DNS VMs. The purpose of this switch is to provide a control path between the DNS servers.
    d. **Controller VM:**
    The controller VM is the access VM for provider to run all the possible scripts to create an infrastructure as requested by a client. This acts as the provisioning server and uses controller switch to perform all the provisioning operations.

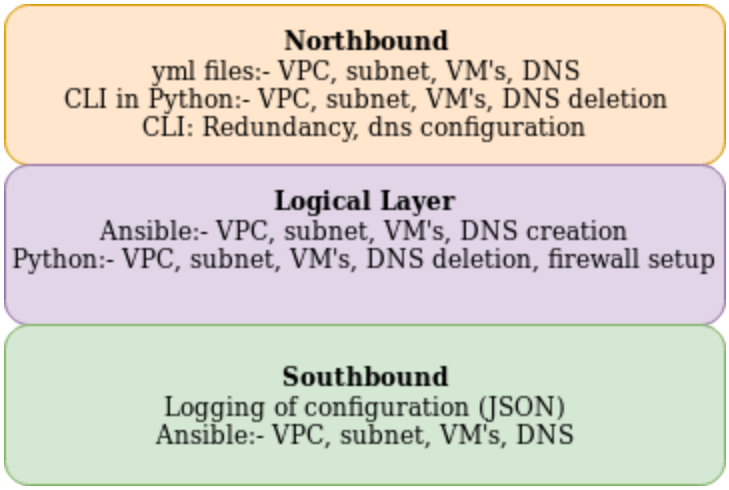
2. **Virtual Private Cloud Space:**
3. **Tenant Namespace:**

   There is a namespace created for each tenant in the virtual private cloud. Every tenant will be provisioned a different namespace to attain L2 and L3 isolation.
4. **Tenant Bridges:** There are separate bridges connected to each namespace for different subnets per tenant. VMs in one subnet can access VMs in another subnet using this L2 bridge through the namespace. This is to create L2 isolation in each subnet of a tenant.
5. **Per subnet tenant VMs:** These are the VMs which will host various applications to be used by a tenant. These VMs are connected to tenant namespaces through subnet specific tenant bridges.
6. **Per subnet tenant specific DNS servers:** These are the tenant's DNS servers which will act as authoritative name servers for tenant VMs/applications. The final response for all the queries will come from these servers if there is no entry present in Level 2 DNS servers.
7. **Level 2 DNS servers:**

   These DNS servers are to support load balancing and high availability for DNS queries. These DNS servers have DNS forwarding implemented to perform load balancing. They are configured to respond with different authoritative name server responses when multiple queries come for the same application.
   a. **Primary DNS server:** Both the servers together create a high availability setup such that the secondary server takes over only when primary goes down.
   b. **Secondary DNS server:** This server stays in slave mode till the time primary server is active and responding to queries.

Northbound , Southbound and Logical Layer explained:

1. The DNS (provider DNS) at the hypervisor is the top level domain. All the north south traffic is resolved by this DNS. This server maintains the records of each of VPCs in the hypervisor.



2. Each tenant is allocated a unique namespace to maintain isolation among multiple tenants. Each of these namespaces has multiple subnets each served by a veth pair interface executing dnsmasq on one end, and a switch on the other. Because of this, any VMs created in this subnet are automatically assigned IPs.
3. Each namespace also has 2 DNS servers running in it at the namespace level(Level2).Both of them execute the keepalived to ensure high availability in case of DNS VM failures (Level2). Every subnet is also allocated a DNS server, at the subnet level.(Level3).
4. This Level2 DNS nameserver is responsible for maintaining the records for every authoritative Level3 DNS server and points to them.
5. All the DNS servers are connected to a controller switch and the controller VM(hypervisor). Controller VM is responsible for creating new infrastructure for the tenants.
6. All queries for the north-south traffic end up at the TLD, Level1 DNS server. The DNS server looks up its forward zone records. If a zone record exists and can be resolved locally it returns the A record for the particular domain.
7. The Level2 server looks up its records and if it can be resolved locally, returns the A record for the particular query, else returns a “record not found” message to the Level1 server. If the authoritative nameserver pointer exists, it forwards the query to the particular Level3 which is responsible for that zone(content-based routing).
8. The Level3 DNS server responds to the query with the IP address of the server responsible for that domain (round robin or static).

Steps to set-up DNS:

Following are the steps for configuring tenant DNS VMs. All the steps mentioned below are automated except some post-configuration which is required in the authoritative name server.

9. VM images with preinstalled Bind9 packages have been provided for DNS VM configuration.
10. Run the script sudo python Nameretrieve.py

11. This script provides the level wise DNS VM names (L3) and their respective IP addresses required to set up the DNS configurations. This script makes it easy to set-up DNS.
12. Configure the DNS server by editing the /etc/named.conf file
    vim /etc/named.conf
13. Add IP of the DNS VM in the options as follows:
    listen-on port 53 { 127.0.0.1; <IP of DNS-VM>; }
14. Add IPs to be allowed to query the DNS in the allow-query as follows:
    allow-query{ any; }; // where any specifies that any IP can query the DNS.
15. Define a zone by editing the /etc/named.conf file
    An example zone has been given below.
    zone "." IN {
        type hint;
        file "named.ca";
        };
16. Create zone files for each zone in /var/named/ directory as follows:

```
$TTL 1D
@       IN SOA  @ rname.invalid. (
                                        0       ; serial
                                        1D      ; refresh
                                        1H      ; retry
                                        1W      ; expire
                                        3H )    ; minimum
@           NS     <zone-ns>
@            A     <ip-of-zone-ns>
tenant       A     <ip of tenant DNS>
```

5. Enable and initiate the DNS service as follows:
   systemctl enable named
   systemctl start named
   If errors are returned, use the following commands to know more about the errors:
   <systemctl status named.service" and "journalctl -xe"
6. Allow DNS service through the firewall using the following commands and restart it:
   firewall-cmd --permanent --add-port=53/tcp
   firewall-cmd --permanent --add-port=53/udp
   firewall-cmd --reload
7. Verify the named.conf configuration using the following commands:
   named-checkconf/etc/named.conf
   It returns nothing if there are no errors. If it returns errors, check named.conf for errors.
8. Verify the zone configuration using the following commands:
   named-checkzone <zone-ns> /var/named/<zone-filename>
   It returns OK if no error is found and loads the zone file.

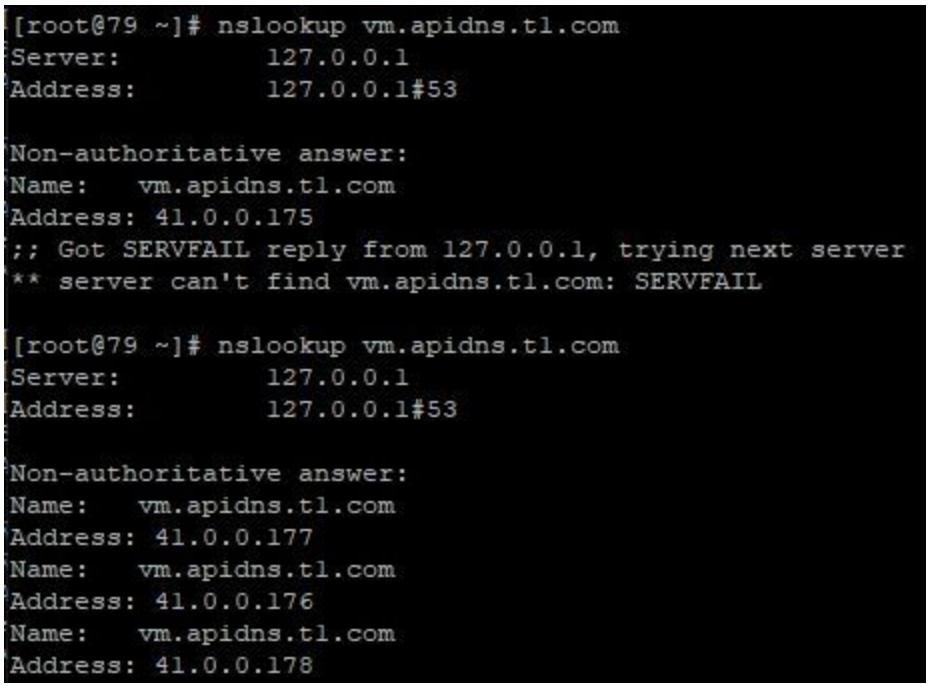
Functional features implementation:

17. Content Routing

The service provides content routing feature such that a client can reach an application which is running using the DNS queries. For an example: a query to project.ece792.CSC.NCSU.com would point to ece792 VM using hierarchical DNS query responses. Provider DNS (NCSU.com) would provide the details of Level 2 DNS (CSC) server by looking up its entries, Level 2 DNS would then respond with the details of authoritative name server which is L3 DNS server for a specific subnet (ece792) by looking up its entries. L3 DNS server would then provide the response of application IP (project) by looking up its entries in the zones created. The client would then be able to access the VM through the data path provided by the cloud infrastructure. The data path would traverse through provider namespace, to tenant space to tenant bridge to tenant VM.

18. Load Balancing

Since DNS is the most important and the basic part of the network, we will have many DNS requests coming to the DNS servers at Level 3 DNS servers. So to manage that and make sure the performance of the network does not go down, we have used the DNS round robin load-balancer to load balance the traffic. Round Robin DNS is a technique in which load balancing is performed by a DNS server instead of a strictly dedicated machine. A DNS record has more than one name-IP address pair. When a request is made to the DNS server which serves this record, the answer it gives alternates for each request. The same can be verified in the below mentioned screenshot:



On the Level 2 DNS server, there is load balancing based on DNS forwarders also. When a request reaches the Level 2 DNS server, it is forwarded to multiple DNS servers. The sequence in which it is sent to each forwarder which is the authoritative name server is randomized for each request. Thus, the DNS server which responds back with the query first is responded back to the client.

Management features implementation:

19. Security:

We have added the following rules to drop bogus DNS queries which have these hex strings. This is to protect DNS from analysing any sort of random queries and thus saving DNS cycles. Queries like 6Gdb1QlP.f.proxypipe.net., mhl00ULG.e.proxypipe.net., clacqxlG.f.proxypipe.net. will be dropped through these iptables rules. Such packets are often used in DDOS attacks over DNS.

iptables -A INPUT -i eth0 -p udp --dport 53 -m string --hex-string "|09|proxypipe|03|net" --algo bm -j DROP
iptables -A INPUT -i eth0 -p udp --dport 53 -m string --hex-string "|06|kitten|02|ru" --algo bm -j DROP
iptables -A INPUT -i eth0 -p udp --dport 53 -m string --hex-string "|03|www|07|puppies|04|woof" --algo bm -j DROP

The DNS server also provides security against multiple ping requests. The iptables rule implemented for this are:

iptables -t mangle -A PREROUTING -p icmp -m hashlimit --hashlimit-name icmp --hashlimit-mode srcip --hashlimit 20/second --hashlimit-burst 1 -j ACCEPT
iptables -t mangle -A PREROUTING -p icmp -j DROP

All the DNS servers are also performing logging to various queries being made to the DNS server. This has been implemented in the bind daemon itself. This is to make sure that auditing of queries made to the DNS server can be done.

There are iptables rule added to protect DNS server from any other traffic by allowing only UDP traffic over port 53.

20. High availability:

The VRRP or Virtual Router Redundancy Protocol helps you create a reliable network by using multiple routers in an active/passive configuration. If the primary router fails, the backup router takes over almost seamlessly. A similar implementation has been made to provide high availability of DNS servers.

Clients connect to a virtual IP-address. It is called virtual because the IP-address is not hard-coded to a particular interface on any of the routers.The primary and the secondary router IPs are configured by the DHClient.

We need to install keepalived and modify the keepalived.conf file. The file must be edited with the information about the primary and secondary router IP, State(Master/Backup), Priority, interface to be configured on and virtual id which should be the same on both the master as well as backup.
The virtual IP has to be configured manually which acts as the gateway for the L1-DNS to reach the active server. Because the virtual IP-address is not configured on any of the interfaces, Linux will not reply to any packets destined for this IP. This behaviour needs to be changed or VRRP will not work. Edit /etc/sysctl.conf and add this line:

We start the keepalived service. Whenever there is a failure of the master server, in a case where it goes down, the backup server becomes the active one now and it is accessible via the same IP address.

Below is the configuration for master and backup in the keepalived.conf

```
/* Configured on Master */
vrrp_instance VI_1 {
    interface eth0
    state MASTER
    virtual_router_id 51
    priority 101

    authentication {
        auth_type AH
        auth_pass monkey
    }

    virtual_ipaddress {
        192.168.230.200
    }
}

/* Configured on Backup */
vrrp_instance VI_1 {
    interface eth0
    state BACKUP
    virtual_router_id 51
```

```
    priority 50

    authentication {
        auth_type AH
        auth_pass monkey
    }

    virtual_ipaddress {
        192.168.230.200
    }
}
```

Setting up provider infrastructure:

21. Setting up Provider DNS:

Creation of Provider DNS VM is taken care by automation scripts. Once the VM is created, there are some basic configurations required on provider DNS which needs to be performed manually.

Editing the vi /etc/named.conf:
Under *options* section, look for listen-on port 53 and add an entry for the IP of the provider DNS server as mentioned below and add an entry to filter AAAA queries on IPv4:

listen-on port 53 { 127.0.0.1; <Provider DNS IP>; };
filter-aaaa-on-v4 yes;

Add a zone entry in the same file as:

zone "com." IN {
type master;
file "/var/named/fwd.root.com.db";
allow-update { none; };
Allow-transfer { any; };
};

Editing the /var/name/fwd.t1.db (zone) file:
A nameserver to IP mapping needs to exist in this file such that DNS knows that this query can be responded by which server. For example:

```
$TTL 1D
@       IN SOA  t1.com. root.t1.com. (
                                        11      ; serial
                                        5       ; refresh
                                        60      ; retry
                                        5       ; expire
```

```
                                5 )     ; minimum

@       IN  NS          t1.com.
@       IN  A           192.168.230.131
vm.apidns       IN  NS          apidns.t1.com.
apidns  IN  A           192.168.230.235
        IN  A           192.168.230.105
testvm1         IN  A   100.0.0.35
```

Here 192.168.230.235 and 192.168.230.105 are the Level 2 DNS IPs for multiple subnets.

22. Creating Provider Namespace:

- We have a provider namespace already created in the hypervisor before the scripts are run. The automation scripts written using ansible and python would take care of adding routes, providing IP addresses, and adding respective iptables rules in the provider namespace.
- The manual configuration done in provider namespace is to provide connectivity to the internet. This is done by adding nat rules in iptables.
- This is achieved by adding a vethpair with one end in the provider namespace and the other end in the hypervisor.
- We have added a vethpair with the IP address as 10.10.10.1/24 in the namespace and 10.10.10.10/24 in the hypervisor.
- There are iptables rules in the hypervisor which does source nat such that the provider namespace is reachable to the internet.
- The iptable rule we have used is :
  Iptable -t nat -I POSTROUTING 1 -s 10.10.10.0/24 -o ens3 -j MASQUERADE
- This rule does the necessary nat operation to make the provider namespace able to reach the internet.

23. Creating Controller bridge and controller VM:

- The controller bridge and controller VM need to be created manually before the scripts are run. This is a one time task which needs to be performed on the provider space. The controller bridge has an interface connected to each DNS VM in our architecture. The purpose of this controller bridge is to provide a control and management path to the tenants to the DNS servers.

Github link to implemented code:
https://github.ncsu.edu/schippa/DNS-as-a-Service-VPC

All the scripts and READ ME is also attached in the ZIP file uploaded to google drive.

The yml scripts and templates are located under PROJECT3/relevant folder in the above mentioned link.

References:

24. **Creating DNS:**
    https://www.isc.org/bind/
    https://www.unixmen.com/setting-dns-server-centos-7/
    https://www.itzgeek.com/how-tos/linux/centos-how-tos/configure-dns-bind-server-on-centos-7-rhel-7.html
    https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-centos-7

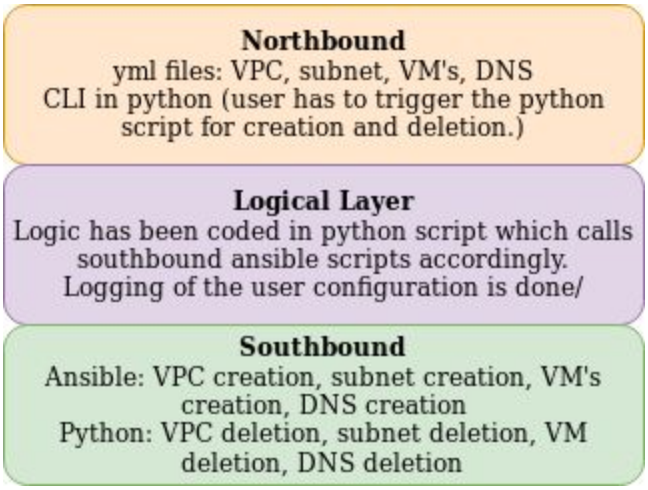
25. **High availability on DNS:**
    https://louwrentius.com/configuring-attacking-and-securing-vrrp-on-linux.html
26. **Security:**
    https://www.perturb.org/display/1186_Linux_Block_DNS_queries_for_specific_zone_with_IPTables.html?utm_source=linuxnewssite.com
    https://superuser.com/questions/427458/deny-all-incoming-connections-with-iptables
    https://www.digitalocean.com/community/tutorials/iptables-essentials-common-firewall-rules-and-commands
    https://askubuntu.com/questions/430069/how-to-monitor-who-is-pinging-me?rq=1

Milestone 2 feedback:

1. Make all the features as service model and not product model.
2. Change logical layer and southbound as told in the discussion.
    - Logical layer should have the logic (python) and Southbound layer should do configurations.(Ansible)
    - Logging of user configurations to be done in logical layer(Python).
3. Keepalived - Make keepalived service level discovery- That is make keepalived monitor the services as well.
4. Automation- Take the required dns configuration files from the user in the form of templates and scp it to the required DNS VM.

Milestone 2 Updated

Updated Northbound, Southbound and Logical Layer:



1. We have updated the northbound,southbound and logical layer as per the discussions.
2. Northbound layer:

   Northbound layer takes input from the user in the form of yaml files and in the form of python script triggers.
3. Logical Layer:

   We have written a python script where the logic is running.

   The python scripts based on the input given by the user in northbound calls the appropriate southbound ansible scripts for creation and southbound python scripts for deletion.

   We also did the logging of user input configurations in the logical layer by using python.
4. Southbound Layer:

   Southbound layer deals with the creation of infrastructure and deletion of infrastructure. The trigger from the southbound layer ansible scripts is done by the logical layer.

Make all the features as service model and not product model:

We have made the features as service model by requiring user input for the enabling of the services.

For firewall, the user has to specify that he wants the firewall service and our script takes care of the firewall setup as per the user input.

For HighAvailability, the user has to edit the template files (with user guide as reference) and our script takes care of copying this files to the appropriate location and enabling the service.

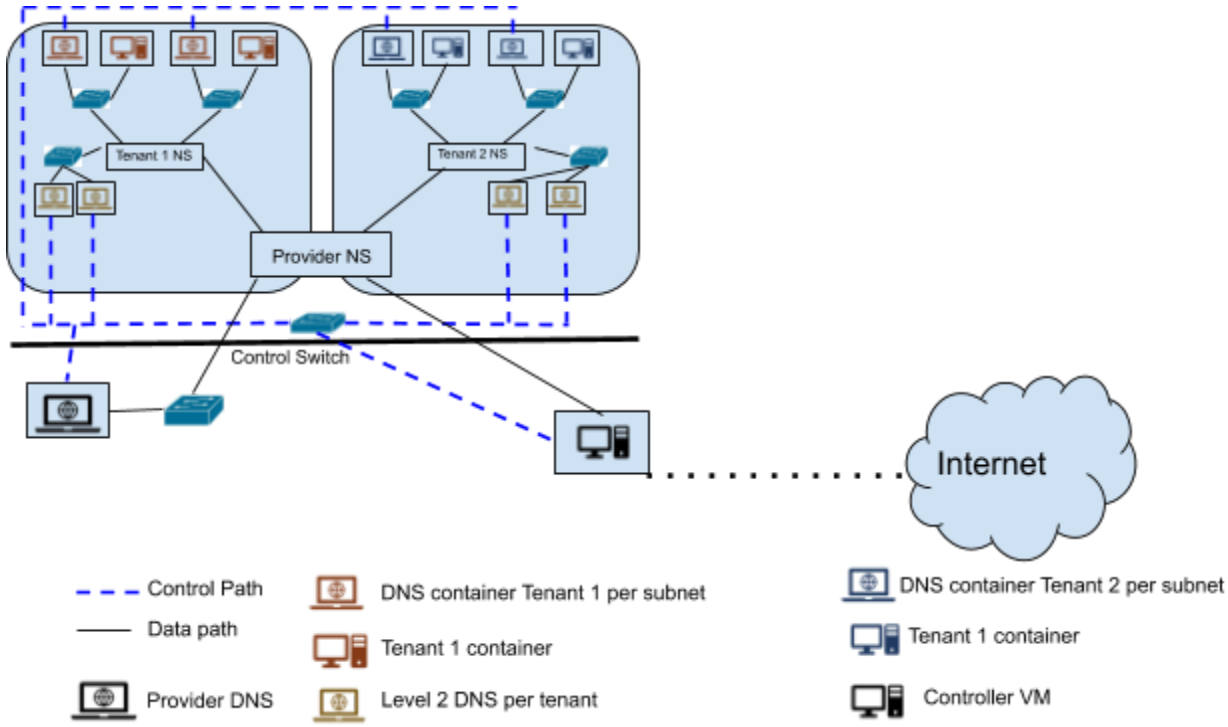
For Load Balancing and Content Based Routing, the user has to edit the template files (with user guide as reference) and our script takes care of copying this files to the appropriate location and enabling the service.

Automation:

As per our discussion, we have written python scripts DNSconf.py.
Based on the arguments given by the user, our python script would do scp and put the required template files to their appropriate locations.
The DNSconfREADME.md for DNSconf.py has been provided as well.
Our script places the user filled template files at appropriate locations(that is which level of DNS VM's) and enables the DNS services.

Milestone 3

Presented Architecture for Containerized VPC:



Containerized description of components:

1. **Provider Space:**
    a. **Provider Namespace:**
    The purpose of provider namespace is to connect the virtual private cloud to the internet. All the traffic from the internet to the VPC and vice versa goes through the provider namespace. This is done by running masquerade rules on provider namespace. It also prevents communication across multiple tenants.
    b. **Provider DNS:**
    The purpose of provider DNS instance is to forward DNS traffic to the Layer 2 DNS servers and get responses which need to be returned to the clients. It acts as the client-facing public DNS server. All the DNS queries to the VPC go through the provider DNS. This is the top level domain for the hierarchical DNS architecture provided by this service. This container is created in the provider space before any further configuration is done.
    c. **Controller Switch:**
    This is a L3 controller switch which is connected to all the DNS container instances. The purpose of this switch is to provide a control path between the DNS servers. A container bridge is being used for this purpose.
    d. **Controller VM:**

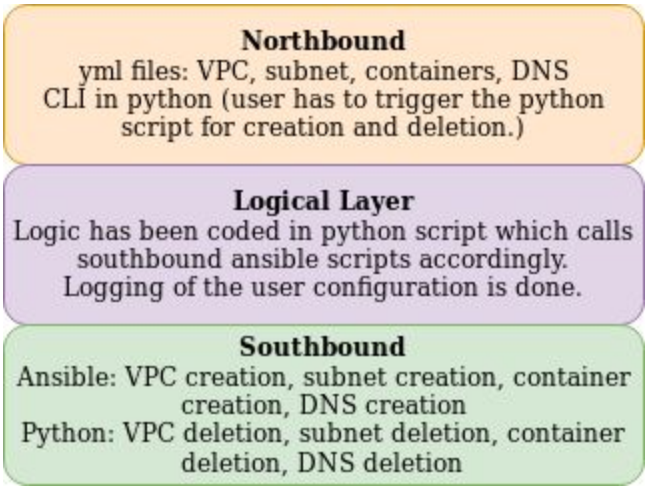
The controller VM is the access VM for provider to run all the possible scripts to create an infrastructure as requested by a client. This acts as the provisioning server and uses controller switch to perform all the provisioning operations.

2. Virtual Private Cloud Space:

a. **Tenant Namespace:**
   There is a namespace created for each tenant in the virtual private cloud. Every tenant will be provisioned a different namespace to attain L2 and L3 isolation.
b. **Tenant Bridges:** There are separate bridges connected to each namespace for different subnets per tenant. Containers in one subnet can access Containers in another subnet using this L2 bridge through the namespace. This is to create L2 isolation in each subnet of a tenant.
c. **Per subnet tenant instances:** These are the container instances which will host various applications to be used by a tenant. These instances are connected to tenant namespaces through subnet specific tenant bridges. The connection between tenant instances and bridges is made through veth pairs.
d. **Per subnet tenant specific DNS instances:** These are the tenant's DNS servers which will act as authoritative name servers for tenant instances/applications. The final response for all the queries will come from these servers if there is no entry present in Level 2 DNS servers.
e. **Level 2 DNS servers:**
   These DNS servers are container instances built to support load balancing and high availability for DNS queries. These DNS servers have DNS forwarding implemented to perform load balancing. They are configured to respond with different authoritative name server responses when multiple queries come for the same application.
   i. **Primary DNS server:** Both the servers together create a high availability setup such that the secondary server takes over only when primary goes down.
   ii. **Secondary DNS server:** This server stays in slave mode till the time primary server is active and responding to queries.

Containerized Northbound , Southbound and Logical Layer explained with architecture:

1. **Northbound:**
   The DNS (provider DNS) at the hypervisor is the top level domain. All the north south traffic is resolved by this DNS. This server maintains the records of each of VPCs in the hypervisor. Northbound layer takes input from the user in the form of yaml files and in the form of python script triggers.



2. Each tenant is allocated a unique namespace to maintain isolation among multiple tenants. Each of these namespaces has multiple subnets each served by a veth pair interface executing dnsmasq on one end, and a switch on the other. Because of this, any VMs created in this subnet are automatically assigned IPs.
3. Each namespace also has 2 DNS servers running in it at the namespace level(Level2). Both of them execute the keepalived to ensure high availability in case of DNS server failures (Level2). Every subnet is also allocated a DNS server, at the subnet level(Level3).
4. This Level2 DNS nameserver is responsible for maintaining the records for every authoritative Level3 DNS server and points to them.
5. All the DNS servers are connected to a controller switch and the controller VM(hypervisor). Controller VM is responsible for creating new infrastructure for the tenants.
6. All queries for the north-south traffic end up at the TLD, Level1 DNS server. The DNS server looks up its forward zone records. If a zone record exists and can be resolved locally it returns the A record for the particular domain.

2. Logical Layer:

1. We have written a python script where the logic is running. Users are asked to provide inputs through the Northbound layer. In this layer, the python script calls the appropriate southbound ansible scripts for creation and deletion as required based on the user inputs. All the logging is also being done in this layer. We log the user input configurations by using python.
2. The Level2 server looks up its records and if it can be resolved locally, returns the A record for the particular query, else returns a “record not found” message to the Level1 server. If the authoritative nameserver pointer exists, it forwards the query to the particular Level3 which is responsible for that zone(content-based routing).

3. The Level3 DNS server responds to the query with the IP address of the server responsible for that domain (round robin or static).

3. Southbound:

Southbound layer deals with the creation of infrastructure and deletion of infrastructure. The trigger for the southbound layer ansible scripts is done by the logical layer.

Steps to set-up DNS:

Following are the steps for configuring tenant DNS instances. All the steps mentioned below are automated:

1. We have provided dns dockerfiles from which our dns instances are created.
2. Run the script sudo python main.py with argument as dns and our scripts takes care of setting up the infrastructure.
3. Once the dns instances have been setup, the tenant has to run a script DNSconfig.py with his tenant id as the argument. This script provides the tenant with all the instance names and their IP addresses.
4. Based on the IP addresses, displayed to the user, the user has to fill the template files forward.db and named.conf as shown below.
5. Configure the DNS server by editing the /etc/named.conf file
   vim /etc/named.conf
   a. Add IP of the DNS container in the options as follows:
      listen-on port 53 { 127.0.0.1; <IP of DNS-container>; }
   b. Add IPs to be allowed to query the DNS in the allow-query as follows:
      allow-query{ any; }; // where any specifies that any IP can query the DNS.
   c. Define a zone by editing the /etc/named.conf file
      An example zone has been given below.

```
zone "." IN {
        type hint;
        file "named.ca";
        };
```

   d. Create zone files for each zone in /var/named/ directory as follows:

```
$TTL 1D
@       IN SOA  @ rname.invalid. (
                                        0       ; serial
                                        1D      ; refresh
                                        1H      ; retry
                                        1W      ; expire
                                        3H )    ; minimum
@           NS     <zone-ns>
@            A      <ip-of-zone-ns>
tenant       A      <ip of tenant DNS>
```

5. Then the user has to trigger our automation script DNSconfig.py with the argument as container name, our script takes the files the user has filled and places them in the respective paths in the container specified by the tenant.
6. This script also takes care of starting the named service and enabling ports in the firewall.
7. The tenant has to follow the above steps for all the configurations of dns instances.

Functional features implementation:

1. **Content Routing**

   The service provides content routing feature such that a client can reach an application which is running using the DNS queries. For an example: a query to project.ece792.CSC.NCSU.com would point to ece792 container using hierarchical DNS query responses. Provider DNS (NCSU.com) would provide the details of Level 2 DNS (CSC) server by looking up its entries, Level 2 DNS would then respond with the details of authoritative name server which is L3 DNS server for a specific subnet (ece792) by looking up its entries. L3 DNS server would then provide the response of an application IP (project) by looking up its entries in the zones created. The client would then be able to access the tenant container through the data path provided by the cloud infrastructure. The data path would traverse through provider namespace, to tenant space to tenant bridge to tenant container.

   The user has to provide details in the template files for named.conf and forward.db (with user guide as reference) and our python script takes care of copying and configuring them at their appropriate locations and enabling the service.

```
[root@70364679d400 /]# dig L3con.L3.L2dns.ece792

; <<>> DiG 9.11.4-P2-RedHat-9.11.4-9.P2.el7 <<>> L3con.L3.L2dns.ece792
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41675
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;L3con.L3.L2dns.ece792.         IN      A

;; ANSWER SECTION:
L3con.L3.L2dns.ece792.  79198   IN      A       20.20.20.20
L3con.L3.L2dns.ece792.  79198   IN      A       20.20.20.22
L3con.L3.L2dns.ece792.  79198   IN      A       20.20.20.21

;; AUTHORITY SECTION:
L3con.L3.L2dns.ece792.  75894   IN      NS      L3.L2dns.ece792.L2dns.ece792.

;; Query time: 1 msec
;; SERVER: 172.17.0.2#53(172.17.0.2)
;; WHEN: Sat Dec 07 00:58:21 UTC 2019
;; MSG SIZE  rcvd: 128
```

2. **Load Balancing**

   Since DNS is the most important and the basic part of the network, we will have many DNS requests coming to the DNS servers at Level 3 DNS servers. So to manage that and make sure the performance of the network does not go down, we have used the DNS round robin load-balancer to load balance the traffic. Round Robin DNS is a technique in which load balancing is performed by a DNS server instead of a strictly dedicated machine. A DNS record has more than one name-IP address pair. When a request is made to the DNS server which serves this record, the answer it gives alternates for each request. The same can be verified in the below mentioned screenshot:

```
[root@70364679d400 /]# dig L3con.L3.L2dns.ece792

; <<>> DiG 9.11.4-P2-RedHat-9.11.4-9.P2.el7 <<>> L3con.L3.L2dns.ece792
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7941
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;L3con.L3.L2dns.ece792.         IN      A

;; ANSWER SECTION:
L3con.L3.L2dns.ece792.  85337   IN      A       20.20.20.21
L3con.L3.L2dns.ece792.  85337   IN      A       20.20.20.20
L3con.L3.L2dns.ece792.  85337   IN      A       20.20.20.22

;; AUTHORITY SECTION:
L3con.L3.L2dns.ece792.  82033   IN      NS      L3.L2dns.ece792.L2dns.ece792.

;; Query time: 0 msec
;; SERVER: 172.17.0.2#53(172.17.0.2)
;; WHEN: Fri Dec 06 23:16:02 UTC 2019
;; MSG SIZE  rcvd: 128

[root@70364679d400 /]# dig L3con.L3.L2dns.ece792

; <<>> DiG 9.11.4-P2-RedHat-9.11.4-9.P2.el7 <<>> L3con.L3.L2dns.ece792
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17340
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;L3con.L3.L2dns.ece792.         IN      A

;; ANSWER SECTION:
L3con.L3.L2dns.ece792.  84967   IN      A       20.20.20.20
L3con.L3.L2dns.ece792.  84967   IN      A       20.20.20.21
L3con.L3.L2dns.ece792.  84967   IN      A       20.20.20.22

;; AUTHORITY SECTION:
L3con.L3.L2dns.ece792.  81663   IN      NS      L3.L2dns.ece792.L2dns.ece792.

;; Query time: 1 msec
;; SERVER: 172.17.0.2#53(172.17.0.2)
;; WHEN: Fri Dec 06 23:22:12 UTC 2019
```

On the Level 2 DNS server, there is load balancing based on DNS forwarders also. When a request reaches the Level 2 DNS server, it is forwarded to multiple DNS servers. The sequence in which it is sent to each forwarder which is the authoritative name server is randomized for each request. Thus, the DNS server which responds back with the query first is responded back to the client.

The user has to edit the template files (with user guide as reference) and provide appropriate IPs and our python script takes care of copying the files to the appropriate locations and enabling the service.

Management features implementation:

1. **Security:**

   One of the features we provide as a service is security. Our automation script DNSconfig.py which the tenant runs asks the tenant whether he/she wants to enable firewall on his/her DNS devices.
   Once the tenant gives his input, based on his input our automation script takes care of configuration of the firewall rules in the top level DNS server.
   The script adds the following rules we have described below:

   We have added the following rules to drop bogus DNS queries which have these hex strings. This is to protect DNS from analysing any sort of random queries and thus saving DNS cycles. Queries like 6Gdb1QlP.f.proxypipe.net., mhl00ULG.e.proxypipe.net., clacqxlG.f.proxypipe.net. will be dropped through these iptables rules. Such packets are often used in DDOS attacks over DNS.

   iptables -A INPUT -i eth0 -p udp --dport 53 -m string --hex-string "|09|proxypipe|03|net" --algo bm -j DROP
   iptables -A INPUT -i eth0 -p udp --dport 53 -m string --hex-string "|06|kitten|02|ru" --algo bm -j DROP
   iptables -A INPUT -i eth0 -p udp --dport 53 -m string --hex-string "|03|www|07|puppies|04|woof" --algo bm -j DROP

   The DNS server also provides security against multiple ping requests. The iptables rule implemented for this are:

   iptables -t mangle -A PREROUTING -p icmp -m hashlimit --hashlimit-name icmp --hashlimit-mode srcip --hashlimit 20/second --hashlimit-burst 1 -j ACCEPT
   iptables -t mangle -A PREROUTING -p icmp -j DROP

   All the DNS servers are also performing logging to various queries being made to the DNS server. This has been implemented in the bind daemon itself. This is to make sure that auditing of queries made to the DNS server can be done.

   These are iptables rules added to protect DNS server from any other traffic by allowing only UDP traffic over port 53.

2. **High availability:**

The VRRP or Virtual Router Redundancy Protocol helps you create a reliable network by using multiple routers in an active/passive configuration. If the primary router fails, the backup router takes over almost seamlessly. A similar implementation has been made to provide high availability of DNS servers.

Clients connect to a virtual IP-address. It is called virtual because the IP-address is not hard-coded to a particular interface on any of the containers.The primary and the secondary servers IPs are configured by the DHClient.

We need to install and build the keepalived image and modify the keepalived.conf file. The file must be edited with the information about the primary and secondary L2 server IP, State(Master/Backup), Priority, interface to be configured on and virtual id which should be the same on both the master as well as backup.
The virtual IP has to be configured manually which acts as the gateway for the L1-DNS to reach the active server.
We start the keepalived service. Whenever there is a failure of the master server, in a case where it goes down, the backup server becomes the active one now and it is accessible via the same IP address.

Below is the configuration for master and backup in the keepalived.conf

```
/* Configured on Master */
vrrp_instance VI_1 {
    interface eth0
    state MASTER
    virtual_router_id 51
    priority 101

    authentication {
        auth_type AH
        auth_pass monkey
    }

    virtual_ipaddress {
        192.168.230.200
    }
}

/* Configured on Backup */
vrrp_instance VI_1 {
    interface eth0
    state BACKUP
```

```
    virtual_router_id 51
    priority 50

    authentication {
        auth_type AH
        auth_pass monkey
    }

    virtual_ipaddress {
        192.168.230.200
    }
}
```

Kubernetes Inspired feature implementation:

1. **Self-Healing:**

   One of the features of kubernetes is the self healing capability which it provides to both the pods and containers. Kubernetes has several key concepts that are related to container's health and auto healing. Each Kubernetes pod has a lifecycle where a pod is in the phases defined by the lifecycle. Because of this characterization, kubernetes is able to self heal both the pods and containers, if anything goes wrong.

   We have chosen this feature to implement in our project.

   We have written a script in python "selfhealing.py" which can be scheduled as a CRON job so that it keeps a track of the container instances and heals them if they go down or get killed.

   Our python script runs at the granularity provided by the tenant in the crontab.

   Our script replicates the features provided by Kubernetes in the following two ways:

   1. When a container has been running and it exits because of interruptions ,our script detects that the container has exited and it restarts the container.
   2. When a container is deleted, our script detects that the running container has been stopped and deleted, and it recreates and starts the container from the last committed image. The recreation of the containers is done by the script as our script commits the running container images. Our script also removes the images which are no longer needed(that is replica images of the same container).

Setting up provider infrastructure:

1. **Setting up Provider DNS:**

   Creation of Provider DNS instances is taken care by the automation scripts. Once the DNS instance is created, there are some basic configurations required on provider DNS which needs to be performed manually. This is to be done by the provider and does not involve users to perform this configuration.

   Editing the vi /etc/named.conf:
   Under *options* section, look for listen-on port 53 and add an entry for the IP of the provider DNS server as mentioned below and add an entry to filter AAAA queries on IPv4:

   listen-on port 53 { 127.0.0.1; <Provider DNS IP>; };
   filter-aaaa-on-v4 yes;

   Add a zone entry in the same file as:

   zone "com." IN {
   type master;
   file "/var/named/fwd.root.com.db";
   allow-update { none; };
   Allow-transfer { any; };
   };

   Editing the /var/name/fwd.t1.db (zone) file:
   A nameserver to IP mapping needs to exist in this file such that DNS knows that this query can be responded by which server. For example:

   $TTL 1D
   @ IN SOA t1.com. root.t1.com. (
   11 ; serial
   5 ; refresh
   60 ; retry
   5 ; expire
   5) ; minimum

   @ IN NS t1.com.
   @ IN A 172.17.0.2
   vm.apidns IN NS apidns.t1.com.
   apidns IN A 172.17.0.4
   IN A 172.17.0.5
   testvm1 IN A 100.0.0.35

   Here 172.17.0.4 and 172.17.0.5 are the Level 2 DNS IPs for multiple subnets.

2. **Creating Provider Namespace:**

- We have a provider namespace already created in the hypervisor before the scripts are run. The automation scripts written using ansible and python would take care of adding routes, providing IP addresses, and adding respective iptables rules in the provider namespace.
- The manual configuration done in provider namespace is to provide connectivity to the internet. This is done by adding nat rules in iptables.
- This is achieved by adding a vethpair with one end in the provider namespace and the other end in the hypervisor.
- We have added a vethpair with the IP address as 10.10.10.1/24 in the namespace and 10.10.10.10/24 in the hypervisor.
- There are iptables rules in the hypervisor which does source nat such that the provider namespace is reachable to the internet.
- The iptable rule we have used is :
  Iptable -t nat -I POSTROUTING 1 -s 10.10.10.0/24 -o ens3 -j MASQUERADE
- This rule does the necessary nat operation to make the provider namespace able to reach the internet.

3. **Creating Controller bridge and controller VM:**

- The controller bridge and controller VM need to be created manually before the scripts are run. This is a one time task which needs to be performed on the provider space. The controller bridge has an interface connected to each DNS container in our architecture. The purpose of this controller bridge is to provide a control and management path to the tenants to the DNS servers.

Deletion of VPC Infrastructure for a tenant:

The script “delete_cont.py” is responsible for clearing the configurations which has been done for a tenant. The user is asked about what is to be deleted which are containers, subnets, VPC. The script inherently takes care of deleting all the components which are related to these components and cleans up the configurations.

Future Scope:

This project can be enhanced to provide more DNS services such as geographical based routing. There will be slight changes needed to edit the named.conf file to have ACLs based on geography. We can further increase the scope to support scalability of records for DNS. Other than increasing the scope of this project to add features, the project as also be made to support other services such as Firewall as a service, Load balancing as a service, etc. The services which are similar to DNS can be easily implemented without much hassle.

Github link to implemented code:
https://github.ncsu.edu/schippa/DNS-as-a-Service-VPC

All the scripts and READ ME is also attached in the ZIP file uploaded to google drive.

The yml scripts and templates are located under PROJECT3/relevant folder in the above mentioned link.

References:

1. **Creating DNS:**
   https://www.isc.org/bind/
   https://www.unixmen.com/setting-dns-server-centos-7/
   https://www.itzgeek.com/how-tos/linux/centos-how-tos/configure-dns-bind-server-on-centos-7-rhel-7.html
   https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-centos-7

2. **High availability on DNS:**
   https://louwrentius.com/configuring-attacking-and-securing-vrrp-on-linux.html
3. **Security:**
   https://www.perturb.org/display/1186_Linux_Block_DNS_queries_for_specific_zone_with_IPTables.html?utm_source=linuxnewssite.com
   https://superuser.com/questions/427458/deny-all-incoming-connections-with-iptables
   https://www.digitalocean.com/community/tutorials/iptables-essentials-common-firewall-rules-and-commands
   https://askubuntu.com/questions/430069/how-to-monitor-who-is-pinging-me?rq=1
4. **Containerization:**
   https://medium.com/google-cloud/kubernetes-liveness-checks-4e73c631661f
   https://github.com/kubernetes/dns/blob/master/docs/specification.md
   https://github.com/CentOS/CentOS-Dockerfiles/tree/master/bind/centos7
   https://kubernetes.io/docs/concepts/services-networking/dns-pod-service