Integration of DevOps practices on a noise monitor system with CircleCI and Terraform

ESTEBAN ELIAS ROMERO* and CARLOS DAVID CAMACHO*, Universidad Distrital Francisco JosÃĺ de Caldas, Colombia
CARLOS ENRIQUE MONTENEGRO, Universidad Distrital Francisco JosÃĺ de Caldas, Colombia
ÃŞSCAR ESNEIDER ACOSTA, Universidad de San Buenaventura BogotÃą - Universidad Distrital Francisco JosÃĺ de Caldas, Colombia
RUBÃĽN GONZÃĄLEZ CRESPO, Universidad Internacional de La Rioja, EspaÃśa
ELVIS EDUARDO GAONA, Universidad Distrital Francisco JosÃĺ de Caldas, Colombia
MARCELO HERRERA MARTÃŊNEZ, Universidad de San Buenaventura BogotÃą, Colombia

Lowering pollution levels is one of the main principles of Sustainable Development goals dictated by the United Nations. Consequently, developments on noise monitoring contribute in great manner to this purpose, since they give the opportunity to governments and institutions to maintain track on the matter. While developing a software product for this purpose, with the growth in terms of functional and non-functional requirements, elements such as infrastructure, source code and others also scale up. Consequently if there are not good practices to face the new challenges of the software product, it could become more complex to refactor, maintain and scale, causing a decrease on delivery rate and the quality of the product. DevOps is an emerging concept but still hazy, which involves a set of practices that helps organizations to speed up delivery time, improve software quality and collaboration between teams. The aim of this paper is to document the implementation of some DevOps practices such as IaC, continuous integration and deployment, code quality control and collaboration on a noise monitor system to increase the product quality and automation of deployment. The final result is a set of automated pipelines which represents the entire integration and deployment cycle of the software integrated with platforms to improve quality and maintainability of the software components.

CCS Concepts: • **Computer systems organization → Cloud computing**; • **Networks → Cloud computing**; • **Computing methodologies → Artificial intelligence**.

Additional Key Words and Phrases: DeVOps, Serverless, CI/CD, CI/CD, Sound, Classification

*Both authors contributed equally to this research.

Authors' addresses: Esteban Elias Romero, eeromeroj@correo.udistrital.edu.co; Carlos David Camacho, cdcamachop@correo.udistrital.edu.co, Universidad Distrital Francisco JosÃĺ de Caldas, Ak.7 40b-53, BogotÃą, Colombia, 11021-110231588; Carlos Enrique Montenegro, Universidad Distrital Francisco JosÃĺ de Caldas, Ak.7 40b-53, BogotÃą, Colombia, cemontenegrom@udistrital.edu.co; ÃŞscar Esneider Acosta, Universidad de San Buenaventura BogotÃą - Universidad Distrital Francisco JosÃĺ de Caldas, BogotÃą, Colombia, oacosta@usbbog.edu.co; RubÃĺn GonzÃąlez Crespo, Universidad Internacional de La Rioja, Av. de la Paz, 137, 26006 LogroÃśo, La Rioja, EspaÃśa, ruben.gonzalez@unir.net; Elvis Eduardo Gaona, Universidad Distrital Francisco JosÃĺ de Caldas, BogotÃą, Colombia, egaona@udistrital.edu.co; Marcelo Herrera MartÃŋnez, Universidad de San Buenaventura BogotÃą, Carrera 8H 172-20, BogotÃą, Colombia, mherrera@usbbog.edu.co.

2158-656X/2022/2-ART $15.00
https://doi.org/10.1145/3505228

1 INTRODUCTION

Through the advances on software engineering, new practices appear over the entire software life cycle with the goal of making software products reliable, resilient and more efficient. In addition, other practices speed up the software life cycle by using automation allowing faster deliveries without affecting the integrity of the software. As a result of, some practices can be gathered into a concept named DevOps (Software Development and IT Operations).

DevOps is described as a culture or set of practices with many benefits, Pinheiro et al. point out some of them "increased organizational IT performance and productivity, cost reduction in software lifecycle, improvement in operational efficacy and efficiency, better quality of software products" [22]. The concept of DevOps is becoming wider, as it is an emerging concept, there are still confusion regarding its definitions and best practices, the above implies that at the time of its implementation there are certain risks [24], so the adoption of the culture of DevOps requires to start an in-depth investigation of the practices and the impact they have in the software product and the organization.

The application domain of DevOps includes cloud architectures with good results, research by Prashant Agrawal and Neelam Rawat concluded that DevOps also offers scalable infrastructure to cloud while speed testing and deployment processes [2]. Practices such as continuous integration (CI), continuous deployment (CD) and infrastructure as code (IaC) are part of DevOps with applications in the cloud, they play an important role in the automation of most tasks in the integration and deployment steps of all the software pieces.

On the other hand, when environmental noise studies are carried out, field measurements or simulations are started to determine the sound pressure level in decibels in a certain place. The foregoing, in order to establish whether the levels comply with the limits according to land use. However, by having only data in decibels, it makes, for certain cases, determining the type of acoustic source that causes environmental noise pollution is not an easy task. Therefore, with the emergence of artificial intelligence and classification algorithms applied to audio, improvements in identification of predominant noise sources have been achieved [25], [28]. This allows environmental authorities to focus their efforts on the control and mitigation of these sources, so that, noise levels in decibels approach values that do not affect people's health and deteriorate the soundscape.

The present case of study is a software product which is called Noise Monitor System (NMS), a system to make real-time sound inferences and visualization of results. It has multiple components in a cloud architecture each one with differences in its infrastructure. The integration of the last mentioned practices and some others on this system is the aim of this paper focusing on removing most parts of manual work, automating tasks, strengthening clean code, and seeing its effects on time and quality of the software. In addition, the present investigation is motivated by the advances that in recent years have been developed in various fields of computer science and acoustics. Within the areas that it integrates, it can be highlighted that it has an infrastructure in the cloud which is continuously interacting with data acquisition systems that provide audio information, from which sound environment characteristics of a city are obtained such as environmental noise and sound source classification. In that way concepts of cyber-physical system (CPS), artificial intelligence and IoT are applied with the aim of subsequently supporting urban planning under the concept of Smart cities

2 STATE OF THE ART

2.1 DevOps, AWS, SonarCloud and IT emerging collaborative technologies

As it stated in [8], DevOps has emerged in an attempt to address several issues encountered in the past, by putting the primary focus on the collaboration between development and operations. Originally, DevOps emerged from the Software Agile Manifesto, where several issues began to be considered: (1) Individuals and interactions over processes and tools, (2) Working software over comprehensive documentation, (3) Customer collaboration over contract negotiation, (4) Responding to change over following a plan.

Patrick Debois used the term DevOps for the first time in 2009 [15]. This concept refers to the set of core principles to effectively address the problems that earlier methodologies of software developing were facing. In 2020, an exploratory study on the DevOps IT Alignment Model was performed [16], where the practice of these techniques was analyzed in order to explore connections with prior theory. Here, it is proved that continuous integration of software and knowledge sharing increases the level of IT alignment.

On the other hand, AWS (Amazon Web Service) supports numerous DevOps principles and practices that enable IT agility. One of the basic principles of this technology is the consideration of Infrastructure as Code, Continuous Deployment, Automation, Monitoring and Security [9]. AWS offers a broad set of global cloud-based products including compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security, and enterprise applications. Another AWS service is Elastic Container Server (ECS) which provides secure, resizable compute capacity in the cloud.

Finally, the last but not least of these technologies used here, SonarCloud is a cloud-based code quality and security service. In this sense, SonarCloud detects bugs, vulnerabilities, and code smells in a wide variety of languages.

2.2 Recent implementations of Noise Monitoring Systems

Several technologies have emerged during the last decade for Network Implementation of efficient Noise Monitoring Systems. WASNs (Wireless Acoustic Sensor Networks can be classified in two categories depending on their nodes spatial location: fixed or mobile stations. In the later, mobile smartphones are key part of the chain. Transmission of real-time data from noise measurement sensors is a challenge that current communication technologies face. In this sense, research has been done in order to fulfill this task.

For environmental monitoring, WSN (Wireless Sensor Network) enables to capture data with high quality and affordable costs. It is composed of a set of nodes deployed in the area of interest for collecting and analyzing acoustic parameters values for definite periods of time. The Tmote Sky platform [31] is used, however with no reports upon accuracy of the results.

The project developed in [14] demonstrated through protocol comparison that CTP (Collection Tree Protocol) with LPL (Low Power Listening) performed better than CTP along with DMAC protocols. Afterwards, a Bluetooth-based solution for noise measurement supported a maximum of 5 noise sensor nodes but without multi-hop communication, therefore with limited application's scale.

In [26], a new monitoring concept is used to automatically assign the measured sound level to different noise sources. For this purpose, a supervised noise source classifier detects the activity of target noise source in the presence of other interferences.

In the same way, the design and implementation of the networking and communication part of a WSN application for measuring industrial and residential acoustic noise is described in [21]. CiNet (Continuous Innovation Network) cross-layer protocol stack is used for this purpose. Nowadays it has been deployed in Kokkola (Finland).

One of the latest implementations of WSN in 2020 is [27] where the application of a Wireless Acoustic Sensor Network is proposed for the long-term analysis of psychoacoustic parameters of the acoustic environment in an university campus. This study is particularly interesting because it compares the noise levels during pandemic period where lock-down was ruled and regular activity time.

Data collected over three years by a network of acoustic sensors deployed in Barcelona, Spain, were used in [29] to train several clustering methods. Since this study ended recently (2021), the effect of Covid Lockdown was evidentiate. Sensor clusters obtained by the algorithm are compared with the areas defined in the strategic noise map (provided by Barcelona city council). The conclusion here, was that the developed k-means model identified most of the locations found on the overcoming map. In general, Wireless communication technologies

exhibit a variety of advantages such as (1) ease of deployment, (2) real-time data capture, (3) storage and data processing, (4) generation of dynamic noise maps. The later enables the user to understand how the phenomenon evolves through time, empowering the user with more robust prediction and evaluation tools.

2.3 Binaural evaluation of soundscapes

During the last decades, studies have been focused on the evaluation of noise with the binaural paradigm. As it is demonstrated in [37], the binaural recordings will render corresponding perception more consistently than the monaural.

Lately, IVR (Immersive Virtual Reality) techniques has been also used for these perceptual evaluations, as it is shown in [38]. These systems strongly rely on sound binaural techniques in order to give to the user the spatial audio impression of the scene. One of the most transcendental findings is the fact that when âĂIJrealism,âĂİ âĂIJreverberance,âĂİ and âĂIJdirectivityâĂİ are involved in evaluation, then binaural recordings bring more accurate results.

Locally, in the South American continent, a comparative study in the cities of Brasilia and Bogota, also elucidated that the binaural parameters application on the evaluation is suitable [17].

2.4 Latest trends in Cloud Computing and Internet of Things

Designing distributed applications as suites of independently deployable interacting components may lead to the adoption of strategies such as Answer Set Programming (ASP), in synergy with agents defined in DALI Microservices. This is well reported in [11].

Regarding sensor technologies, the unification of streaming sensor data has always been understood as a challenging problem. Linked to this technology, in [7], strategies such as Incremental Clustering Driven Automatic Annotation for IoT Streaming Data (IHC-AA-IoTSD) using SPARQL to improve annotation efficiency. The inclusion of data classification, incremental hierarchical clustering and querying the extracted data is described here.

Finally, TD2SecIoT (Temporal Data-Driven and Dynamic Network Layer Based Security Architecture for Industrial IoT incorporating Elliptic Curve Cryptography and Nth-degree Truncated Polynomial Ring Units ensure confidentiality and integrity and have become lately as standards against different attacks and performance measures.

3 NOISE MONITOR SYSTEM ARCHITECTURE COMPONENTS

In this section the main components employed to develop the NMS architecture are described. Figure 1 shows all the components, its technology, and their relation in the NMS. Currently only the services which do an inference process over an audio file use a mapper.

3.1 Stations

Stations are devices located on different geographical locations, they use a embedded system Raspberry Pi 4 and two electret UMIK-1 omnidirectional microphones to record sounds from the environment to be processed later. This electronic hardware solution was selected due to the good cost-benefit ratio offered by this type of embedded systems for solutions based on IoT. Additionally, these microphones have shown good performance when it comes to making audio captures for measurements of ambient noise [3], in addition to facilitating and simplifying the connection with a raspberry Pi. The use of two microphones is justified in the fact of conducting future studies related to binaural audio. Monitoring stations will record audio data during 24 hours in order to obtain noise acoustic descriptors based on energy as Equivalent noise Level (Leq) Day-evening-night level (Lden) and Day-night level (Ldn). Additionally, spatial acoustic indicators as $\tau IACC$ and $\omega IACC$ will be calculated to characterize noise sources. On the other hand, the stations are powered by a 60w solar panel and an electric

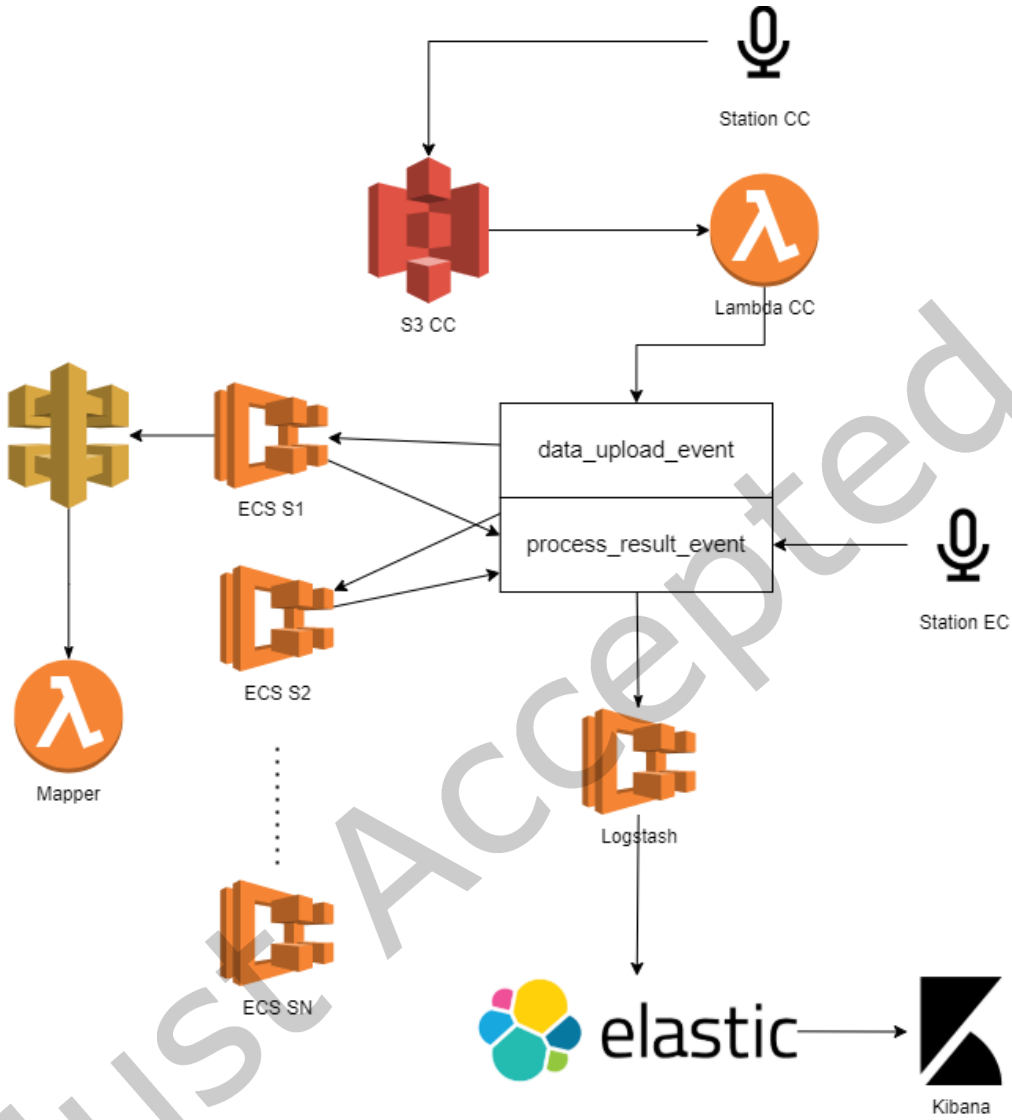

Fig. 1. NMS Architecture

battery to work 24 the whole day seven days per week. Finally, data is transmitted through wifi to Amazon Web Services (AWS). This platform was chosen due to the facilities we had, however, because Terraform is used, the implemented architecture allows easy migration to another cloud services. In general, audio recorded by the monitoring stations is treated as explained below.

3.2 AWS S3 as temporal audio storage platform

AWS S3 is a object storage system in the cloud created by Amazon Web Services. This type of storage save information in a flat way (all data at the same level). Each object is composed by a unique key, metadata and the data itself. This type of storage is used on cloud computation to work with huge amounts of data. It's recommended on this type of storage to use static data like audio files and images [18, 30].

This platform is used to save records and all the required metadata recovered from stations. Once data is created in the AWS S3 bucket it creates and event named ObjectCreated to notify other Amazon service about the creation of a new object. In the case of the (NMS), the event is delivered to a AWS lambda function.

3.3 Apache Kafka as streaming Platform

Kafka is an event streaming platform which allows to publish (write), subscribe (read) and store streams of data. Kafka defines some concepts relative to the main function of the platform [5].

- Topic: Set of related data.
- Producer: An API which send streams of data to a topic.
- Consumer: An API which reads streams of data to a topic.
- Broker: Kafka runs as a cluster and can be expanded to multiple regions, the nodes or servers are called the brokers.
- Group: A set of consumers identified by the same identification, data is delivered only once and processed by just one member of the group.
- Partition: A division of the topic, where data of the topic is distributed as the producer requires. There can be as many consumers as partitions in the topic.

Thus, on the NMS two topics on Kafka were defined:

3.3.1 data_upload_event. In this topic the set of messages produced by an AWS lambda function are published. Each message has a number of consumers that do not overcome which is the name of the record.

3.3.2 process_result_event. Finally, by means of this topic the response made by the services on ECS (Elastic Cloud Computing) are published. Additionally, device information, name of the algorithm, inference result, and other operations are saved as a JSON (JavaScript Object Notation) in order to be processed by ES (Elastic Stack).

On the NMS the relation between a group of consumers and a topic with its number of partitions is important in terms of scalability. As illustrated in Figure 2, each partition is associated to only one consumer of a consumer group, but if it is possible, a consumer can read from multiple partitions only when they are not associated to other consumer on the same consumer group. Every time a new consumer is available (a new service reading from Kafka) automatically is created. Likewise, partitions are balanced and the new producer is assigned, but it is important to highlight that the number of consumer do not overcome the number of partitions available, otherwise the new consumers will not be assigned. The last behavior is how the partitions and consumers of the topic "data_upload_event" works. The topic "process_result_events" only requires one partition.

3.4 AWS Lambda as Kafka producer and inference mapper

AWS Lambda is serverless service that only requires the code to be provisioned, it supports multiple programming languages such as: Java, Go, PowerShell, Node.js, C Sharp, Python and Ruby. When this service is used pricing is calculated by the number of executions and duration of process with the memory assigned, in some configurations this service can be cheaper than other services as EC2 [4]. It is recommended to continuously verify prices on AWS as they can change any time. On NMS two lambda functions were defined:

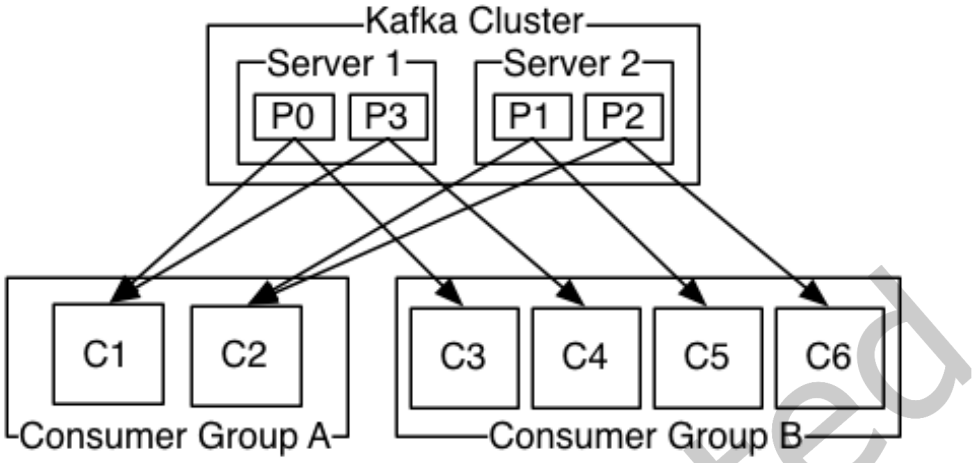

Fig. 2. Distribution of partitions with consumers associated to a defined group [19]

3.4.1 Data upload event producer. Events from AWS S3 are delivered to the lambda function with the name of the object created, then lambda publish such name in the topic "audio-upload-event" using the library KafkaJS with a round-robin distribution [20].

3.4.2 Inference mapper. This function maps results made by the inference algorithm to a common set of tags. In addition, this function is used by the inference service.

3.5 ECS Fargate for inference services

ECS (Elastic Container Service) is a service created to address the complexity of managing containers, allowing scalability and monitoring over all the components on the service.

Clusters can be executed using Fargate, which is a serverless engine, which removes the need to deal with operational logic (servers management, creation or termination); thus, only some container specifications like CPU, memory and network must be set.

NMS uses ECS primarily to run all the services which process the incoming audio files and publish the results.

3.6 Elastic Stack to transform and visualization

Elastic Stack is a set of tools integrated by Kibana, Elastic search and Logstash, where each tool is a component of a pipeline to transform, save and visualize data. The function of each tool is presented as follows [13]:

- Logstash: Consumes, transforms, and outputs data using a configuration file.
- Elastic Search: A distributed analytics and analytics engine.
- Kibana: A tool to visualize and query data using elastic search. Kibana provides graphs of different types, dashboards and maps for geographical data.

Figure 3 describes the stream of information from Kafka to Kibana. On NMS Logstash transforms the messages published on the topic "process_result_event" to a JSON before sending them to elastic search. Then, results can be visualized and filtered using Kibana.

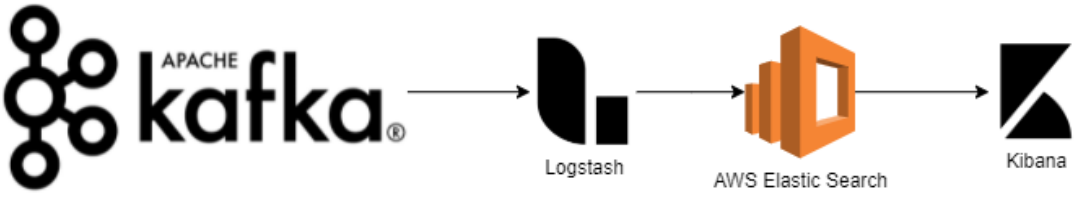

Fig. 3. Elastic pipeline on the NMS with Kafka as source of data

3.7 Asynchronous data processing

On NMS each service connected to the Kafka topics process the data and publish the results without a dependency, on other services. Each result will update a document on elastic search with the same identifier of the audio file processed.

3.8 Computing models

NMS architecture supports both Cloud and Edge computing models. On cloud computing data is processed by the services on the architecture while edge computing data is processed in a station which supports and can handle all the operations made in the cloud.

3.8.1 Cloud computing. On NMS all the data which is published on the topic “data_upload_event" is processed in the cloud. Once the event is published all the services subscribed to the mentioned topic will execute its respectively process (Currently NMS counts with two services, a service of audio inference based on a CNN implementation [1, 12] and a noise level calculator). Cloud computing have some advantages over edge computing:

- If a new algorithm must be integrated to NMS is easier to added to the architecture than to each station. (It could be solved using versioning and updating software of stations).
- Instances capacity of each service can be changed any time. This is more difficult or impossible on stations.
- Adding new functions on stations have the risk of monolithic applications. Cloud computing is based on micro-services.
- The architecture of some devices is not compatible with some libraries (like Tensorflow or torch), then some solutions come from unofficial repositories. Services on cloud uses an architecture compatible with those libraries.

3.8.2 Edge computing. On NMS when data is published on the topic “process_result_event" it means that it was already processed inside the station so there is no need to call the services. Currently on edge computing an audio inference algorithm (using YAMNet a pretrained deep net which employs the Mobilenet_v1 depthwise-separable convolution architecture [35]) and the noise level calculator were implemented. Edge computing has some advantages over cloud computing:

- Works better on places with a high latency or where networks are based on data (pay for each byte).
- The flow over the architecture can be faster, because doesn’t require to upload data to S3 and goes directly to publish the results.
- Reduces the billing on AWS (less services and less space usage on S3).

4 CONTINUOUS INTEGRATION AND DEPLOYMENT ON NMS

Mitesh [34] defines Continuous Integration (CI) considering that software development practice needs the constant work of the team in an integrated way, also mentions one of its uses where organizations can perform

continuous integration of its code and verify the integrity of the code in early stages of development. Consequently, continuous integration allows teams to integrate their work and verify (with unit tests) that everything is on optimal conditions before going forward to the next step. On the other hand, continuous deployment (CD) is also a software practice which takes place after CI where the changes produced throughout the production environment must be made constantly and automatically [32], in other words, make any change available for customers.

Each repository of the NMS System has two main git branches, Master and QA, the function of each one is the follows:

- Master: The production branch, where the software product completes its last step where it will be available for consumers.
- QA: This is main branch to integrate all changes on the code. All new code is tested and analyzed before can be pushed to this branch.

Moreover, a Pipeline is a set of stages that run one after another or simultaneously and can have an associated behavior if a specific event takes place. On NMS pipelines have three stages:

- Test: The application of unit test over all the code to verify the integrity of the development and avoid that a functionality could be compromised.
- Build: The component is prepared to be deployed (create an image, compress the code, build the code, etc.).
- Deploy: Executes and make available the component to the environment.

As illustrated in Figure 4 the main branches (Master and QA) have a complete CI/CD pipeline, while FIX/FEATURES (branches to integrate new code) only requires test and build steps. This is because FIX/FEATURES branches only need to be tested before being integrated to main branches so from QA and upwards code is already tested and the risk to compromise the development is lower.

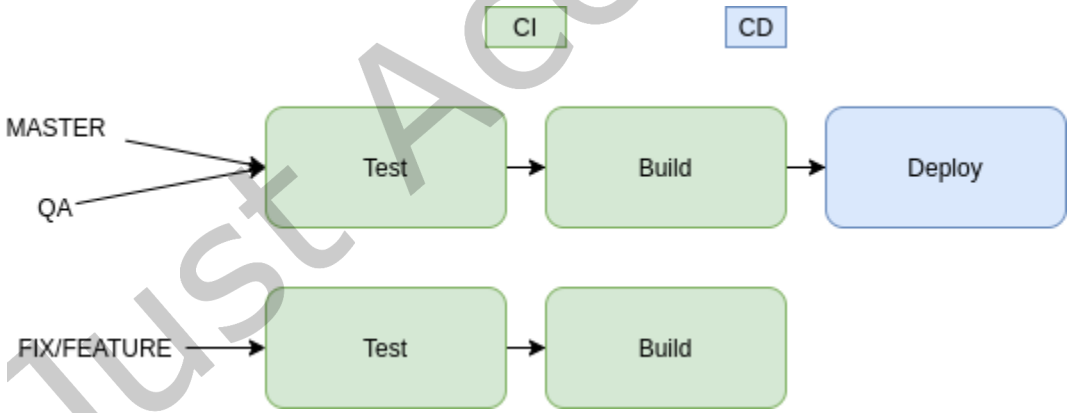

Fig. 4. Diagram of branches and pipelines, color indicates whether is part of continuous integration or continuous deployment

5 AUTOMATION OF PIPELINES WITH CIRCLECI

5.1 Basic Workflow

On NMS CircleCI [10] is the platform used for continuous integration and deployment. It can be associated to git repositories on GitHub to start to develop a pipeline. All the configuration is made by using a YAML file.

Some main concepts must be introduced in order to build a pipeline in circleCI:

- Step: A command which is executed within a job.
- Job: Collection of steps which runs in a single unit (a container, machine).
- Workflow: Set of rules to define the jobs and their order of execution.

A job requires a set of steps and a container (where all the steps are executed), as seen in Algorithm 1 “test" is a job which prepares the machine and run the tests. If a step fails the job breaks down and consequently the pipeline stops so the code can‘t be integrated protecting main branches.

In a Workflow jobs are set in order of execution, the attribute “requires" (see Algorithm 2) defines the step which must be completed before running the actual job; the attribute “only" specifies the branches allowed to run that job. The workflow is the representation of the pipeline defined on Figure 4.

5.2 Orb

Orbs, which are a fundamental part of CircleCi, allows to reduce steps and configuration by just calling a determined job (already defined in the orb) in the workflow adding only additional attributes, filters, and environment variables if it is required. As a result, Orbs reduce complexity in a configuration and the time to implement new workflows in a pipeline.

Orbs are defined in the attribute “orbs" in the YAML file of circle CI. An example is the orb “aws-ecr/build-and-push-image" which is used around the system when an image must be built and uploaded to AWS ECR (Elastic Container Registry). As seen on Algorithm 3 there is no need to create a job or write any command, it just needs the required attributes to work. Without this Orb additional steps must be added like installation of AWS CLI and all the image building process.

5.3 Automation of the inference service pipeline

Figures 5 and 6 shows the inference service pipeline in the circleCI interface, the jobs are classified by stage as follows.

- Test: Runs the jobs test and sonar (a tool to verify the quality of the code) simultaneously.
- Build: Builds and push the image to AWS ECR using orb. It requires both test and sonar jobs to be completed to start.
- Deploy: Runs a job where terraform (a tool for IaC) provides the infrastructure of AWS ECS and deploy the image using the configuration files (terraform is discussed later on this document).

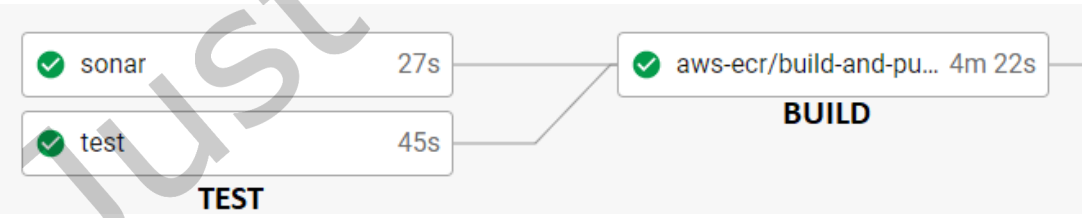

Fig. 5. Inference service pipeline part A, test and build stage

The workflow in the configuration file that represents the pipeline on Figures 5 and 6 can be seen in Algorithm 4. The use of the attribute “context" allows to share environment variables through multiple projects.

5.4 Automation of AWS Lambda services and workflow persistence

AWS Lambda functions uses zip files with both source code and dependencies to deploy. Creation of a zip file can be made inside the container of a job during the build process.

Fig. 6. Inference service pipeline part B, build and deploy stage

Sometimes some artifacts (files or directories) must be saved in order to be used later by another step. The pipeline of lambda on NMS uses the step “persist to workspace" (See Algorithm 5) to save the zip file created to be used later in the deploy step.

Finally, the step âĂIJattach workspaceâĂİ brings the last saved zip file to the actual job, then deployment is made by terraform as can be seen in Algorithm 6.

5.5 Automation of Logstash

Logstash is also deployed on AWS ECS using terraform and the orb of AWS ECR. It only requires a configuration file which defines the pipeline between Kafka and Elastic Search.

6 INFRASTRUCTURE AS CODE

Relation between DevOps and infrastructure as code (IaC) can be seen from the point of view where DevOps encourages, as far as software development is concerned, to use the same standardized language when expressing automation code, scripts, models, configuration parameters, among others [6].

IaC takes care over all the infrastructure operations (creation, update and removing) using a definition of the resources without the intervention of developers or operations members, so both only need to monitor the deploy status or be notified whether it was successful or not. Terraform [36] is a tool for IaC and uses a syntax for configurations which is called HashiCorp Configuration Language (HCL), with this syntax it’s possible to describe the infrastructure of NMS on Amazon Web Services (but it is also compatible with Azure, Google cloud, Kubernetes, and other providers).

6.1 Terraform provider and backend on NMS

Providers are “plugins that implement resource types" [36], they allow to use terraform with the cloud provider required (NMS uses AWS). As seen in Algorithm 7 it is required some AWS keys and the region. In NMS every key and other variables are defined inside environment variables.

Terraform keeps a state file to map resources to the terraform configuration, keep track of metadata, and to improve performance for large infrastructures [36]. NMS saves the state file in a AWS S3 bucket, so it is always available when deploying from different machines. Algorithm 8 shows the configuration to save it in a bucket.

6.2 Deployment of Amazon services on NMS services with Terraform

Configuration behind each component of NMS involves multiple entities or Amazon services. Making all this configuration by hand on every time a deploy or integration event takes place, reduces efficiency, and could lead to errors which could take more time to resolve (maybe repeat the entire pipeline by hand again).

Keeping track of all those entities is one of the reasons why NMS uses IaC. With Terraform each entity is configured only one time, after that, creation and update are automatic. Elements showed in Figure 7 are the

services and associated characteristics which must be set on NMS, likewise, components with a red square are “Shared components", this means that multiple AWS services could use them, for this reason they have an individual deployment.

Fig. 7. AWS architecture components in NMS

Algorithm 9 shows the configuration of a service using terraform. If something is changed within the configuration, terraform updates the entities on AWS (It updates the infrastructure but do not create or removes unless if it necessary), so there is no need to go to the AWS console or use the CLI tool to make changes manually. It is also possible to monitor the status of terraform with CircleCI (Figure 6).

A way to observe the entire tree of dependencies between AWS entities on a project is by using a dependency graph which is generated by terraform. Graphs allow to verify how terraform will create or destroy infrastructure. Figure 8 shows, the dependency graph of infrastructure, arrows indicate dependency and the nodes have the name of the AWS entity and the identifier.

Using variables on terraform is a good practice to centralize names, keys and other configuration values which use to change. As presented in Algorithm 10, variable definition includes a description and a default value (both optional). To associate a terraform variable with a environment variable, the last must include the prefix “TF_VAR_".

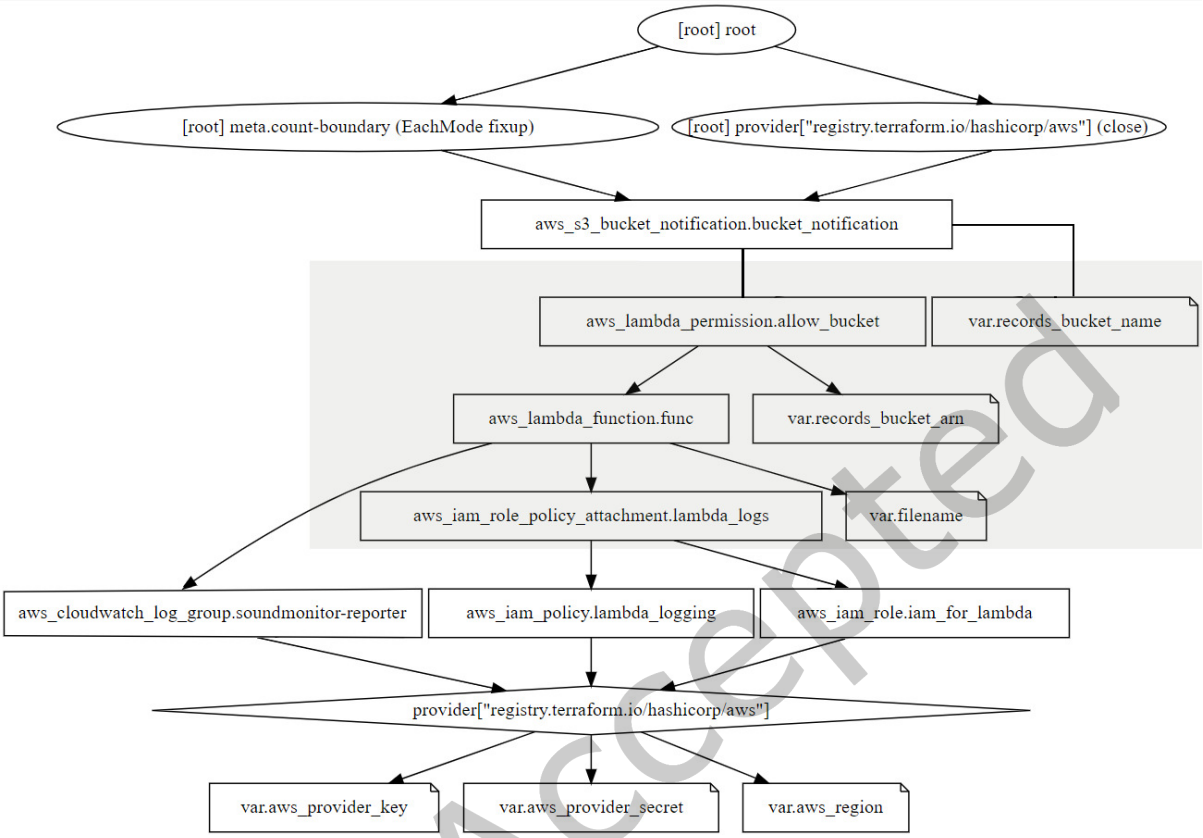

Fig. 8. Dependency graph of the lambda producer

6.3 Integration of terraform and CircleCI

On NMS, terraform runs in the deploy stage (Figure 6). On CircleCI a job is defined using a terraform image “hashicorp/terraform".

Algorithm 11 shows the job configuration, the image is built with terraform to initialize and deploy the infrastructure.

7 MONITORING

Monitoring tools allow to gather and process data about the system’s health and behavior of services [23]. It’s extremely important to avoid interruption of the services and improve the product overall quality by generating feedback and support plans according with the monitoring data. Monitoring results are not just for the operational team, it’s usable the developing team too. NMS uses CloudWatch with information provided by the services (Logs) to keep track of the health of the system, CloudWatch allows to verify system health with logs and graphs, some values which can be verified on CloudWatch and the service itself are:

- Number of active instances.
- Number of tasks active.
- Number of services active.

- Logs generated internally by services.
- Result of execution and termination of instances.

On ECS services and Lambda functions CloudWatch is deployed by Terraform, here with the code:

```
resource "aws_cloudwatch_log_group"
"monitor-adapa" {name = "/ecs/monitor-adapa"}
```

can be observed a log group, which is a set of logs printed by the service, on NMS each amazon service has one.

8 CLEAN CODE WITH SONARCLOUD

SonarCloud is a platform for code analysis which detects code quality issues strengthening the maintainability, reliability and security of the code. Accord with the official web page SonarCloud "uses state-of-the-art techniques in static code analysis to find problems, and potential problems, in the code " [33]. This platform shows the overall status of the repository with code smells, security hotspots, bugs, and vulnerabilities.

SonarCloud has a main interface with the general status of the repository and the quantity of each kind of problem which is showed in Figure 9. On the other hand, it is possible to see each problem and its position in the code by means of the code interface (Figure 10), it also indicates the reason why should be fixed and sometimes ways to fix it. Finally, these two interfaces reduce the time to implement fixes and strengthen better practices making the code on NMS cleaner.

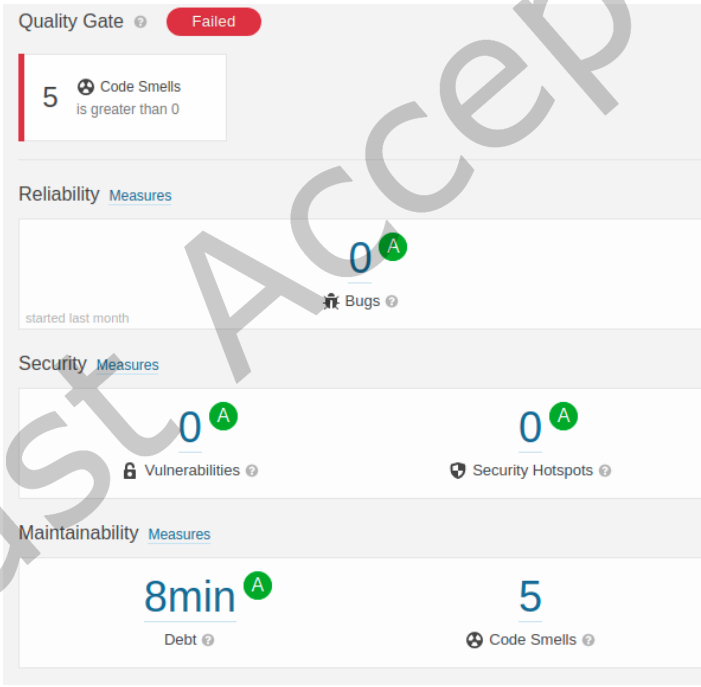

Fig. 9. Project interface of SonarCloud

9 LIMITATIONS OF MANUAL DEPLOYMENT AND INTEGRATION ON NMS

If there is not automation of pipelines, quality and efficiency of the software product can be affected. Thus, limitations when there is not an automation are mentioned:

Fig. 10. Code inteface of SonarCloud

- There is not traceability between the steps of the deployment.
- If the environment is not ready in the local machine, AWS CLI and Docker must be installed and configured.
- Steps like Unit testing and building could be omitted before merging to QA or other branches.
- It could be required the creation of scripts files or review documentation when commands are forgotten.
- Environment variables must be set locally and in the AWS console.
- Performance of deployment depends upon the capacity of the local machine.
- Image is built on local machine and then uploaded (When using mobile data it can cause expensive costs), TAGs must be set manually.
- Task definitions and services must be created or updated manually.

Finally, Table 1 shows the time it takes a series of steps with no automation (Tested on a CPU Intel I5-9400f, 16GB RAM, 496GB SSD, 110 Mbps download speed, 10Mbps upload speed and Ubuntu 20.04).

Table 1. Deployment steps without automation (minutes)

Step	Duration (minutes)
AWS CLI and Docker install	6.11
Update service and task definition	0.53
Build image and upload to ECR -inference-	0.53
Build image and upload to ECR -logstash-	0.53
Zip and update lambda function	3.25

10 RESULTS

Once the aforementioned developments have been carried out, DevOPS application is presented below:

- The integrity and quality of the NMS grows by using Continuous integration and pipelines. Unit testing keeps software behavior untouched so refactor does not affect the functionality and integration (QA and Master branches) is only accepted when tests and sonarCloud finish well.
- Pipelines allow automate deployment and integration keeping a track of the steps and status of all components on the NMS. There is no need to create environments for deployment or build images on local machines. This reduces the risk that a resource is not deployed or fails and allows faster changes in addition to focusing on other issues while the pipeline completes.
- IaC with terraform allows to deploy and keep a state of the infrastructure on AWS, so NMS can execute fast deployments.
- With sonarCloud code is kept clean by detecting bugs and repeated code also it shows tips to keep standards when writing code, the last improves the maintainability of the code.
- Using contexts (shared environment variables) allows NMS to define shared resources, so new components can be integrated and associate them to the shared infrastructure. They also reduce the time to implement new pipelines on NMS because there is no need to define again environment variables.
- Kibana offers the possibility of using its own query language (Kibana Query Language), through this the user can filter the stored information to obtain details about a noise event such as, for example, location, type, sound pressure level, among others as shown in Figure 11. It is worth mentioning that some of the values shown below are based on simulations because the physical infrastructure of the stations is currently under development. Likewise, it offers the possibility of visualizing in the graphic interface different variables including noise level in decibels captured by the stations which are in implementation phase. Similarly, the classification of acoustic sources for audio recordings can be provided to the user by means of statistical graphs. The results of some simulations can be seen in Figure 12.
- DevOps practices on NMS prepare it to a future integration of new components and new teams.

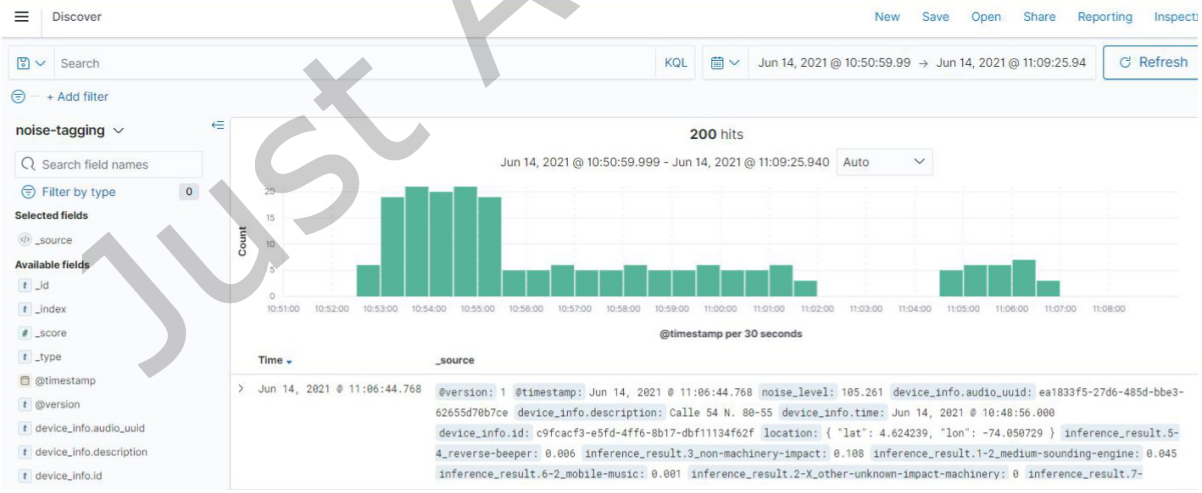

Fig. 11. Kibana query result

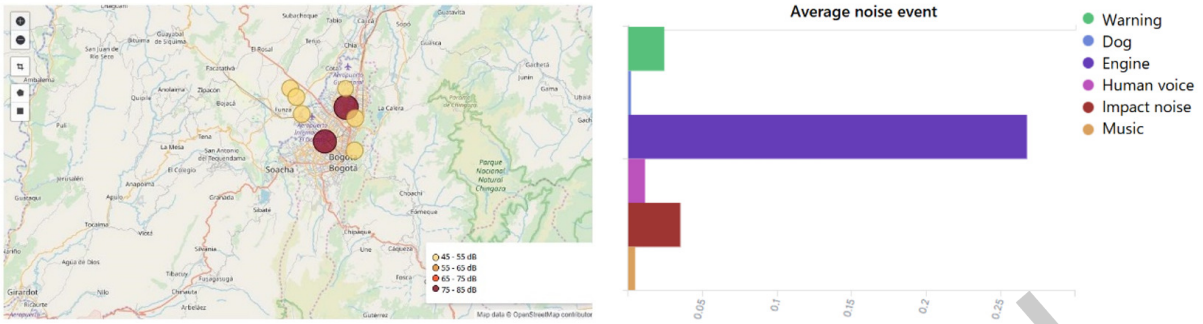

Fig. 12. Noise data simulated on map and inference result for an audio recording

11 DISCUSSION OF RESULTS

As part of the main contributions of this research, it is possible to mention the automation in the loading and processing of information for the characterization of an urban sound environment, where not only noise information is obtained through acoustic and spatial indicators, but also from the main sources of noise present in the place using deep learning. This automation will allow the operation of a noise monitoring network that will be implemented in the city of BogotÃą (Colombia). In the same vein, the development of Noise Monitoring System under the specified constraints is performed and the practice of DevOps is positively experienced, and SonarCloud has shown to be an effective clean coding mechanism, bringing to the system scalability and integration capabilities along with software quality issues.

Among the main advantages of the proposed infrastructure, several aspects can be highlighted, among which they stand out. First, the event-based architecture style allows subscribing to more services that process audio data from stations in parallel and at their own pace without the need for an orchestrator. This architecture additionally allows the total decoupling of services. Second, each audio data processing service runs in separate instances or containers, so failure of one would have no effect on the others. Third, microservices allow you to add new algorithms without modifying source code associated with other services. Finally, pipelines allow more people to contribute to the system, without having to know the deployment operations, they only require knowledge of how to integrate the code.

Summarizing, DevOps practices in a project involving Network Monitoring of environmental variables such as noise is a primary necessity. However, new challenges and new perspectives arise under the scope of the present study such as: (1) Portability of systems, (2) Energetic constrains of the overall systems, (3) Real-time operation and processing of acoustic variables and image (or even video) sequences, and (4) security of the system.

12 CONCLUSION

In this paper implementation of some DevOps practices on NMS through pipelines (CircleCI), clean coding (SonarCloud), IAC (Terraform), and monitoring (CloudWatch) brings for this system the potential to scale and integrate more components, besides to start a culture of continuous improvement in strengthen, maintainability, and software quality. Aditionally, NMS is a system which keeps growing in terms of infrastructure, so managing all of them manually gets risky and time consuming, for this reason, automated pipelines are the most important implementation for this system. They allow to deploy and monitor the status over all the resources of NMS, also testing and clean coding can be verified during pipelines to keep the integrity on the main branches and the software itself. Inside a culture of continuous improvement many practices and resources can be integrated in

the software cycle. On NMS future work will focused on: integration testing, black box testing, test coverage with SonarCloud, customized orbs or actions (the strategy of Github) for NMS deployments, and local linting.

REFERENCES

[1] Sainath Adapa. 2019. Urban Sound Tagging using Convolutional Neural Networks. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE2019)*. New York University, 5–9. https://doi.org/10.33682/8axe-9243

[2] Prashant Agrawal and Neelam Rawat. 2019. Devops, A New Approach To Cloud Development amp; Testing. In *2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, Vol. 1. 1–4. https://doi.org/10.1109/ICICT46931.2019.8977662

[3] Oscar Esneider Acosta Agudelo, Carlos Enrique Montenegro Marín, and Rubén González Crespo. 2021. Sound measurement and automatic vehicle classification and counting applied to road traffic noise characterization. *Soft Computing* (apr 2021). https://doi.org/10.1007/s00500-021-05766-6

[4] Amazon. 2021. Lambda. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[5] Apache Kafka. 2021. Kafka 2.8 Documentation. https://kafka.apache.org/documentation/

[6] Matej Artac, Tadej Borovssak, Elisabetta Di Nitto, Michele Guerriero, and Damian Andrew Tamburri. 2017. DevOps: Introducing Infrastructure-as-Code. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 497–498. https://doi.org/10.1109/ICSE-C.2017.162

[7] Sivadi Balakrishna, M Thirumaran, Vijender Kumar Solanki, Edward Rolando Núñez Valdéz, et al. 2020. Incremental hierarchical clustering driven automatic annotations for unifying IoT streaming data. *International Journal Of Interactive Multimedia And Artificial Intelligence* (2020).

[8] Sujeet Bheri and SaiKeerthana Vummenthala. 2019. An Introduction to the DevOps Tool Related Challenges.

[9] David Chapman. 2014. Introduction to DevOps on AWS. *Amazon Web Services* (2014).

[10] CircleCI. 2021. CircleCI Documentation. https://circleci.com/docs/

[11] Stefania Costantini, Giovanni De Gasperis, and Lorenzo De Lauretis. 2021. An Application of Declarative Languages in Distributed Architectures: ASP and DALI Microservices. *International Journal of Interactive Multimedia & Artificial Intelligence* 6, 5 (2021).

[12] Dcase. 2019. Urban Sound Tagging. http://dcase.community/challenge2019/task-urban-sound-tagging

[13] Elastic. 2021. Elastic Stack. https://www.elastic.co/es/elastic-stack

[14] Luca Filipponi, Silvia Santini, and Andrea Vitaletti. 2008. Data collection in wireless sensor networks for noise pollution monitoring. In *International Conference on Distributed Computing in Sensor Systems*. Springer, 492–497.

[15] Brian Fitzgerald and Klaas-Jan Stol. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software* 123 (2017), 176–189.

[16] Michael Hart and John Burke. 2020. AN EXPLORATORY STUDY ON THE DEVOPS IT ALIGNMENT MODEL. *Interdisciplinary Journal of Information, Knowledge & Management* 15 (2020).

[17] Luis Hermida and Ignacio Pavón. 2019. Spatial aspects in urban soundscapes: Binaural parameters application in the study of soundscapes from Bogotá-Colombia and Brasília-Brazil. *Applied Acoustics* 145 (2019), 420–430.

[18] IBM. 2021. Object Storage. https://www.ibm.com/cloud/learn/object-storage

[19] KafkaJS. 2021. KafkaJS Documentation. https://kafka.apache.org/20/documentation.html

[20] KafkaJS. 2021. Producing Messages · KafkaJS. https://kafka.js.org/docs/producing

[21] Ilkka Kivelä, Chao Gao, Jari Luomala, and Ismo Hakala. 2011. Design of noise measurement sensor network: Networking and communication part. In *Proceedings of the 5th International Conference on Sensor Technologies and Applications, Côte dâĂŹAzur, France*. 21–27.

[22] Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. 2018. Building a collaborative culture. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Vol. abs/1809.0. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/3239235.3240299 arXiv:1809.05415

[23] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. 2015. Dimensions of DevOps. In *Lecture Notes in Business Information Processing*. Vol. 212. 212–217. https://doi.org/10.1007/978-3-319-18612-2_19

[24] Ruth W Macarthy and Julian M Bass. 2020. An Empirical Taxonomy of DevOps in Practice. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 221–228. https://doi.org/10.1109/SEAA51224.2020.00046

[25] Panu Maijala, Zhao Shuyang, Toni Heittola, and Tuomas Virtanen. 2018. Environmental noise monitoring using source classification in sensors. *Applied Acoustics* 129 (2018), 258–267. https://doi.org/10.1016/j.apacoust.2017.08.006

[26] Panu Maijala, Zhao Shuyang, Toni Heittola, and Tuomas Virtanen. 2018. Environmental noise monitoring using source classification in sensors. *Applied Acoustics* 129 (2018), 258–267.

[27] José Montoya-Belmonte and Juan M Navarro. 2020. Long-Term Temporal Analysis of Psychoacoustic Parameters of the Acoustic Environment in a University Campus Using a Wireless Acoustic Sensor Network. *Sustainability* 12, 18 (2020), 7406.

[28] Charlie Mydlarz, Justin Salamon, and Juan Pablo Bello. 2017. The implementation of low-cost urban acoustic monitoring devices. *Applied Acoustics* 117, August 2015 (2017), 207–218. https://doi.org/10.1016/j.apacoust.2016.06.010
[29] Antonio Pita, Francisco J Rodriguez, and Juan M Navarro. 2021. Cluster Analysis of Urban Acoustic Environments on Barcelona Sensor Network Data. *International Journal of Environmental Research and Public Health* 18, 16 (2021), 8271.
[30] S Samundiswary and Nilma M Dongre. 2017. Object storage architecture in cloud for unstructured data. In *2017 International Conference on Inventive Systems and Control (ICISC)*. IEEE, 1–6. https://doi.org/10.1109/ICISC.2017.8068716
[31] Silvia Santini, Benedikt Ostermaier, and Andrea Vitaletti. 2008. First experiences using wireless sensor networks for noise pollution monitoring. In *Proceedings of the workshop on Real-world wireless sensor networks*. 61–65.
[32] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access* 5 (2017), 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629
[33] SonarCloud. 2021. Documentation. https://sonarcloud.io/documentation
[34] Mitesh Soni. 2015. End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery. In *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. IEEE, 85–89. https://doi.org/10.1109/CCEM.2015.29
[35] TensorFlow Hub. 2021. Yamnet. https://github.com/tensorflow/models/tree/master/research/audioset/yamnet
[36] Terraform. 2021. Write, Plan, Apply. https://www.terraform.io/
[37] Chunyang Xu and Jian Kang. 2019. Soundscape evaluation: Binaural or monaural? *The Journal of the Acoustical Society of America* 145, 5 (2019), 3208–3217.
[38] Chunyang Xu, Tin Oberman, Francesco Aletta, Huan Tong, and Jian Kang. 2021. Ecological Validity of Immersive Virtual Reality (IVR) Techniques for the Perception of Urban Sound Environments. In *Acoustics*, Vol. 3. Multidisciplinary Digital Publishing Institute, 11–24.

APPENDIX: ALGORITHMS

Here main programming codes are presented, however as the infrastructure is in continuous development, authors can be contacted in order to obtain more code details.

ALGORITHM 1: Algorithm section taken from the configuration of a job for unit testing

```
jobs:
    test:
        docker:
            - image: circleci/python:3.6
        steps:
            - checkout
            - run:
                command: comand to install requirements
                name: Install requirements
            - run:
                command: comand to run tests
                name: Running unit tests
```

ALGORITHM 2: An example from a workflow configuration

```
workflows:
    main:
        jobs:
            - test
            - build:
                requires:
                    - test
            deploy:
                requires:
                    - build
                filters:
                    branches:
                        only:
                            - qa
                            - stage
                            - master
```

ALGORITHM 3: Configuration example with an Orb

```
orbs:
    aws-ecr: circleci/aws-ecr@6.12.2

workflows:
    main:
        jobs:
            - test
            - aws-ecr/build-and-push-image:
                filters:
                branches:
                    only:
                        - master
                        - qa
                requires:
                    - test
                repo: "${AWS_RESOURCE_NAME_PREFIX}"
                tag: "${CIRCLE_SHA1}"
```

ALGORITHM 4: Inference service workflow on the configuration file

```
workflows:
    main:
        jobs:
            - sonar
            - test
            - aws-ecr/build-and-push-image:
            context: AWSContext
            filters:
                branches:
                    only:
                        - master
                        - qa
            requires:
                - test
                - sonar
            repo: "${AWS_RESOURCE_NAME_PREFIX}"
            tag: "${CIRCLE_SHA1}"
        - infrastructure-providing-and-deploy:
            context:
                - AWSContext
                - TerraformContext
            requires:
                - aws-ecr/build-and-push-image
```

ALGORITHM 5: Build step of an AWS Lambda

```
jobs:
    dependencies-zip-generation:
        docker:
            - image: node:12.20.1-stretch-slim
        steps:
            - checkout
            - run:
                name: install node modules
                command: npm install
            - run:
                name: Install zip
                command: |
                    apt-get update
                    apt-get install zip -y
            - run:
                name: zip and save source code
                command: zip -r function.zip index.js node_modules
            - persist_to_workspace:
                root: ./
                paths:
                    - function.zip
```

ALGORITHM 6: Deploy step of a AWS Lambda

```
deploy-lambda:
    docker:
        - image: hashicorp/terraform:light
    steps:
        - checkout
        - attach_workspace:
            at: ./terraform
        - run:
            name: Terraform init
            command: |
                cd terraform &&
                terraform init
        - run:
            name: Terraform provide and deploy
            command: |
                cd terraform &&
                terraform apply -auto-approve
```

ALGORITHM 7: AWS as provider on terraform configuration

```
provider "aws" {
 region = var.aws_region
 access_key = var.aws_provider_key
 secret_key = var.aws_provider_secret
}
```

ALGORITHM 8: Backend definition on terraform

```
terraform {
 backend "s3" {
   bucket  = "terraform-monitor-provide-states-files"
   key     = "s3event.tfstate"
   region  = "us-east-1"
 }
}
```

ALGORITHM 9: Definition of a cluster and a service on terraform on NMS

```
resource "aws_ecs_service" "main" {
 name            = var.service-name
 cluster         = aws_ecs_cluster.main.id
 task_definition = aws_ecs_task_definition.main.arn
 launch_type     = "FARGATE"
 desired_count   = 1

 lifecycle {
   ignore_changes = [
     desired_count]
 }

 network_configuration {
   subnets          = [var.sound-monitor-subnet]
   assign_public_ip = true
 }
}
```

ALGORITHM 10: Variables on terrafor

```
variable "service-name" {
 default = "monitor-adapa-service"
}

variable "aws_region" {
 description = "AWS Region to deploy"
 default = "us-east-1"
}

variable "aws_provider_key" {
 description = "AWS key for deploy and infrastructure providing"
}

variable "aws_provider_secret" {
 description = "AWS secret for deploy and infrastructure providing"
```

ALGORITHM 11: CircleCI job configuration using terraform

```
infrastructure-providing-and-deploy:
 docker:
   - image: hashicorp/terraform:light
 steps:
   - checkout
   - run:
       name: Terraform init
       command: |
         cd terraform &&
         terraform init
   - run:
       name: Terraform provide and deploy
       command: |
         cd terraform &&
         terraform apply -auto-approve -var="ecr_image_tag=${CIRCLE_SHA1}"
```