Laboratory Practice Report

Practice 7

October 25, 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

The purpose of the development to be shown it's to demonstrate how several lengthy cloud implementations can be done in matter of seconds with the appropriate setup and through automation capabilities.

Computing, being physical or remote, has come a long way and has advanced so much to the point that "new" computers can be created with just a few click inside a really convenient graphical interface. This seamless features provided with at least one cloud vendor enhances developers production time for fast paced environments where things need to be done correctly and as soon as possible.

This particular practices also shows the development of natural soft-skills: problem solving abilities and resiliency, this because multiple high-level problems are encountered when deploying several technologies at once in so reduced timeframes.

State of the Art

When talking about cloud usage in modern software development, multiple terms come into play such as *services*. This components inside a remote architecture can also be cataloged as *IaaS* (Infrastructure as a Service), *PaaS* (Platform as a Service), and *SaaS* (Software as a Service) [1], at least for the most common denominations. Cloud discovery learning path would often start by pointing out the differences between this concepts, along with their benefits and disadvantages. This current document, being the seventh part of a series of practical implementations will overlook this step and assume that the reader already knows what's been described; however, *PaaS* will remain relevant for the rest of its contents.

Platform as a Service removes the need of physical components inside customers immediacies and provides them with remote cloud resources in which applications can be executed, hosted and analyzed [1]. With this approach, developers can focus entirely in business logic implementations through code and for multiple paradigms such as web development, the main and most popular line of modern development [2].

Having this type of service available at any times can be as damaging as it can be beneficial, mainly when monetary charges are involved. These services will often provision an environment with all sorts of resources needed for complete deployment, however, if the architect or whoever it's making the implementation is not sufficiently careful, unexpected exponential charges will be found at the end of the billing period.

With the knowledge and recognition of the dangers found in all of these this architecture facilitators, the main advantage of PaaS can be taken when dealing with multiple scenarios such as testing and production environments. When referring to this specific part of software development life cycle, a very powerful approach can be found used: blue green deployment.

Blue green deployment makes reference to the application release model that gradually transfers user traffic from a previous version of an app to a nearly identical new release [3]. This approach evaluates two infrastructure nodes, green and blue nodes. Green node will often contain production

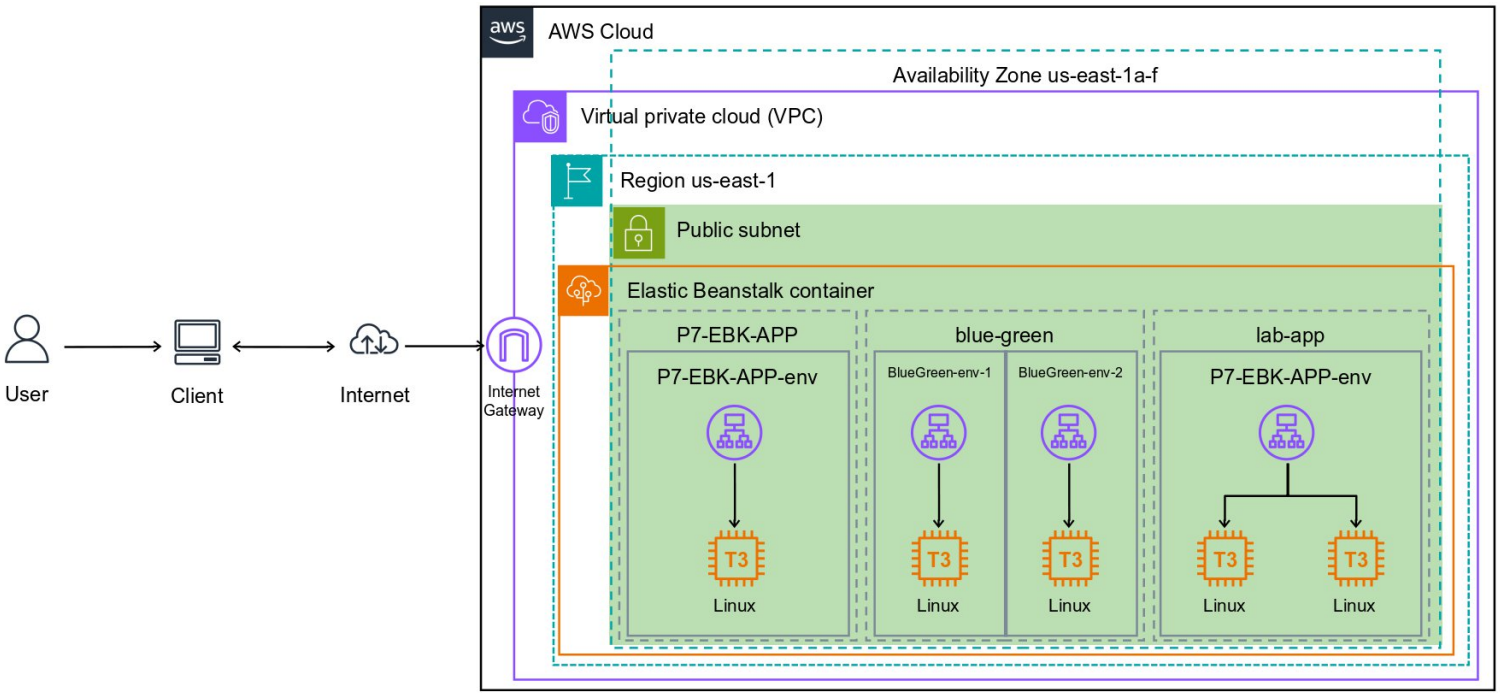
quality code and its contents can be accessible to common public or desired target users. Meanwhile, blue node would be the old version of the same code that's been replaced by the green node contents. In a CI/CD production environment, green node will eventually become blue node, providing the users with a new green node (usually blue node with updates) containing upgraded contents of the same application.

One of the main advantages of this approach it's live time code replacement even at peak application utilization hours, meaning that there is no need for implementers to wait until low access moments of the day (often midnight) to roll in the updates. To achieve this, load balancing and domain swapping techniques are involved in order for users to keep using the application normally and through the usual access domain/endpoint.

This practice not only will explore a blue green implementation, but will also demonstrate the usage of a PaaS service.

Diagram

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



Practice Development

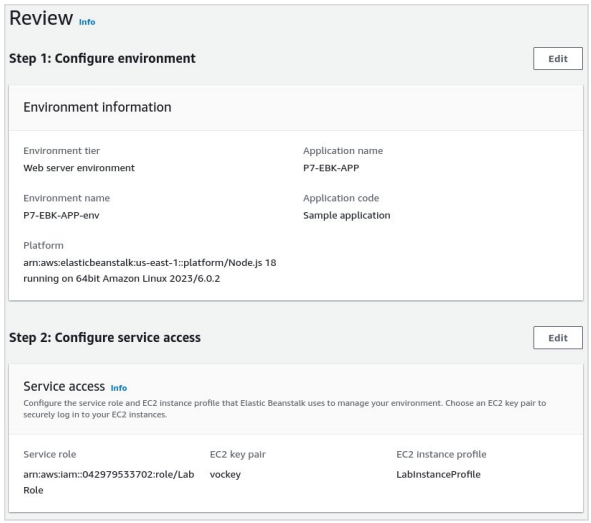
Elastic Beanstalk

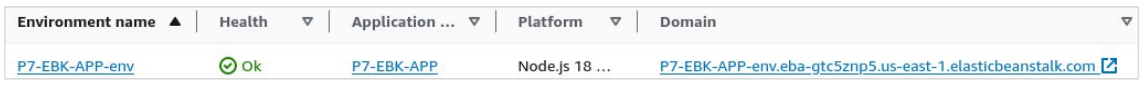
The entry point of this development will be through the use of AWS Elastic Beanstalk service [4], a end-to-end web application management tool that allows developers focusing on code rather than infrastructure. As well as other Amazon Web Services, this service also has its own section inside the console.

Inside dedicated Elastic Beanstalk section, a new application has to be created with the following features:

- Environment tier: Web server
- Name: P7-EBK-APP
- Environment: default values
- Platform: Node.js (leave default values for the rest)
- Application code: Sample application
- Presets: High availability
- Service access
    - Service role: existing LabRole
    - Key pair: vockey
    - Instance profile: LabInstanceProfile

Remaining configurations can be leaved as default. After the previous tweaks are made, app creation can be confirmed (takes at least 5 minutes).

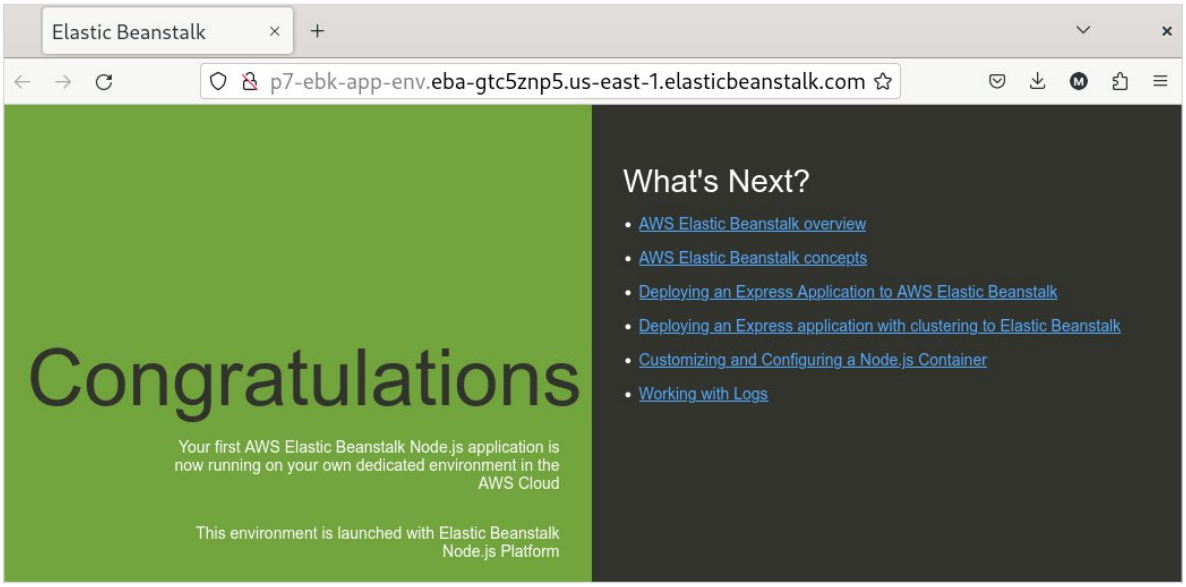




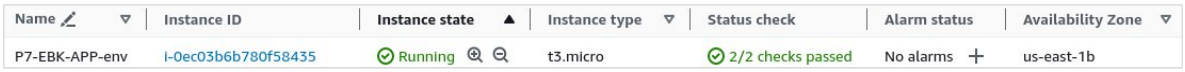
Note: high availability preset has been selected to ensure the web application's fault tolerance in order for it to be always accessible.

Note: *LabRole* has been selected because of the (academic) profile used to develop this practice.

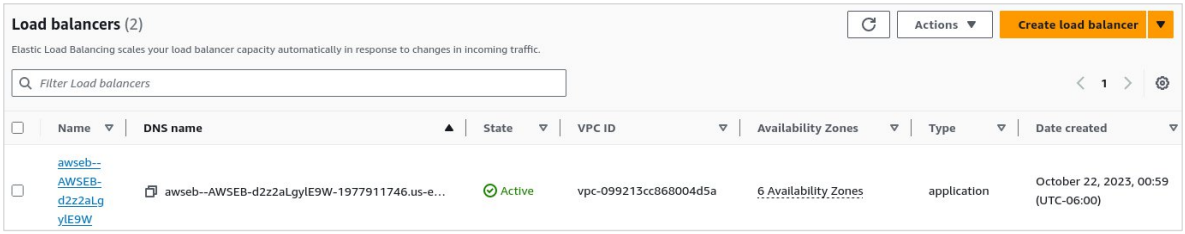
After the application environment has been successfully created, the endpoint provided can be tested. It should return a simple placeholder view of the application's entry.



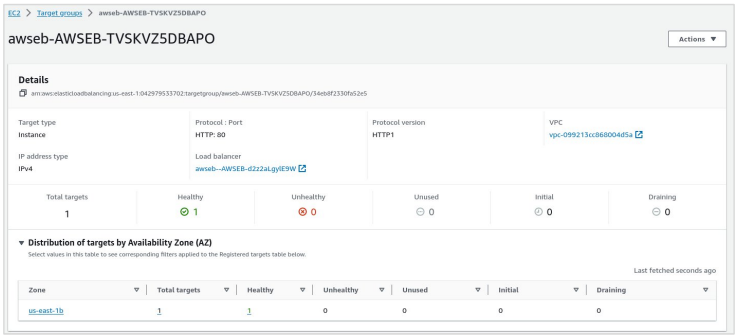
Elastic Beanstalk can be seen as a task automator composed of EC2, ECS, auto scaling, and load balancing task related actions [5]; thus, every one of this components can be reviewed in their own sections inside the console.



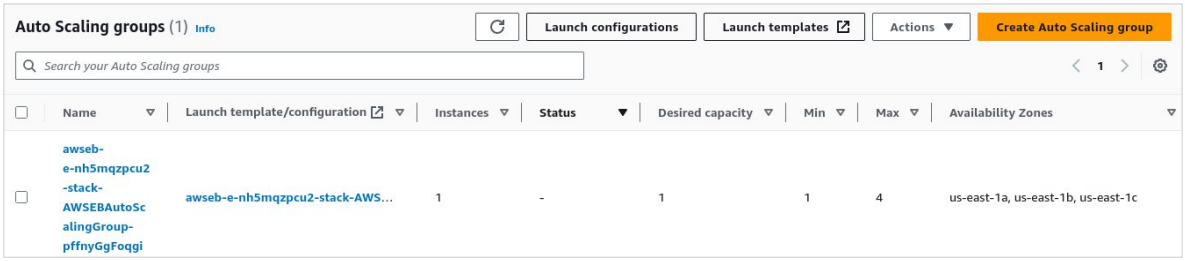
One single EC2 instance was deployed.



A single application load balancer spanning over *6* availability zones it's also created.

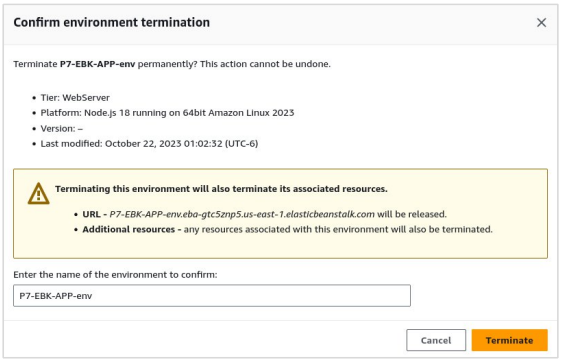


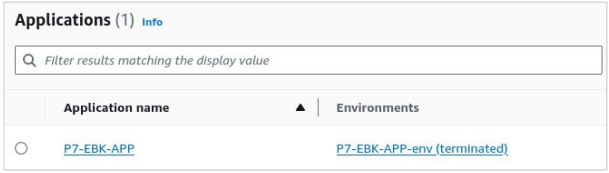
The target group bounded to the load balancer shows a single healthy instance (expected behavior).



Also, an auto scaling group it's created with a desired capacity and minimum of one instance (the one seen before) and a maximum of four.

After this information has been confirmed, the application *environment* (not the app) can be deleted. This can be done by selecting the created environment and then selecting the "terminate environment" action from its dedicated contextual menu. The application itself will still remain available.

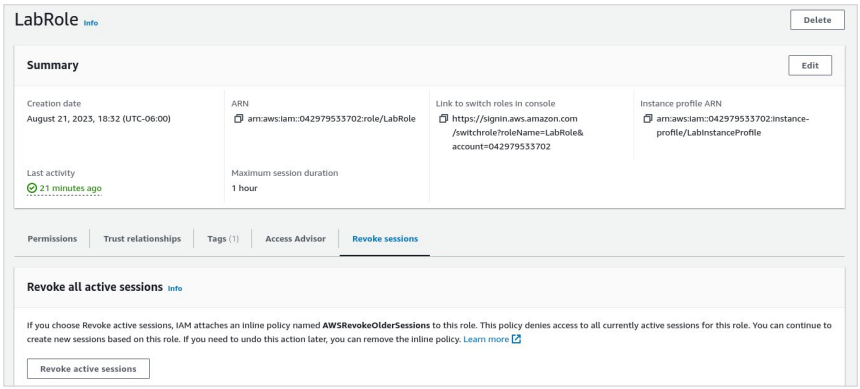


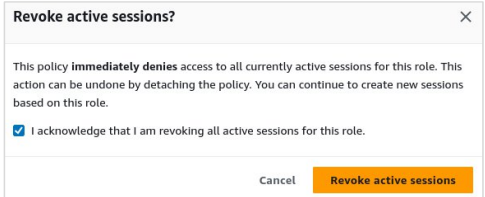


Blue/Green architecture

This previously discussed architecture now will be implemented, using the remaining application as an auxiliary resource.

> **Note:** the following steps are exclusive to this particular development, for the same academic account reasons as before.
>
> Inside the Identitiy and Access Management [IAM] [6], the *LabRole* role active sessions need to be revoked in order for new environment creation allowance.

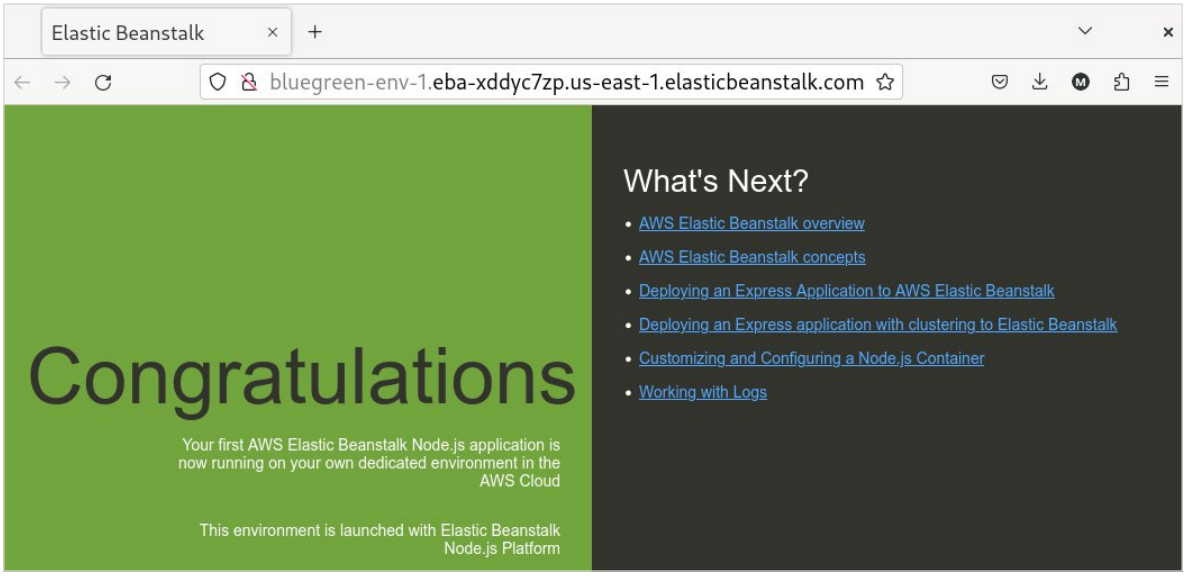




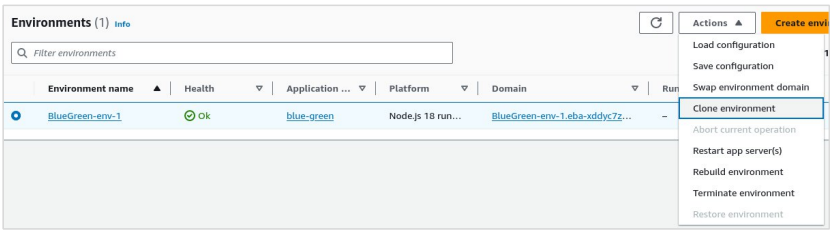
Heading back to the Beanstalk section, a new environment has to be created:

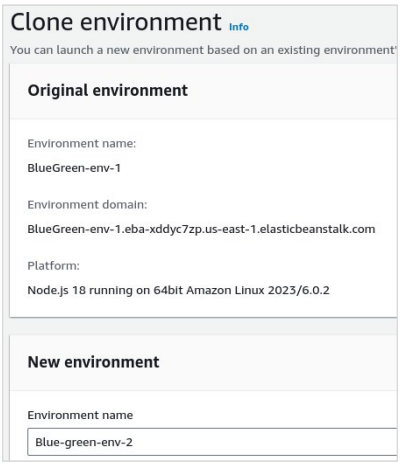
- Environment tier: Web server
- Name: blue-green
- Environment name: BlueGreen-env-1
- Platform: Node.js (leave default values for the rest)
- Application code: Sample application
- Presets: High availability
- Service access
    - Service role: existing LabRole
    - Key pair: vockey
    - Instance profile: LabInstanceProfile

As in the previous creation, remaining configurations can be leaved as default. After creation time has passed, the provided endpoint can be tested; it should return the same view as before.



Evidently, two environments are needed to achieve the blue/green architecture. With the current resources, the easiest way of achieving this is by cloning the past environment and changing its name.





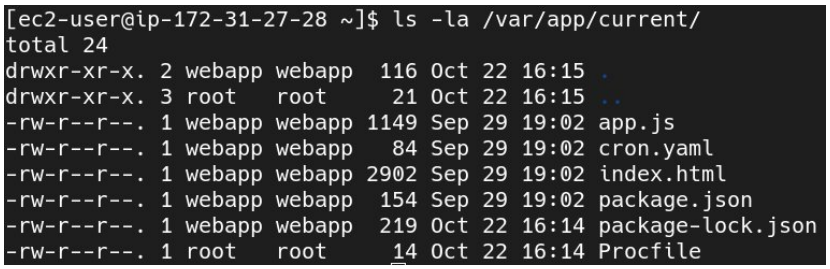
The new environment with its corresponding domain should now be listed along with the original one.

Environment name	Health	Application ...	Platform	Domain
BlueGreen-env-1	Ok	blue-green	Node.js 18 run...	BlueGreen-env-1.eba-xddyc7zp.us-east-1.elasticbeanstalk.com
Blue-green-env-2	Ok	blue-green	Node.js 18 run...	Blue-green-env-2.eba-xddyc7zp.us-east-1.elasticbeanstalk.com

Now, to mimic blue/green behavior, this sample application will be modified to change the previously seen green background to a blue one. To do so, the code provided would be downloaded, however, the same previous limitations impede doing this in an easy way, however, there's a way around this issue.

Knowing that one basic premise of Beanstalk it's EC2 instances deployment, one could access the created instance via SSH (as they're Linux based).

Research made [7] indicates that the application's source code can be found inside */var/app/current*.



As Elastic Beanstalk requires the entire with each new version, all the files shown above will need to be downloaded (the same premise as before). Following some Linux conventions, the user wouldn't have access to the current path when working with the *SCP* [8] utility, a tool to securely transfer files over the SSH protocol. To address this matter, this whole path contents would need to be moved to an accessible directory, such as the user's home at */home/ec2-user*. Once that step it's completed, two alternatives are found: editing the code inside the instance, *or*, outside the instances. In this case, just for practical purposes, the first option will be used to change the application background color.

```
[ec2-user@ip-172-31-27-28 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-27-28 ~]$ cp -r /var/app/current/ .
[ec2-user@ip-172-31-27-28 ~]$ ls
current
```

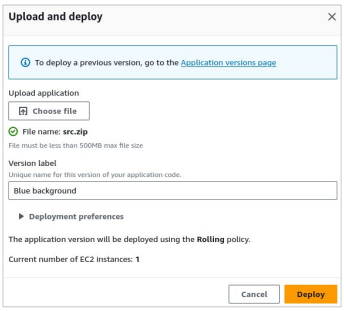
```
[ec2-user@ip-172-31-27-28 ~]$ cd current/
[ec2-user@ip-172-31-27-28 current]$ cat index.html | grep background-color
        background-color: #73A53E;
        background-color: #33342D;
[ec2-user@ip-172-31-27-28 current]$ # The first one it's the green background
[ec2-user@ip-172-31-27-28 current]$ vim index.html
[ec2-user@ip-172-31-27-28 current]$ cat index.html | grep background-color
        background-color: #062651;
        background-color: #33342D;
```

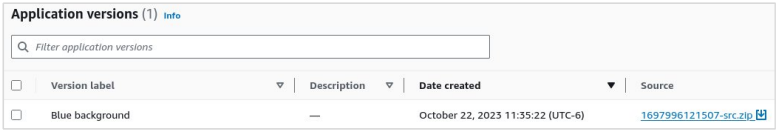
Outside of instance shell, in a local system, the previously mentioned tool can be used to fetch these contents.

```
[marcordero@fedora L07]$ scp -i ../../Resources/Keys/labsuser.pem -r ec2-user@54.145.241.221:~/current/. ./current
Procfile                                      100%   14     0.1KB/s   00:00
package-lock.json                             100%  219     0.8KB/s   00:00
app.js                                        100% 1149     3.0KB/s   00:00
cron.yaml                                     100%   84     0.4KB/s   00:00
package.json                                  100%  154     0.5KB/s   00:00
index.html                                    100% 2901     6.9KB/s   00:00
[marcordero@fedora L07]$ ls current/
app.js  cron.yaml  index.html  package.json  package-lock.json  Procfile
```

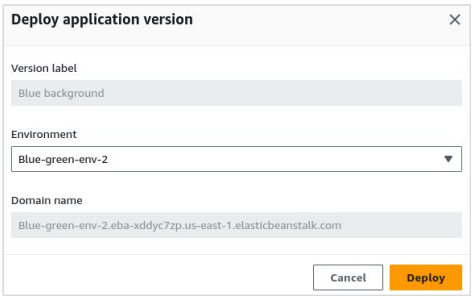
Once local access to these files it's gained, they have to be uploaded to the second environment in .zip format. **Note**: do not compress the directory containing these files, rather compress all the files inside. After compression it's done, the file can be uploaded as a new version and it'll be listed inside application versions.

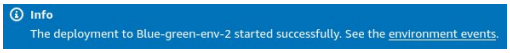
```
[marcordero@fedora L07]$ zip -r -j src.zip current/
updating: app.js (deflated 52%)
updating: cron.yaml (deflated 24%)
updating: index.html (deflated 71%)
updating: package.json (deflated 23%)
updating: package-lock.json (deflated 45%)
updating: Procfile (stored 0%)
[marcordero@fedora L07]$ unzip -l src.zip
Archive:  src.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     1149  10-22-2023 11:24   app.js
       84  10-22-2023 11:24   cron.yaml
     2901  10-22-2023 11:24   index.html
      154  10-22-2023 11:24   package.json
      219  10-22-2023 11:24   package-lock.json
       14  10-22-2023 11:23   Procfile
---------                     -------
     4521                     6 files
```

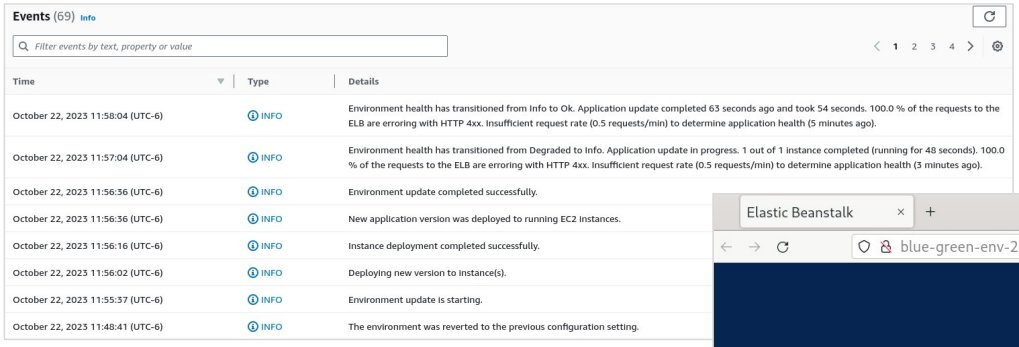




Inside this last view, the action of "Deploy application version" has to be with the second environment created.





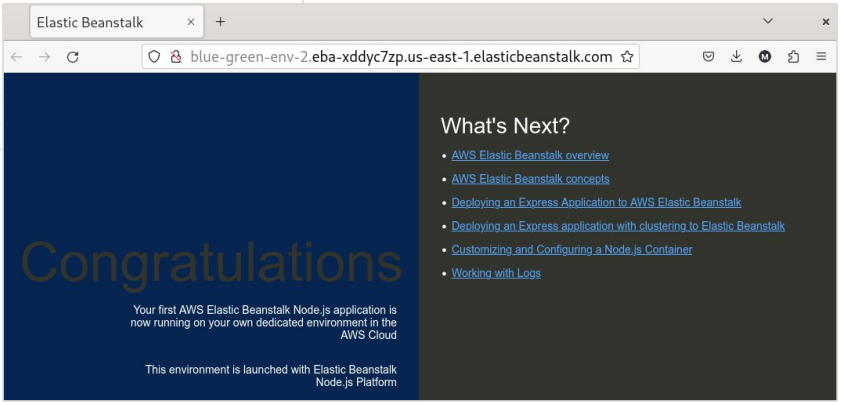


Events (69) Info

Filter events by text, property or value

1 2 3 4

Time	Type	Details
October 22, 2023 11:58:04 (UTC-6)	INFO	Environment health has transitioned from Info to Ok. Application update completed 63 seconds ago and took 54 seconds. 100.0 % of the requests to the ELB are erroring with HTTP 4xx. Insufficient request rate (0.5 requests/min) to determine application health (5 minutes ago).
October 22, 2023 11:57:04 (UTC-6)	INFO	Environment health has transitioned from Degraded to Info. Application update in progress. 1 out of 1 instance completed (running for 48 seconds). 100.0 % of the requests to the ELB are erroring with HTTP 4xx. Insufficient request rate (0.5 requests/min) to determine application health (3 minutes ago).
October 22, 2023 11:56:36 (UTC-6)	INFO	Environment update completed successfully.
October 22, 2023 11:56:36 (UTC-6)	INFO	New application version was deployed to running EC2 instances.
October 22, 2023 11:56:16 (UTC-6)	INFO	Instance deployment completed successfully.
October 22, 2023 11:56:02 (UTC-6)	INFO	Deploying new version to instance(s).
October 22, 2023 11:55:37 (UTC-6)	INFO	Environment update is starting.
October 22, 2023 11:48:41 (UTC-6)	INFO	The environment was reverted to the previous configuration setting.



Finally, the ultimate benefit of this blue/green architecture it's the seamless exchange between one and another. Inside AWS, a usual implementation would integrate Route 53 service [9], in which a CNAME modification would need to be done. Fortunately, Beanstalk provides with a URL swap function.

Environment name	Health	Application ...	Platform	Domain
Blue-green-env-2	Ok	blue-green	Node.js 18 run...	Blue-green-env-2.eba-xddyc7zp.us-east-1.elasticbeanstalk.com
BlueGreen-env-1	Ok	blue-green	Node.js 18 run...	BlueGreen-env-1.eba-xddyc7zp.us-east-1.elasticbeanstalk.com

Environment domain swap

Environment

Environment:

BlueGreen-env-1 (e-pbqkmvrizg)

Environment domain:

BlueGreen-env-1.eba-xddyc7zp.us-east-1.elasticbeanstalk.com

Environment domain to swap

Traffic will be redirected to the domain of the chosen environment

Environment:

Blue-green-env-2 (e-63rbrvpnzg)

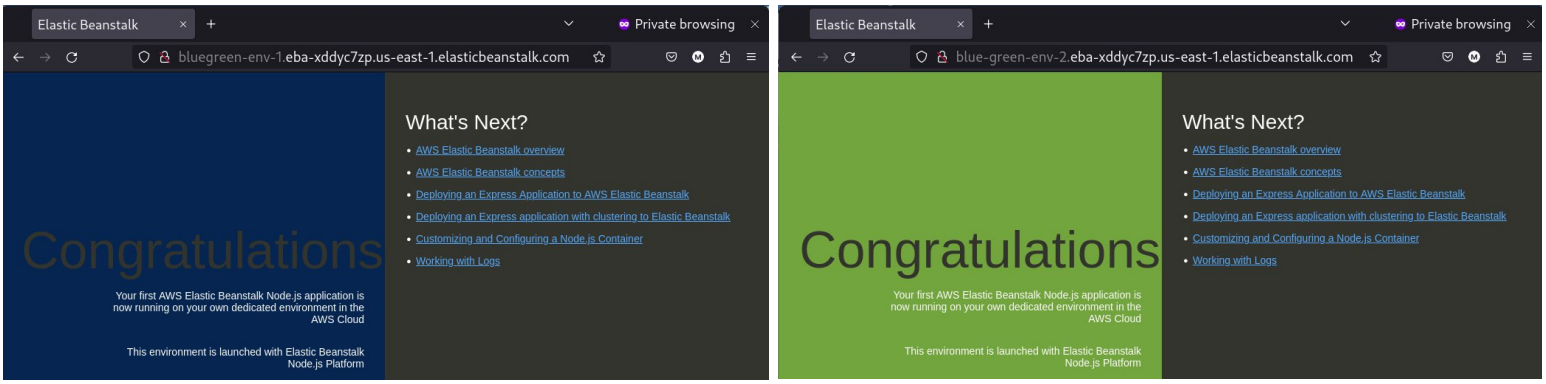
View all environments

Environment domain:

Blue-green-env-2.eba-xddyc7zp.us-east-1.elasticbeanstalk.com

Environment name	Health	Application ...	Platform	Domain
Blue-green-env-2	Ok	blue-green	Node.js 18 run...	BlueGreen-env-1.eba-xddyc7zp.us-east-1.elasticbeanstalk.com
BlueGreen-env-1	Ok	blue-green	Node.js 18 run...	Blue-green-env-2.eba-xddyc7zp.us-east-1.elasticbeanstalk.com

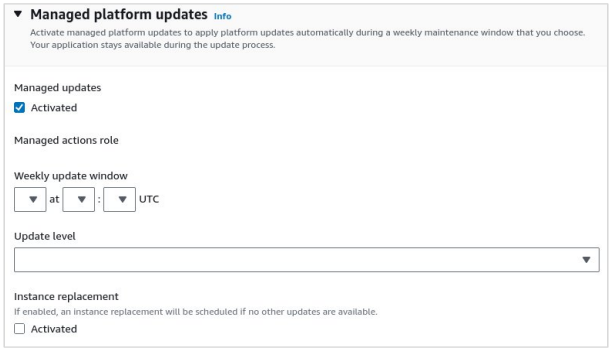
After the swapping it's done, the web view should also reflect this changes.



Now that this behavior has been demonstrated, all the resources created can be deleted, otherwise, AWS will continue charging for them. As this tool it's an automation entry point, all the stuff that it created will be automatically deleted without any manual intervention.

Additional features

Managed updates: AWS releases platform updates regularly, something that comes with two considerations: unexpected constant updates, and eventual update requirement. To address both points at the same time, managed updates can be configured to take place at a defined moment in time. This is a feature that Elastic Beanstalk also provides, but it needs to be manually enabled.



Script execution: Sometimes, system information or another type of action it's required to address certain requirements of the web application that would be deployed through Beanstalk. This can be done at various moments of instance state processing such as reboot or power off. Scripts can be run inside the instances to gather all sorts of data or to conduct a series of commands. To do so, the path */opt/elasticbeanstalk/* and *.platform/hooks* can be used to store scripts that will be executed inside the instances [10] [11].

Immersion day

As an additional practice step, a second demonstration of Elastic Beanstalk will be carried out through an AWS made immersion day tutorial. The contents of it aren't that far from what's been seen until this point.

First, the source code from another application will be downloaded through this link (might not be available in the future). This alternative approach demonstrates the use of what could be an own-developed application.

Note: as of today, the code provided *won't work* for compatibility reasons between the contents of the repository and AWS Linux instances. To correct this, unzip the file downloaded and modify the file *options.config* found inside *.ebextensions/*. Change the line

'`aws:elasticbeanstalk:container:nodejs:staticfiles:`'

for

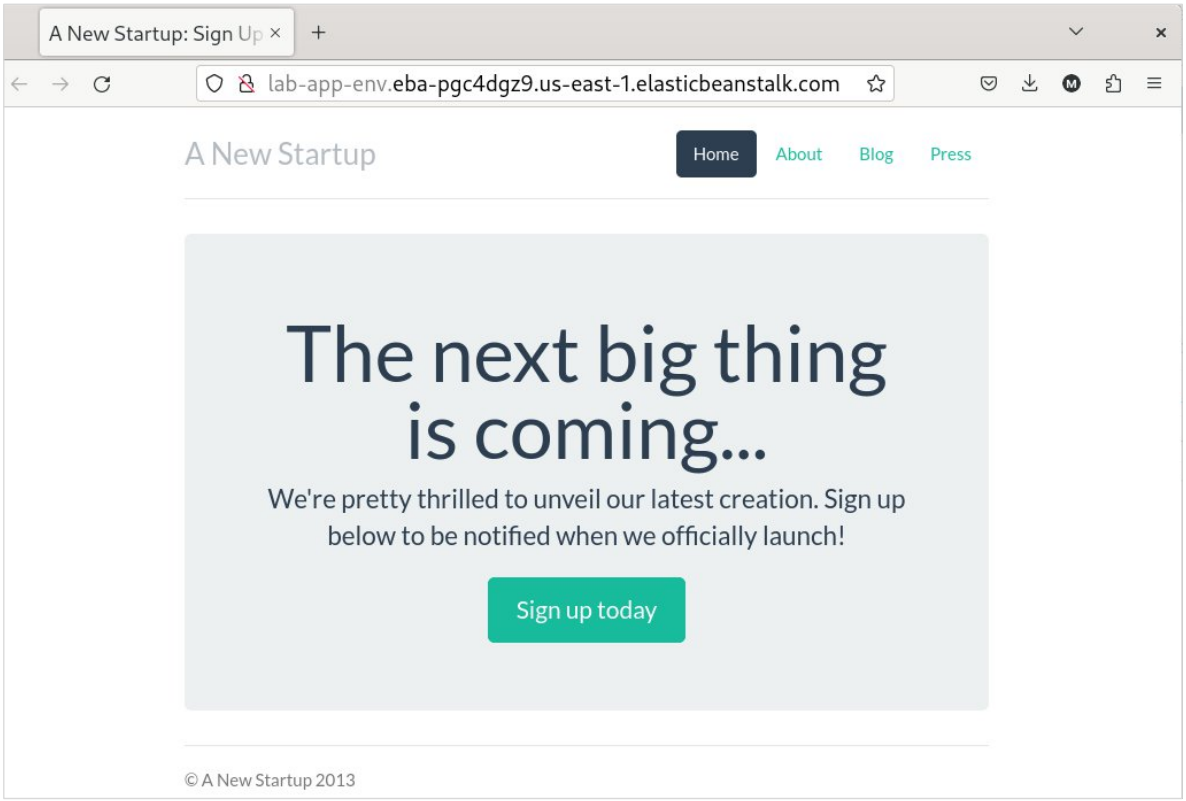
'`aws:elasticbeanstalk:environment:proxy:staticfiles:`'

After the change it's made, rezip all the contents.

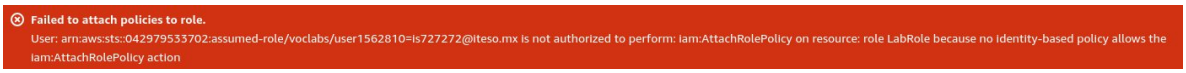
Once the codebase has been obtained and modified, a new environment inside Beanstalk section will be created:

- Application name: lab-app
- Platform: Node.js
- Application code: Upload your code —> Upload local downloaded file
- High availability
- Service access: LabRole – vockey – LabInstanceProfile

After the usual waiting time, the deployed web application can be accessed normally.



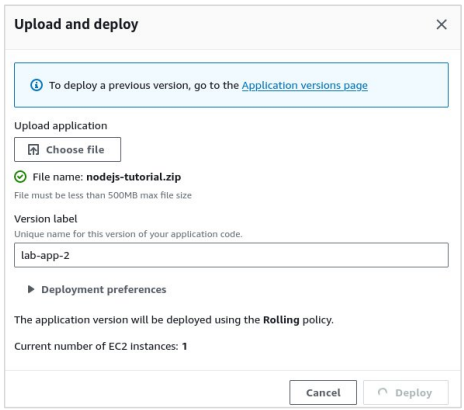
The next step indicated by the tutorial it's to *modify IAM Role permissions*, however, the same problem from before arise, as the account used for this demonstration it's limited to certain actions, and unfortunately, this one is prohibited.

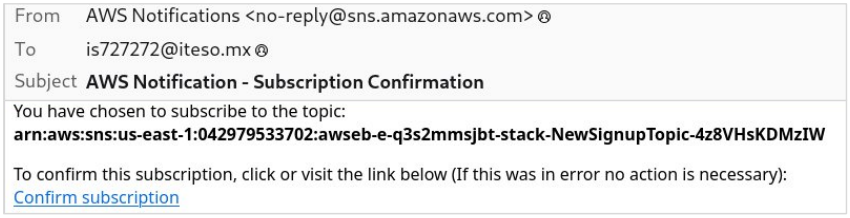


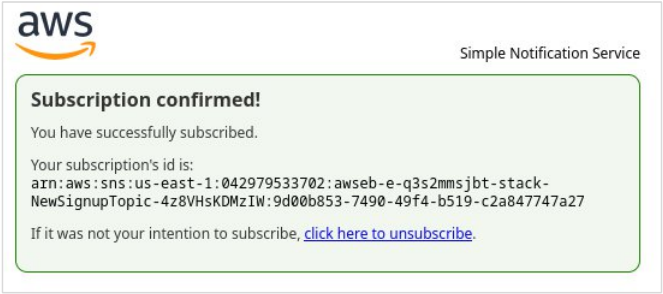
The next step requires yet another source code modification to receive notifications through SNS. To do so, modify the same file from before (options.config) and change the line starting with *NewSignupEmail* and replace the existing mail from a personal one (is727272@iteso.mx).

```
option_settings:
  aws:elasticbeanstalk:customoption:
    NewSignupEmail: is727272@iteso.mx
  aws:elasticbeanstalk:application:environment:
    THEME: "flatly"
    AWS_REGION: '`{"Ref" : "AWS::Region"}`'
    STARTUP_SIGNUP_TABLE: '`{"Ref" : "StartupSignupsTable"}`
    NEW_SIGNUP_TOPIC: '`{"Ref" : "NewSignupTopic"}`'
  aws:elasticbeanstalk:container:nodejs:
    ProxyServer: nginx
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /static: /static
```

Zip again all files and upload them as a new deploy. Once this is done, a mailing list subscription confirmation mail will arrive to email provided.

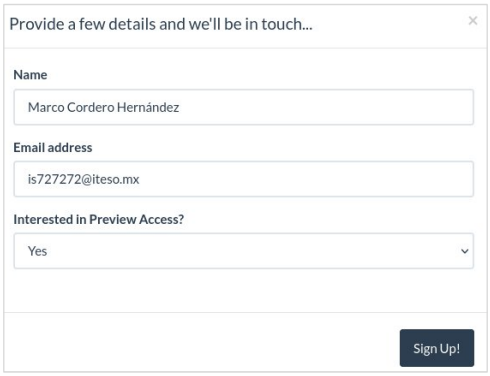


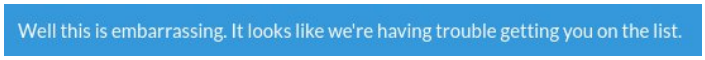




The previously seen page has a button inside it. It also has a functionality to mail interested people about this mockup application official launch. When clicked, a modal with a form inside will be

prompted. In this demonstration, an unexpected error arose, however, mail subscription isn't needed for this tutorial to be completed.



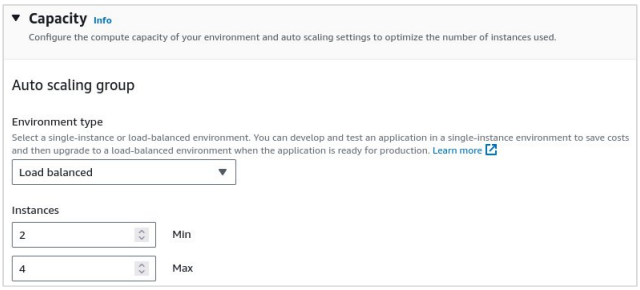


The intended behavior of this function was to mail the subscriber every time a new entry was made. The internal server error fix it's beyond the scope of this development.

The current implementation has a never seen before (in the current course) service: DynamoDB [12]. Inside it's own panel, in the tables section, a single table can be seen (created automatically by the application's deployment).

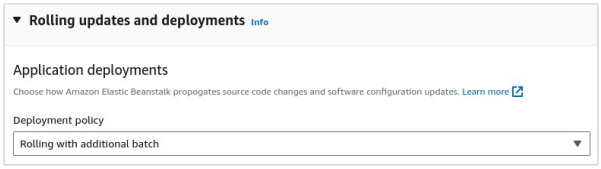
Due to the previous error, this table won't have any contents, however, a normal application flow would update it by uploading new entries containing mailing information.

Now, although high availability has been already configured, additional tweaking can be made to specify number of instances. Inside the created environment, go to "Instance traffic and scaling" and edit the minimum amount of instances from 1 to 2. Apply the changes.



After this modification is done, a new EC2 instance will be initiated. Of course, this can be seen in the corresponding panel.

Inside the exact same configuration, “Rolling updates and deployments” can be modified to roll with an additional batch. This option allows Beanstalk to perform application deploy on all instances, one at a time, but first creating an additional instance before doing so. This can be used to prevent errors when new version rollouts are scheduled and potential fails can occur.



Lastly, the tutorial suggest an additional environment deployment to corroborate a correct deployment setup, however, this behavior has already been explored in the blue/green implementation among the normal development of this practice.

The last step for this immersion day guide will be the exact same as before: deleting the environment. And as already stated, all dependencies and components of it will also be terminated without manual intervention.

Problems and Solutions

Several problems were encountered inside this development and they’ve already been explained in its contents. This is their list along with their solution:

- Revoke session failing: this problem couldn’t be solved per se, but, the workaround to it was to sign out from AWS Console, restart the laboratory from which access is provided, an then access again.
- Unhealthy status in Beanstalk environments: this problem was seen several times even after the cautionary action for its prevention was stated. The root of the problem was an incorrect code zipping, having two levels of depth (containing directory > parent directory > contents) rather than one. This hierarchy format it’s incorrect, and so Beanstalk couldn’t work because of it. The solution was to execute again the zip commands, but this time with appropriate parameters.
- Immersion day bad gateway: the tutorial used for the last part of the development it’s a little bit outdated, dating from April 2020 as publish date. Because of this, some of the steps described in its contents are no longer correct without file and structure modification. Research was made and the ultimate solution was to modify the *options.config* file, found inside the *.ebextesions* directory. Once that was done, the 502 status was gone and the mockup application could be seen.

Experiments and Results

After a very long and exhaustive development was made, the desire for experiments vanished completely, however, the immersion day realization can be considered as an experiment, resulting in deeper knowledge of Elastic Beanstalk. This semi-experiment demonstrate possible usage of the predominant service used in medium to large projects, or even small to medium personal projects.

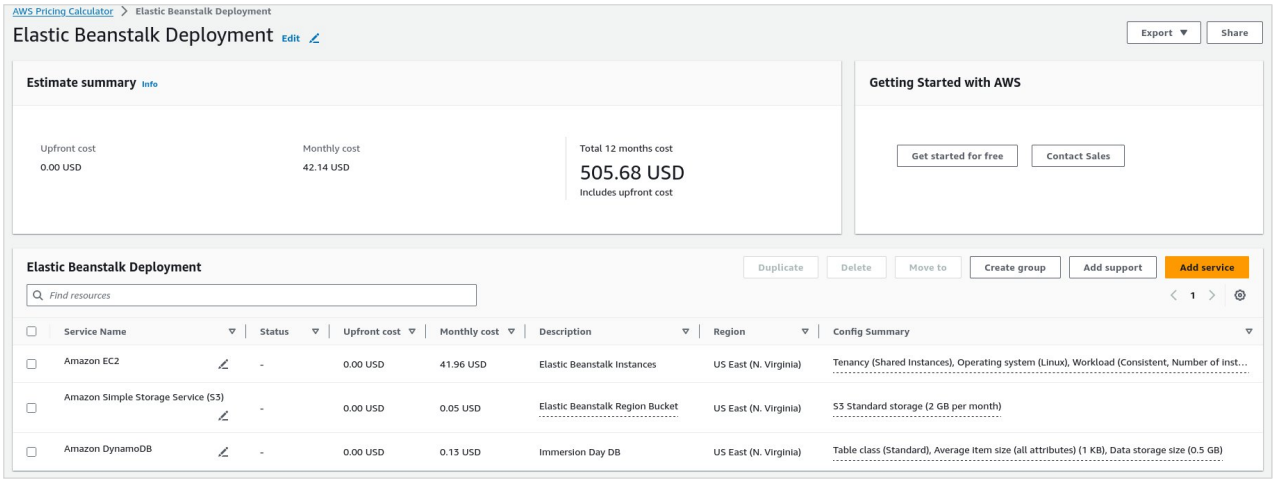
Budget Justification

The tool used for budget estimating doesn't allows for Elastic Beanstalk specification, however, knowing the fact that this service it's composed of several others, an approximation can be made.

Having:

- 5 EC2 Linux instances with t3.micro architecture and 8GB (inferred) gp2 storage
  - 1 for Elastic Beanstalk basic demonstration
  - 2 for Blue/Green architecture
  - 2 for Immersion day
- 3 Application Load balancers spanning over 6 AZ's
  - 1 for Elastic Beanstalk basic demonstration
  - 1 for Blue/Green architecture
  - 1 for Immersion day
- 1 Region S3 bucket
- 1 DynamoDB database for Immersion day

... The costs would be the following:



Conclusions

The development of previous practices has helped to understand exactly what has been done in the current one, as not a single automation usage can't be fully implemented without knowing what's being implemented. Elastic Beanstalk it's such an amazing feature of AWS for it's ease of configuration, velocity of deployment, and highly configurable capabilities.

Modern line of technology it's drifting towards heavy web development to replace any imaginable kind of software out there, so, developers can have the reviewed tool under their belt to focus primarily on their code and just deploy when they feel the need to do it.

Bibliography

[1] Aws.amazon.com. 'Types of Cloud Computing'. [Online]. Available: https://aws.amazon.com/types-of-cloud-computing/.

[2] BrainStation. 'Is Web Development a Good Career?'. [Online]. Available: https://brainstation.io/career-guides/is-web-development-a-good-career.

[3] Red Hat. 'What is blue green deployment?'. [Online]. Available: https://www.redhat.com/en/topics/devops/what-is-blue-green-deployment.

[4] Aws.amazon.com. 'AWS Elastic Beanstalk'. [Online]. Available: https://aws.amazon.com/elasticbeanstalk/.

[5] Aws.amazon.com. 'AWS Elastic Beanstalk Features'. [Online]. Available: https://aws.amazon.com/elasticbeanstalk/details/.

[6] Aws.amazon.com. ' AWS Identity and Access Management'. [Online]. Available: https://aws.amazon.com/iam/.

[7] Stack Overflow. 'Where does my application code sit on AWS Elastic Beanstalk EC2 Instance?'. [Online]. Available: https://stackoverflow.com/questions/64826125/where-does-my-application-code-sit-on-aws-elastic-beanstalk-ec2-instance.

[8] University Information Technology Services. 'Use SCP to securely transfer files between two Unix computers'. [Online]. Available: https://kb.iu.edu/d/agye.

[9] Aws.amazon.com. 'Amazon Route 53'. [Online]. Available: https://aws.amazon.com/route53/.

[10] Aws.amazon.com. 'Platform script tools'. [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/custom-platforms-scripts.html.

[11] Aws.amazon.com, 'Extending Elastic Beanstalk Linux platforms'. [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/platforms-linux-extend.html.

[12] Aws.amazon.com. 'Amazon DynamoDB'. [Online]. Available: https://aws.amazon.com/dynamodb/.