Case Study

Group: RDEnvi

Robertas Asvicas

Dmitry Lvov

**Contents**

[Project definition 3](#_Toc55925772)

[Requirements 3](#_Toc55925773)

[Infrastructure Diagram 4](#_Toc55925774)

[Infrastructure Components 5](#_Toc55925775)

[VPC 5](#_Toc55925776)

[Subnets 5](#_Toc55925777)

[NAT subnets 5](#_Toc55925778)

[Web/App subnets 5](#_Toc55925779)

[Database subnets 5](#_Toc55925780)

[Transit Gateway 6](#_Toc55925781)

[VPC Flow Logs 6](#_Toc55925782)

[Elastic Load Balancer 6](#_Toc55925783)

[Billing 6](#_Toc55925784)

[Terraform (WIP) 6](#_Toc55925785)

[Ansible 6](#_Toc55925786)

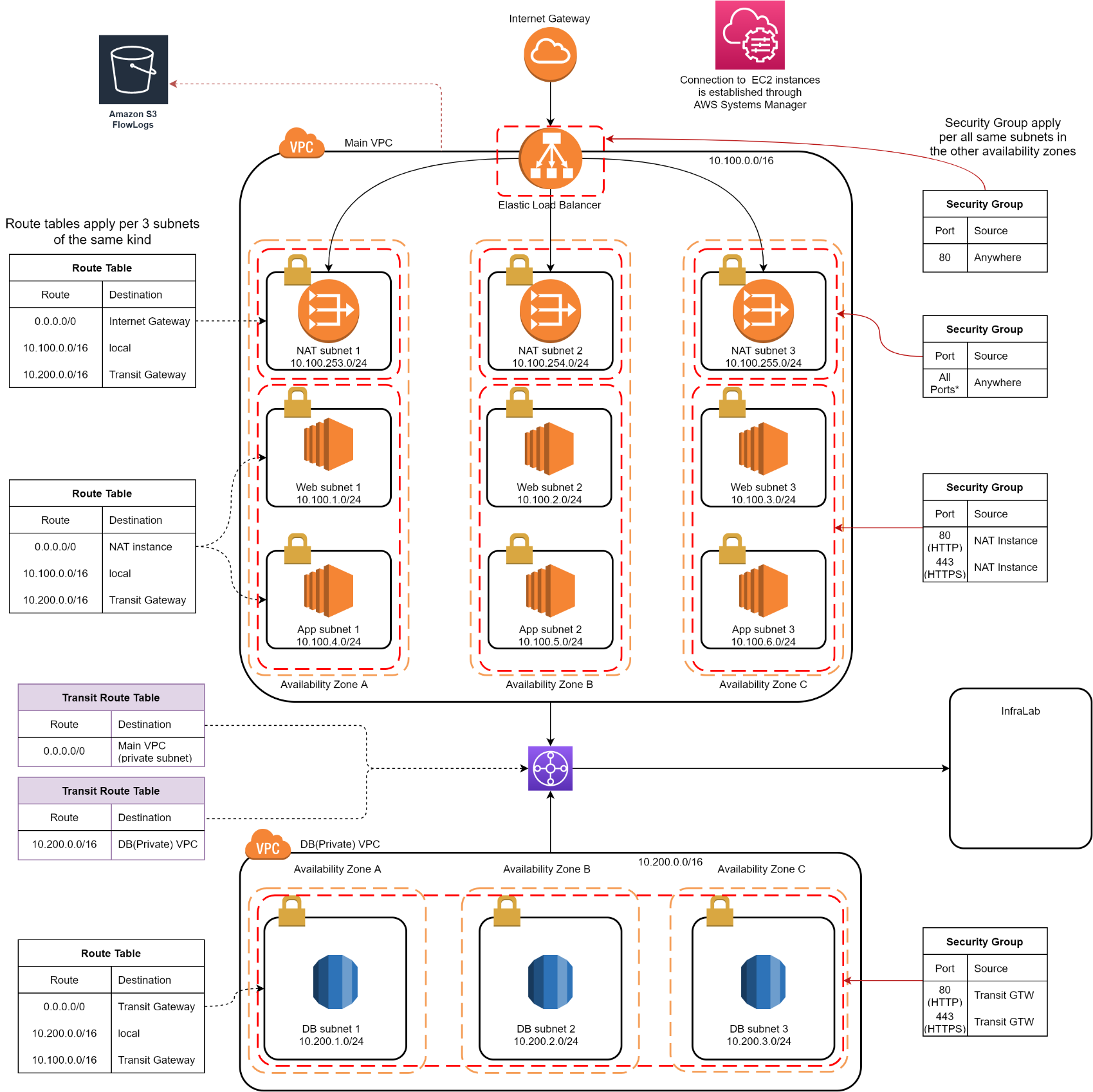
# Project definition

Our main goal in this project was to create suitable infrastructure for software students to host websites, applications. Infrastructure was supposed to be automated, scalable, and secure. We were using Amazon Web Services (further as AWS) for infrastructure creation, Ansible and Terraform for automation and different techniques for security implementations.

# Requirements

* Network Infrastructure on AWS
  + Secure – usage of private subnets, private VPCs, NAT instances, configured Security Groups
  + Scalable – usage of Autoscaling Groups, Availability Zones
  + Resilient – usage of Elastic Load balancer, Availability Zones
  + Automated – deployment is automated via Ansible, Terraform
* Isolated Database
* Automation – Ansible:
  + Apache deployment
  + Flask deployment
* Automation – Terraform

Infrastructure Diagram



Infrastructure Components

## VPC

Infrastructure is made out of 2 VPCs:

|  |  |
| --- | --- |
| Main VPC | Private VPC |
| CIDR block: 10.100.0.0/16 | CIDR block: 10.200.0.0/16 |
| Used to host everything BUT databases | Used to host ONLY databases |
| NAT subnets can be accessed from WAN | Can NOT be accessed from WAN |
| Has Internet Access | No Internet Access (only through Transit Gateway) |
| EC2 instances can be accessed only through SSM | EC2 instances can be accessed only through SSM |
| Has 3 Availability Zones | Has 3 Availability Zones |

## Subnets

There are 4 types of subnets in our infrastructure:

* NAT subnets
* Web subnets
* App subnets
* Database subnets

### NAT subnets:

Is hosting **NAT** **instances** which serve as a **middleman** between **Private** **subnet** and **Internet**. **Route Table** is configured that NAT subnet instances see Internet Gateway (further as IGW) – route to the Internet, Transit Gateway (further as TGW) – route to Database and local VPC Instances. Has its own **Security Group**, which allows all traffic Inbound into NAT subnet to all ports. **IP** addresses **range** varies **from 10.100.253.0/24 to 10.100.255.0/24**. Is deployed in all **3 Availability Zones** (further as AZ).

Web/App subnets:

Web and App subnets have **same configuration**. Are hosting **websites** (Apache, Nginx, Flask, etc.). **Route Table** is configured that it sees TGW – route to DB, NAT instance – route to IGW and local VPC instances. Has its own **Security Group**, which allows port **80** and port **443** (HTTP and HTTPS) traffic from NAT instances. **IP** addresses **range** varies **from 10.100.1.0/24 to 10.100.6.0/24.** Is deployed in all **3** **AZ**s.

### Database subnets:

Database subnets are hosting **database instances**. **Route Table** is configure that it sees TGW (route to Main VPC and IGW) and local VPC instances. Has its own **Security Group**, which allows port **80** and port **443** traffic **from** **TGW**. **IP addresses** range varies **from 10.200.1.0/24 to 10.200.3.0/24**. Is deployed in all **3 AZ**s.

Note: We are NOT using RDS on database, because of the big costs of it. For demonstration purposes we use EC2 instances.

## Transit Gateway

Transit Gateway connects **2 VPCs** and our **InfraLab** (server on-premises) through VPN connection (which is not complete yet). It provides **one-way** internet connection for Private VPC. Has its **own** Route Tables (see diagram above). Therefore, for our infrastructure TGW is configured differently with **no** default propagation and **not unified** route table.

## VPC Flow Logs

**All** traffic that is going through Main VPC is **registered** by **Flow Logs** (both, **accepted** and **rejected** traffic). Logs are stored in **S3** **bucket** called *flow-log-bucket-rdenvi*. AWS is automatically sorting everyday traffic throughout different folders.

## Elastic Load Balancer

Elastic load balancer is responsible for managing the traffic load that will be generated by the users among 3 availability zones for much faster accessibility. Has its own **Security Group** which allows port **80** so that the users can only connect through ELB without violating the privacy of the infrastructure.DNS address: **WEBLB-885405402.eu-central-1.elb.amazonaws.com**

# Billing

Whenever budget exceeds 4.00USD an alarm message is sent to the root account e-mail ([r.asvicas@student.fontys.nl](mailto:r.asvicas@student.fontys.nl)).

# Terraform (WIP)

Terraform configuration used to deploy basic VPC setup (2 VPC instances) and 3 subnets inside each VPC. Work in Progress. Will be finished for 3rd MVP.

# Ansible

There are 3 Ansible scripts (further called playbooks):

* SSH key exchange – exchanges SSH keys with servers, therefore next time Ansible may use not AWS generated key, but id\_rsa (local generated key). Key is made for security implementation.
* Apache + Flask deployment – deploys Flask with Apache on server. Deletes Apache files, therefore Flask would not conflict with it.
* Nginx deployment – Deploys Nginx on server.

# MoSCoW

**Must have:**

* Public VPC
* Private VPC
* NAT instance
* Internet Gateway
* 3 Subnets
* Apache deployment with Flask
* EC2 instances

**Should have:**

* VPC Flow logs
* Systems manager
* Elastic Load Balancer
* Transit Gateway

**Could have:**

* Nginx deployment
* SSH key exchange deployment

**Won’t have (this time):**

* Site to Site VPN connection
* Terraform configuration
* Auto scaling groups