Laboratory Practice Report

Practice 4

September 27, 2023

Computer Systems Engineering

Cloud Architecture

Prof. M.S. Rodolfo Luthe Ríos

Marco Ricardo Cordero Hernández

is727272@iteso.mx

ITESO

Universidad Jesuita
de Guadalajara

Abstract

Through this practice, high availability for applications will be explored through the same cloud provider that's been utilized in all of these demonstrations.

Several technique combinations are implemented in order for a demonstrative application high availability demonstration will be explored, which at the end of this file will provide the sufficient baseline knowledge to start implementing these contents in a real world scenario.

The material that's been walkthrough serves a greater purpose of aiming the contents of the curse towards the formation of a decent profile and that of an engineer capable of deploying fault tolerance developments, something useful in any kind of project and for any kind of client.

State of the Art

In the current state of the world, several professions develop their duties day to day within certain lapses of time. For example, medics and other health related workers would often conduct surgeries and such operations. But, how would they achieve it? Besides the vast knowledge found behind their work means, acquired through a long range of studies, they'd often find themselves using medical/surgical kits, compound of gauzes, scalpels, scissors, forceps, etc. This collection of items allows them to perform a variety of health related tasks, often one independent from the other.

How such a strange yet obvious description fits in a cloud class report? The thing is, the mere concept of a kit containing tools to help achieving a greater purpose it's applicable to most of developments out there, because at the end, a medical operation it's a short to medium lived project or development by itself. Information technology field and applied computer science engineering field isn't exempt from the use of kits; the twist is that these collections are often called "stacks", making reference to the set of technologies used to develop an application, including programming languages, frameworks, databases, and so on [1].

Nowadays, the course that modern tools are taking it's that of pure web development and some sort of combination with mobile applications (some also having a web-first approach), rendering desktop applications almost outdated when it comes to new software, and opting for progressive web apps and other types of wrappers. Leaving aside the benefits and disadvantages of this fact, web developing has grown so much since it first appeared that there as many approaches of creating a web application as there are medical specializations. Evidently, tech stacks make their appearance here, with popular stacks such as MEAN (MongoDB, Express.js, AngularJS, Node.js), MERN (same as previous but React swapped with Angular), or one of the first stacks intended for the web, *LAMP* (Linux, Apache, MySQL, PHP) [2]. The thing in common of these stacks is that they provide developers with a database, a server, a platform and a main developing programming language to interface with the previous components. The current practice will use the LAMP stack for a quick demonstration of other cloud features.

Along with web developing, computer usage and internet access became common things in the average household, leading to network traffic never seen before. In the 1990's, special hardware

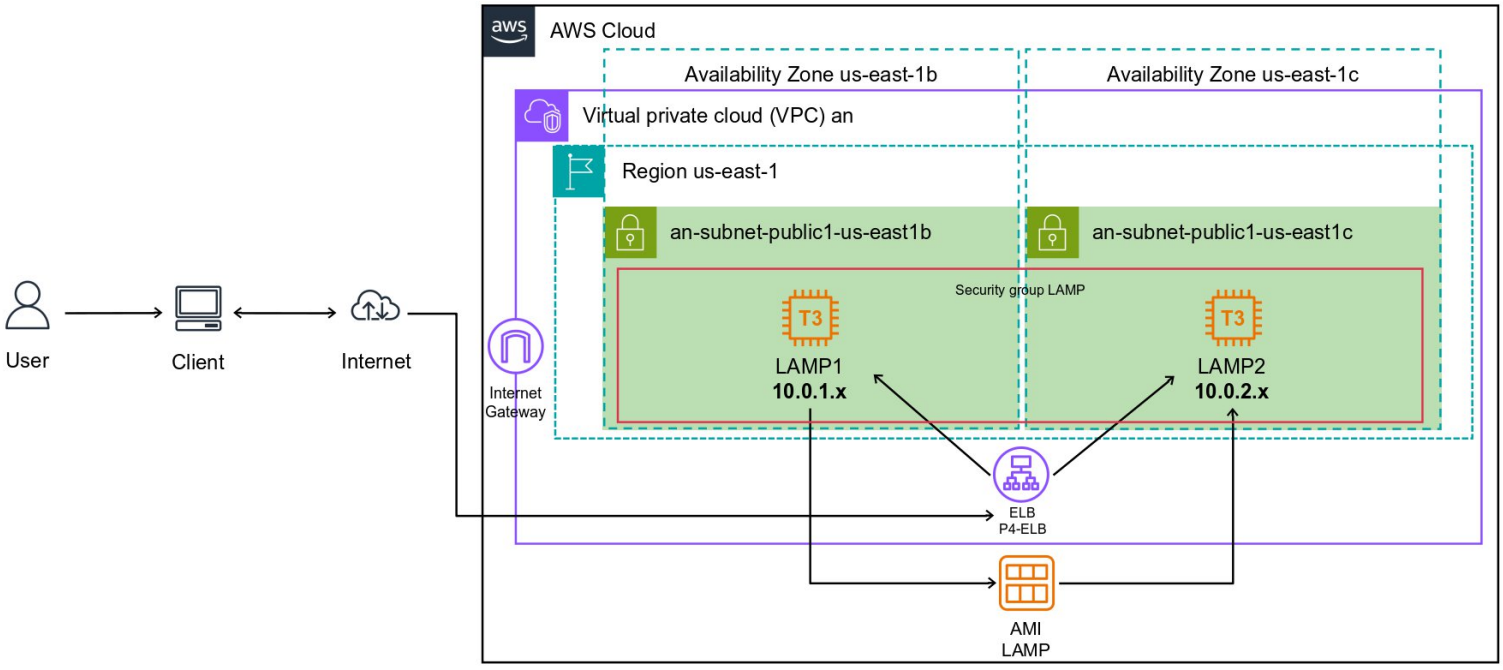
known as “application delivery controller” was deployed to distribute said traffic across a network [3]. The concept evolved to “load balancers”, with the action of “load balancing” being the action of efficiently distributing incoming network traffic across a group of backend servers, also known as a “server farm” or “server pool” [4].

A load balancer can be achieved through specialized hardware, like the most primitive ones, or through software, like the newest. The basic idea of this architecture components is taking at least two different computers and distributing incoming traffic between them using low level algorithms such as round robin (sequential distribution), least connection (new traffic goes to component with less current connections), or even random distribution.

In this practice, AWS Elastic Load Balancing [5] will be used to redirect traffic along two EC2 [6] instances with a simple LAMP architecture inside each one. The load balancer to implement is an special one, an “application” load balancer. The difference with other kinds of balancers is that the one that’s being talked about functions at the seventh (application) layer of the OSI model, meaning that some rules can be applied specifically for web applications.

Diagram

The following architecture it’s proposed as a graphic solution for the stated goals.



Practice Development

Previously, a custom VPC was created to demonstrate networking inside AWS Cloud. Now, starting from this premise, the first thing to do in this practice will be the creation of a new *public* subnet with the following features:

- Name: public3
- Availability zone: us-east-1c (not previously used)
- CIDR block: 10.0.13.0/24

- Redirect 0.0.0.0/0 to internet gateway

What's been done it's merely the first step of this practice development, but a critical one inside the requirements.

LAMP Instance

To deploy the previously mentioned technologic suite, a new EC2 instance has to be create with the following features:

- Name: LAMP1
- Operating system: Amazon Linux 2023 or newest version
- Instance type: t3.small
- Associate it with the first public subnet
- Auto-assign public IP
- New *LAMP* security group
    - Accept port 22 (SSH), port 80 (HTTP) and port 443 (HTTPS) connections from the internet (0.0.0.0/0)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
LAMP1	i-09f40f1057bc80089	Running	t3.small	2/2 checks passed	No alarms	us-east-1b

In the current state of the instance not much can be done besides performing basic Linux tasks. To address this matter, the LAMP stack will be installed; this has to be made within the instance, so (ideally) an SSH connection has to be established to run the following commands.

```
sudo dnf update -y

sudo dnf install -y httpd wget php-fpm php-mysqli php-json php php-devel

sudo dnf install mariadb105-server1
```

```
[marcordero@fedora Keys]$ chmod 400 labsuser.pem
[marcordero@fedora Keys]$ ssh -i labsuser.pem ec2-user@ec2-3-80-153-234.compute-1.amazonaws.com
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Sat Oct 21 00:16:39 2023 from 189.203.103.26
[ec2-user@ip-10-0-1-164 ~]$ sudo dnf update -y
sudo dnf install -y httpd wget php-fpm php-mysqli php-json php php-devel
sudo dnf install mariadb105-server
```

Once all dependencies are successfully installed, these other commands will be executed to start the required services for the LAMP server.

```
sudo systemctl start httpd # Start Apache server

sudo systemctl enable httpd # Ensure server initialization each system boot

sudo systemctl is-enabled httpd # Verify service status
```

```
[ec2-user@ip-10-0-1-164 ~]$ sudo systemctl start httpd # Start Apache server
sudo systemctl enable httpd # Ensure server initialization each system boot
sudo systemctl is-enabled httpd # Verify service status
enabled
```

For this kind of stack, the usual file container route is defined as */var/www/html*. This will contain the entry point for static files to be displayed in a common web browser. Before this validation, the default files will be modified to show a single line text “LAMP1”. Vim text editor will be used for this.

```
sudo rm /var/www/html/*

sudo vim /var/www/html/index.html

sudo cat /var/www/html/index.html

sudo systemctl restart httpd.service

systemctl status httpd.service
```

```
[ec2-user@ip-10-0-1-164 ~]$ sudo rm /var/www/html/*
sudo vim /var/www/html/index.html
sudo cat /var/www/html/index.html
sudo systemctl restart httpd.service
systemctl status httpd.service
<!DOCTYPE html>
<html lang="en">
        <head>
                <title>Practice 4</title>
        </head>

        <body>
                <h1>LAMP1</h1>
        </body>
</html>
● httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
    Drop-In: /usr/lib/systemd/system/httpd.service.d
             └─php-fpm.conf
     Active: active (running) since Sat 2023-10-21 00:18:40 UTC; 55ms ago
```

At this point, the instance view can be seen from a web browser through the AWS provided public DNS.

Before load balancing demonstration, another AWS feature will be used: instance image creation. Inside EC2 console panel, through the path of LAMP1 instance > options > Image and templates > Create image, an own-defined image can be created for further instance creation and deployment.

- Name: LAMP
- No reboot disabled
- Default volume size
- Delete on termination enabled

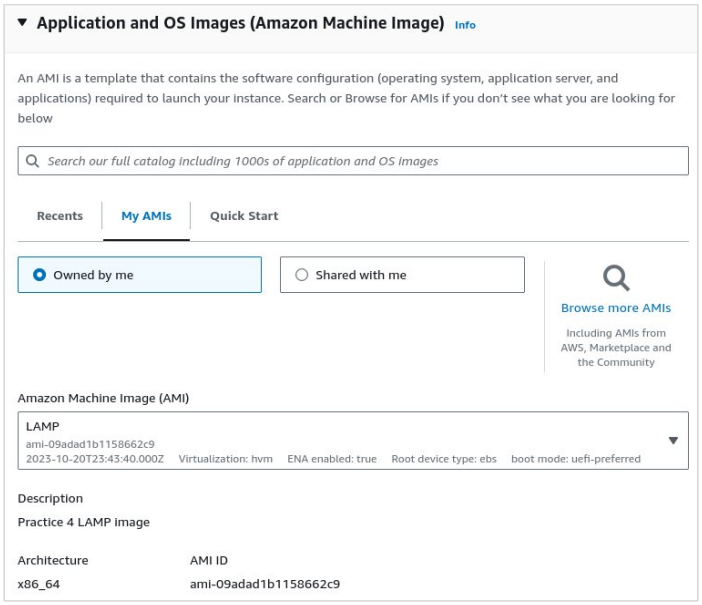
Name	AMI ID	AMI name	Source	Owner	Visibility	Status
P4 AMI	ami-09adad1b1158662c9	LAMP	042979533702/LAMP	042979533702	Private	Available

This AMI will come in handy in the next steps.

Second LAMP Instance

Right after the previous AMI creation, a second LAMP instance has to be created using the same template with the following modifications:

- Name: LAMP2
- Associate it with the newest subnet (first step of this development)
- Auto-assign public IP
- *LAMP* security group



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
LAMP1	i-09f40f1057bc80089	Running	t3.small	2/2 checks passed	No alarms	us-east-1b
LAMP2	i-0f56b589df27791f5	Running	t3.small	2/2 checks passed	No alarms	us-east-1c

Equally as the previous LAMP instance, the stack has to be installed within the console and the entry index page has to display “LAMP2” as the default single line text.

Load balancer

Finally, the load balancer will be created through the same EC2 AWS panel in its own section. In this case, LAMP stack it’s aimed for web applications, therefore, an *app load balancer* it’s needed. This can be done in two steps: target group creation and load balancer creation.

Target group:

- Name: P4-TG

- Target type: instances
- HTTP at port 80
- IPv4
- AN VPC (Previously created) with HTTP1
- Default health checks settings
- Register both LAMP instances (then include as pending below)

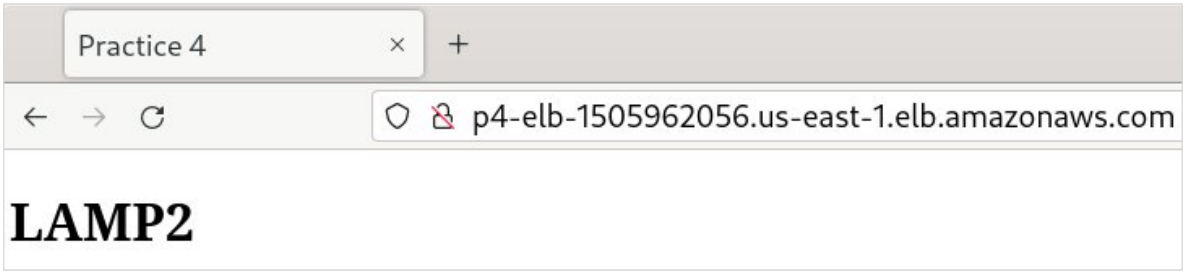
Name	ARN	Port	Protocol	Target type	Load balancer	VPC ID
P4-TG	arn:aws:elasticloadbalanci...	80	HTTP	Instance	None associated	vpc-0bbb6842b66d5c959

Load balancer:

- Application load balancer
- Name: P4-ELB
- Default scheme and IP settings
- Network mapping
  - AN VPC
  - us-east-1b and us-east-1c mappings (ensure both are public subnets)
- Default and LAMP security groups
- Default listeners and routing settings with bounded target group

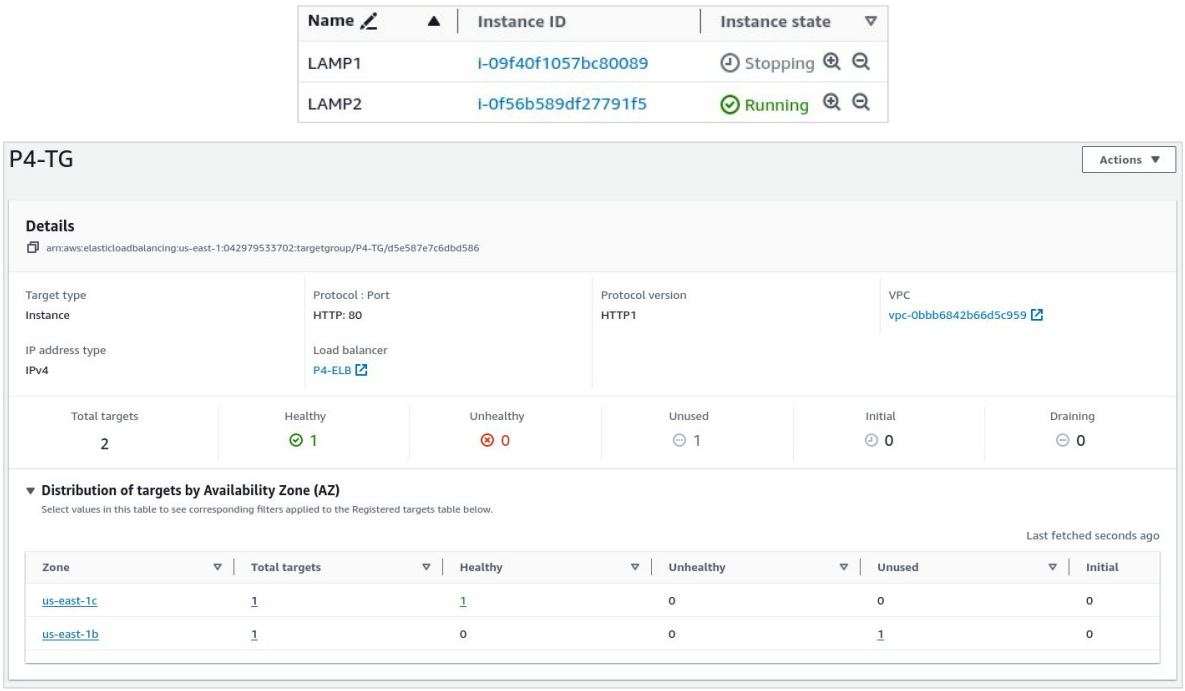
Name	DNS name	State	VPC ID	Availability Zones	Type
P4-ELB	P4-ELB-1505962056.us-east-1.elb.amazonaws.com	Active	vpc-0bbb6842b66d5c959	2 Availability Zones	application

Once created, the public DNS name provided will be accessed through a web browser to test the load balancer. In the first access, this is displayed.



In the second access, this is displayed.

This behavior demonstrate a correct load balancer function. Now, the instance that responds the second access will be stopped and the load balancer targets should indicate this action.



If the previously used DNS name is accessed again, the second LAMP server will *always* be displayed, at least until the first instance is restarted.

Problems and Solutions

The current development did not had any problems, and therefore, no solutions were proposed to address these matters.

Experiments and Results

Do instances need to be public?

With a load balancer, the ultimate goal for an application distributed across several computers or servers per se it's so access its contents through a single endpoint or domain, that'd be the address bounded to the balancer. In this practice development, both instances created could be publicly accessible individually, defeating the main purpose of the load balancer in this context.

In most real case scenarios, the instances would be behind a private subnet *or* a public subnet that would only accept HTTP/S traffic from the load balancer. This would prevent cases in which a possible IP leak takes place from the application architecture; malicious intents of high concurrence traffic then would be aimed for the leaked instance that'd be still bounded to the balancer, but, as the instance it's already at full capacity, the traffic would be redirected to the remaining instance and will probably also be congested by normal traffic.

Although is not implemented the steps to avoid this would be the following:

1. Create a VPC with at least *two* public subnets

2. Create a security group for the load balancer, accepting traffic from anywhere to HTTP/S ports and all outbound traffic
3. Create a security group for the instances, accepting HTTP/S traffic from the load balancer security group

After this configurations are made, the EC2 instances shouldn't be accessible publicly from outside of the VPC. SSH inbound traffic could still be accepted for administrative purposes, however, this should also be configured accordingly, as it could also pose as a potential security risk.

Could load balancing work with a single availability zone?

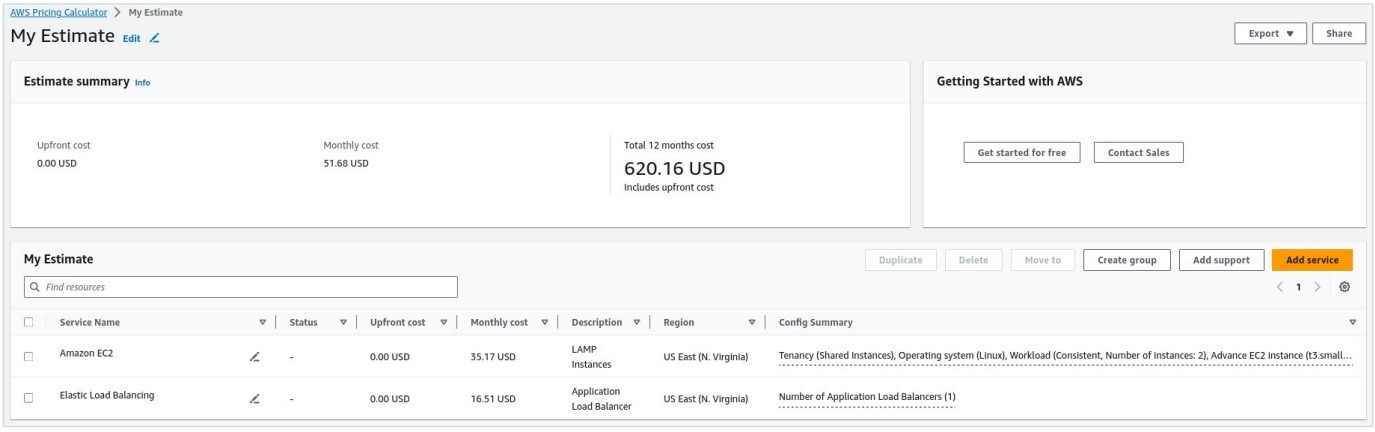
Short answer: yes. Is it recommended? Most definitely not. On premise infrastructure that implements load balancing technologies are bounded to a single geographical space, meaning that if a computing instance or server architecture it's compromised, probably the remaining ones bounded to the balancer will also be in the same situation, rendering useless the whole application. As AWS, most cloud providers offers high availability if possible distributable resources are located along several availability zones, not only providing applications with an additional degree of resiliency, but making load balancing more redundant and prone to implement.

Budget Justification

Having:

- Two LAMP instances
- One application load balancer with 20 new connections per minute

The budget for the solution would be the following:



Conclusions

Load balancing it's such a powerful tool that can take days of architecture planning to just a few minutes. The relative ease in which these components can be created inside AWS it's astonishing and can surely be implemented in any kind of future development.

It's such a shame that concepts like those reviewed in this practice are seen at a later stage of the current educational path, as this kind of technology will be really interesting to analyze in early stages of this professional track. However, it can be understood how this revisions are only seen at the mere end of the career, as most of these concepts couldn't be well comprehended before knowing the bare basics behind seen functionality.

Bibliography

[1] Heap. 'What is a Tech Stack: Examples, Components, and Diagrams'. [Online]. Available: https://www.heap.io/topics/what-is-a-tech-stack.

[2] A. Mendes. 'Top 10 tech stacks for software development in 2024'. [Online]. Available: https://www.imaginarycloud.com/blog/tech-stack-software-development/.

[3] AppViewX. 'Load Balancer and Types'. [Online]. Available: https://www.appviewx.com/education-center/load-balancer-and-types/.

[4] Nginx. 'What Is Load Balancing?'. [Online]. Available: https://www.nginx.com/resources/glossary/load-balancing/.

[5] Aws.amazon.com. 'Elastic Load Balancing'. [Online]. Available: https://aws.amazon.com/es/elasticloadbalancing/.

[6] Aws.amazon.com, 'Amazon EC2'. [Online]. Available: https://aws.amazon.com/es/ec2/.