# Cloud-based web application for multivariate time series analysis

A language-agnostic integrative architecture for short- and long-running machine learning algorithms

Vasco Candeias
Instituto Superior Técnico
vascocandeias@tecnico.ulisboa.pt

***Abstract*—The surge of multivariate time series records in biomedicine has driven researchers to develop algorithms towards their clear interpretation. Dynamic Bayesian networks are among the most popular methods, allowing to transparently impute, classify, and make predictions on medical records while enabling users to understand the underlying assumptions. Although significant theoretical progress has been made on these methodologies, the development of related programs and their adoption is still far from fully accomplished. Indeed, these packages remain as resource-intensive command-line applications that analysts are reluctant to use. The lack of a public web application that integrates these methods is crucial in an era where every service is quickly moving to the Internet. In this thesis, this missing website is conceptualised and implemented as a cloud-based application that comfortably scales with the number of requests and can be easily extended with any scriptable software for data analysis. MAESTRO (dynaMic bAyESian neTwoRks Online), available at https://vascocandeias.github.io/maestro, relies on a microservice architecture deployed on Amazon Web Services and can handle the most demanding tasks, with the flexibility to readily increase processing power and reduce execution times. This application's scalability is proven by making thousands of simultaneous calls, which does not cause any performance degradation. Its versatility is further conveyed through a case study with real data. Additionally, a local server to be used in an intranet, preventing data from leaving a managed network, is proposed and distributed at https://github.com/vascocandeias/maestro-backend. These tools should allow clinicians and investigators to effortlessly use state-of-the-art tools for longitudinal data analysis.**

***Index Terms*—cloud architecture, dynamic Bayesian networks, MAESTRO, multivariate time series, web application**

## I. Introduction

Despite the increased development of machine learning (ML) techniques for automatically diagnosing human diseases from electronic medical records (EMRs) [1], [2], there is still a lack of accessibility for the end-user. In a world where every service is moving towards the web and people are less willing to download and execute software on their devices, it is imperial to provide a web-based platform that integrates the state-of-the-art methods for issues such as classification and clustering of multivariate time series (MTS). This system would allow such analyses to be readily available for any practitioner and let them take full advantage of the increased availability of EMRs, which hold the patients' clinical evaluations, and hence originate MTS when collected over time.

Since doctors cannot just put their patients' health in the hands of black-box solutions [3], they need models that not only can handle problems with a vast number of attributes and extensive data sets but are also transparent and meet the required level of accountability of the medical field [4]. By resorting to probabilistic graphical models (PGMs), these experts may get interpretable tools that produce reliable predictions and facilitate efficient interpretation, allowing them to improve each inference over their patients' EMRs.

As follows, it is no surprise that dynamic Bayesian networks (DBNs), a subset of PGMs, are emerging in this field, being used for tasks such as detecting outliers in MTS [5], making predictions using these time series (TS) or grouping them into clusters [6]. It would thus be highly beneficial for the medical community to have a platform at their disposal that implements these already available packages in a way that allows understanding the underlying algorithms and assumptions.

To fill in this gap, we studied state-of-the-art implementations of web applications and developed MAESTRO (dynaMic bAyESian neTwoRks Online) [7]. This publicly available web tool aggregates imputation, outlier detection, clustering and inference tools based on DBNs as well as discretisation and visualisation capabilities that allow users to upload and analyse their MTS. The resulting architecture was constrained to the following requirements:

- Using free or low-cost solutions;
- Tolerating both short- and long-running tasks;
- Being extensible to new packages;
- Scaling horizontally and vertically;
- Assuring concurrency;
- Securing users' data;
- Notification the users upon completion;
- Handling errors gracefully;
- Ensuring transparency;
- Using orchestration tools.

Concurrency is particularly challenging when trying to serve exponentially complex algorithms to learn DBNs online. In fact, the one web application that makes use of these models for outlier detection – METEOR [8] – exhibits severe concurrency issues. When a user is training a network, the entire website becomes unavailable to others, even the landing page. As such, a crucial objective is ensuring that not only does the website continue to operate but that simultaneous requests will not influence one another.

Besides, while it is clear that other data-analysis software like SPSS [9] already exist, they often require licenses that might not be affordable and do not allow research or medical teams to add custom packages. Furthermore, even though some tools such as Weka [10] may seem to satisfy the requirements mentioned above for being open source and extensible, they still require users to download software and have limited extensibility by only accepting Java packages. Moreover, having to run the program locally is not just an inconvenience, as practitioners might not even have access to computers with the necessary hardware to rapidly make these computations and without sacrificing their performance on other vital tasks.

In contrast, MAESTRO is more than a freely available service. By downloading its source code, any medical or academic institution can have this modular microservice infrastructure running in their intranet by solely setting some environment variables and executing a single command. Moreover, adding custom packages has never been easier, as the developers only need to create two straightforward interface files and copy them, along with the executables, to the server.

To face the challenges mentioned above, MAESTRO relies on a microservice architecture with a gateway that receives the users' requests, a message queue where workers get tasks from and a data service to store input files and results. On top of these, an email service notifies analysts upon each task's completion, a log service tracks potential errors, and an authenticator ensures authorised access.

While an on-premises version of the web tool is also made available, the publicly available one – which is the focus of this thesis – is deployed in the cloud using Amazon Web Services (AWS) and tested for its concurrency and scalability. We will verify whether, by resorting to both serverless functions and dynamically allocating virtual machines (VMs), the system withstands bursts of thousands of simultaneous requests without any performance degradation. Likewise, we intend to prove that this implementation can vertically scale for improved computational power and reduced execution times.

In the following pages, we start by presenting MAESTRO's functionalities and implementation in Section II. In Section III we demonstrate the use of the web application by analysing real data and in Section IV we test the scalability of the cloud implementation. Finally, in Section V, we discuss the developed application, and draw some conclusions in Section VI, suggesting some future work that can be done to improve this tool in Section VII.

## II. MAESTRO
## (DYNAMIC BAYESIAN NETWORKS ONLINE)

In this section, we start by walking through every tool provided by this web application and later dive into MAESTRO's architecture.

### A. Functionalities

As already explained, this web tool encompasses many packages useful for data analysis. These follow the general pipeline depicted in Fig. 1 and will be presented in this section.

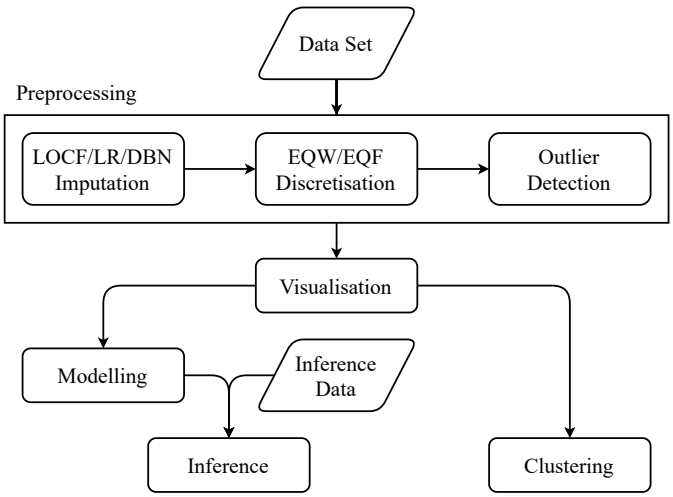
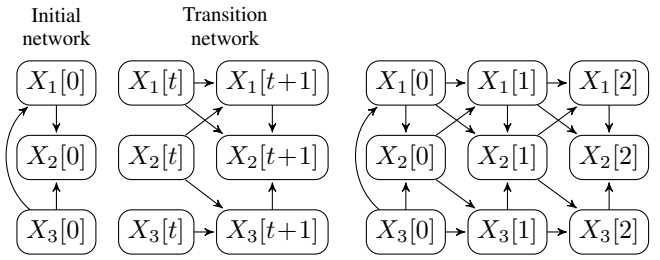



Fig. 1. Web tool pipeline for analysing data sets.

To better understand these tools, DBNs should be first explained, as most packages use them. We focus this discussion on temporal discrete-time models with discrete variables.

DBNs extend standard Bayesian networks (BNs) [11] to deal with time, allowing them to model MTS. They are composed of an initial network, representing the initial state of the temporal process, and a set of transition networks that say how the system evolves from time $t$ to $t+1$. When this system's dynamics do not change over time, the model is called stationary, and only one transition network is used. Moreover, the number of time slices used in this network is called the Markov lag.

A simple DBN is depicted in Fig. 2a, making it clear that both the initial and transition networks are straight BNs, where time-dependencies must flow forward in time. The transition networks' dependencies account for the interaction among multivariate trajectories in time, and conditional probability tables reveal their strength.




(a) Stationary DBN. (b) Corresponding unrolled network.

Fig. 2. Illustration of a DBN with a Markov lag of one.

A useful way to understand a DBN is to unroll it, as illustrated in Fig. 2b. Unrolling means converting a DBN into its equivalent BN. Starting at the baseline $t = 0$, the values of the variables at $t = 0$ are used in the transition network to predict the most probable configuration at time $t = 1$. And then, from these, the ones at time $t = 2$, and so on.

To learn these models, it is common to employ score-based algorithms, such as the tDBN [12], cDBN [13], bcDBN [14] and sdtDBN [15]. These are used by the functionalities detailed below, which apply either minimum description length (MDL) or log-likelihood (LL) scoring criteria.

*1) Imputation:* Since most method implementations assume the MTS are complete, the given data either has to have no missing values, or these should be imputed using one of the following methods:

- **Linear regression (LR)** – generates a linear interpolation using the available values in the TS and assigns the missing values accordingly;
- **Last observation carried forward (LOCF)** – copies the most recent state prior to the missing one. If there is none, the next value is used;
- **learnDBN** [16] – tries to learn a tDBN, cDBN or bcDBN despite the missing values and then uses it for imputation.

When the input is continuous, the first two methods are available, with the last two being provided otherwise.

*2) Discretisation:* After correcting for missing data, it is paramount to discretise it as the proposed models are only suited for this sort of input, which can be done by either one of the following:

- **Equal frequency (EQF)** – creates bins containing approximately the same number of data points;
- **Equal width (EQW)** – splits the range of values into a fixed number of bins and distributes the data respectively.

*3) Outlier detection:* The final step of preprocessing the input is to detect its outliers, that is, both the transitions (from time-slice $t$ to $t+1$) and the subjects (i.e., entire time series) that considerably deviate from the rest. This step replicates METEOR [5], which learns a tDBN and then scores each transition and TS. By setting a score threshold, it is then possible to classify them as being legitimate or outliers.

To continue to the next step, it is only possible to automatically filter out entire subjects, as transitions cannot be removed from the input. Regardless, the classifications and scores may be downloaded as comma-separated values (CSV) files.

*4) Visualisation:* Since a data set of multiple MTS has four dimensions (time, subject, value and attribute), we chose only to display a single feature per diagram, allowing users to compare subjects over the same metric. Moreover, since this heatmap is coloured according to limited bins of values, continuous TS have to be discretised solely for representation. This is done using EQW using nine bins, and the original values are still presented by hovering over the heatmap.

*5) Modelling:* Having preprocessed the file, it is possible to model an sdtDBN using the `sdtDBN` package [15], which accepts static features (besides temporal ones) and prior knowledge, i.e., constraints on which attribute relations are mandatory or forbidden.

*6) Inference:* Using the already learnt model, new observation files and another functionality of the `sdtDBN` package, it is possible to either predict how the new time series will progress until a certain time point or infer a given attribute in specific time slices.

*7) Clustering:* Using the program developed by Arcadinho [17], it is possible to group subjects by clusters. To determine them, the author proposes finding the $k$ DBNs which best describe $k$ clusters of subjects in the data, and then classify them accordingly. Besides downloading the clustering results, the user may analyse the learnt networks – which can be tDBNs, cDBNs or bcDBNs – to understand the model that could generate each cluster.

### *B. System architecture*

MAESTRO was developed as a client-server model, where the website is dynamically rendered in the client using Angular and only makes representational state transfer (REST) [18] calls to the back-end, with the server never having to render the web pages. This separation of concerns minimises network congestion, as the size of the requests is significantly reduced.

This application was developed for deployment both in the cloud, where it is publicly available, and in a local server, for organisations that must keep data on-premises or extend the website's packages. Although these architectures may differ in some aspects, they both rely on two major architectural patterns: microservices and competing consumers.

A microservice architecture contrasts with a monolithic one and consists in separating the services according to their concerns. For instance, instead of having one service containing the primary process, file storage and email service, these are split into three services. As for the communication pattern, the competing consumers (or task queue) satisfy the need for asynchronous execution. Since some of the proposed packages have an exponential time complexity, the user cannot wait for the result to be sent as a reply to the original REST request. Instead, the gateway places the task in a message queue, and a background process – the worker – retrieves it and runs the analysis, updating the results' table afterwards. In both implementations, the workers execute the command-line programs, recording their output and resulting files. The user may then monitor the task's completion by using the id returned by the gateway and, if this takes longer than 30 seconds, he will be notified by email upon completion or error.

*1) Cloud implementation:* MAESTRO's front-end was deployed on GitHub Pages [19] and its back-end on Amazon Web Services (AWS), since this cloud provider has proven to have the lowest and most consistent average cold start latency for both VMs [20], [21] and serverless functions [22] while continually providing excellent performance [23]. The resulting implementation is schematised in Fig. 3.

To ensure the user can analyse data sets of any size, the files should be directly uploaded to the S3 [24] bucket, as API Gateway [25] requests have a maximum payload size of 10 MB. By using the AWS Amplify [26] framework for Angular, the front-end can log in to Cognito [27] and then access S3, which will automatically validate his permissions with AWS Identity and Access Management [28]. Using Cognito's JSON Web Token (JWT), the user can also make authenticated requests to API Gateway.

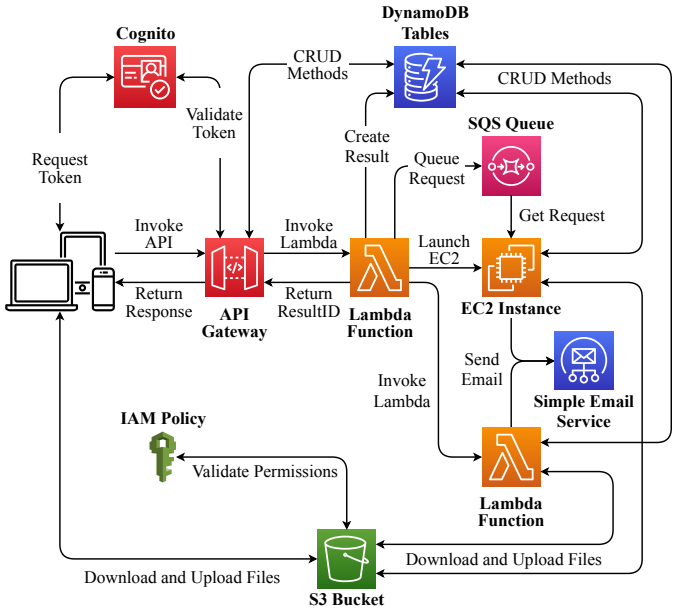



Fig. 3. Cloud back-end architecture.

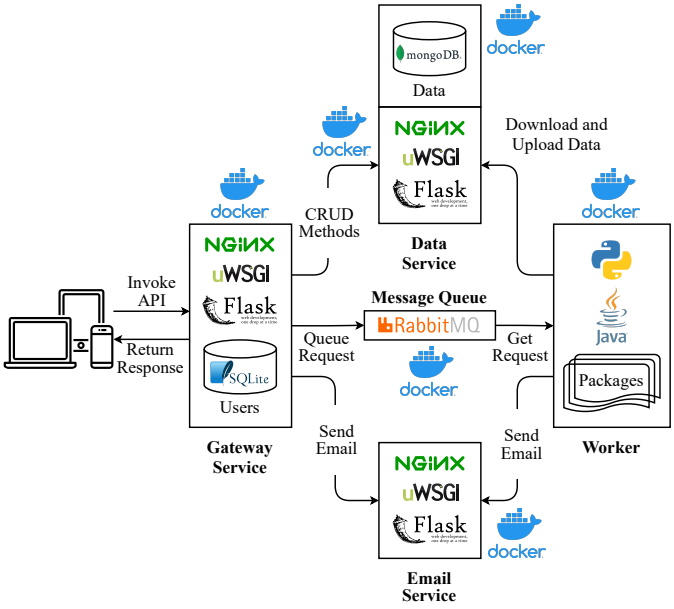



Fig. 4. Local back-end architecture.

As for handling the analysis requests, the gateway must launch a Lambda [29] function which will queue the message and then start a new Elastic Compute Cloud (EC2) [30] instance to consume it. This guarantees horizontal scalability, as a new instance is created for each request. However, this solution has a downside: cold start. While EC2 instances can run indefinitely, being perfect for long-running requests, they may take up to a minute to be ready to start processing [20], [21], which is not acceptable for tasks that would run in a few seconds locally. To solve this issue, another Lambda function is invoked in parallel, running the packages similarly to the EC2. Yet, even though these take considerably less time to start [31], they have a timeout and do not provide the same performance as EC2 VMs, meaning that they should merely handle the short-running requests.

*2) On-premises implementation:* For the intranet version of the application, the back-end was developed using Docker [32] containers orchestrated using Docker Compose [33], with the resulting network being presented in Fig. 4.

To implement the services, the traditional NGINX, uWSGI and Flask architecture [34], [35] was chosen, with all of them running on an Alpine Linux container, where NGINX is the web server and reverse proxy, uWSGI is the application server, and Flask is the application itself, where the logic written in Python lies. As for the gateway, NGINX also serves the Angular static files, to be fetched when the user first accesses the website, and its Flask program uses an SQLite [36] table to save the authentication information.

In this architecture, the number of workers no longer corresponds to the number of requests, but to the number of servers running these containers, which can be easily added to the network. This is not to say they can only handle one request at a time, as they launch a new thread for each package execution. And while, by default, the workers come with Java and Python installed, as the former is needed to run most of the packages and the latter to fetch messages from a RabbitMQ [37] work queue and execute the analyses, any other language can be included by adjusting this service's Dockerfile [38]. In addition, if there are no extra servers to add more workers when they exhibit signs of resource starvation, it is always possible to limit the number of messages each worker can process simultaneously, allowing them to process pending tasks before retrieving any more from the queue.

Moreover, for the database tables, the predominant NoSQL MongoDB [39] management system was adopted, due to its ease of use and remarkable performance [40]. As for the logs, similarly to the cloud implementation, a logging service was not designed, since Docker handles this centralised role itself. When running containers as part of Docker Compose, every service's logs are displayed on the screen and saved to a file.

Finally, although Docker Compose will run all the containers in the same server by default, they were developed to be distributed over a network. This means that it is possible to spin up as many containers as necessary in different hosts and have each service deployed in an adequate server (for instance, the data service needs a lot of free disk space and fast I/O operations while the email only needs minimal resources). Furthermore, this orchestration tool guarantees that services are restarted if they halt for any reason.

## III. Case Study

This section illustrates the web tool's usage with real data extracted from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database [41]. The ADNIMERGE data set consists of 13,413 time series with 113 attributes (including identification, dates and other codes) measured from 1,973

patients from multiple case report forms – such as Alzheimer's Disease Assessment Scale (ADAS) or Rey's Auditory Verbal Learning Test (RAVLT) – and biomarker lab summaries across the ADNI protocols. Before uploading, we selected the patients that had examinations after 24 months, as suggested in previous studies [42]. Afterwards, we merged duplicates and inserted empty rows for the missing observations, which allowed to compute the percentage of missing values for each feature. By removing every attribute where these exceeded $20\,\%$, we picked the 13 dynamic features presented in Table I, consequently reducing the number of patients to $1{,}286$.

TABLE I
SELECTED CONTINUOUS FEATURES AND THEIR RANGES, AVERAGES (A), MEDIANS (MDN) AND PERCENTAGES OF MISSING VALUES (%M).

| Feature | Range | A | Mdn | %m |
|---|---|---|---|---|
| CDRSB[a] | $[0, 17]$ | 1.8 | 1 | 16.8 |
| ADAS11[b] | $[0, 70]$ | 10.6 | 9 | 16.5 |
| ADAS13[c] | $[0, 71.3]$ | 16.5 | 14.3 | 17.3 |
| ADASQ4[d] | $[0, 10]$ | 5.1 | 5 | 16.5 |
| MMSE[e] | $[1, 30]$ | 27.0 | 28 | 16.4 |
| RAVLT_immediate[f] | $[0, 75]$ | 35.0 | 34 | 16.8 |
| RAVLT_learning[g] | $[-5, 14]$ | 4.1 | 4 | 16.8 |
| RAVLT_forgetting[h] | $[-9, 15]$ | 4.3 | 4 | 17.0 |
| RAVLT_perc_forgetting[i] | $[-450, 100]$ | 58.9 | 60 | 17.5 |
| TRABSCOR[j] | $[0, 996]$ | 119.5 | 89 | 18.7 |
| FAQ[k] | $[0, 30]$ | 4.7 | 1 | 16.6 |
| mPACCtrailsB[l] | $[-39.7, 12.9]$ | $-5.5$ | $-3.7$ | 16.4 |
| mPACCdigit[m] | $[-39.7, 7.1]$ | $-5.7$ | $-4.1$ | 16.4 |

[a]Clinical Dementia Rating Scale sum of boxes.
[b]ADAS (11 items).
[c]ADAS (13 items).
[d]ADAS with delayed word recall.
[e]Mini-Mental State Examination.
[f]Sum of five RAVLT trials.
[g]RAVLT trial 5 minus trial 1.
[h]RAVLT trial 5 minus delayed recall.
[i]RAVLT_forgetting divided by trial 5.
[j]Trail Making Test part B.
[k]Functional Activities Questionnaire.
[l]Modified Preclinical Alzheimer's Cognitive Composite (mPACC) with TRABSCOR.
[m]mPACC with Digit Symbol Substitution.

The visualisation of the MMSE is shown in Fig. 5 and allows us to see some patterns. In this case, it is clear that most patients do not have examinations for the fourth time point and usually tend towards high (blue) values.


Fig. 5. Visualising the patients' original MMSE results.

Since this data is continuous and has missing values, we are given the option of either imputing or discretising the time series, with the first one being restricted to LOCF and LR. Using the latter results in the data presented in Fig. 6. As expected, the missing values were filled using linear interpolation.


Fig. 6. Result of using linear regression to impute the data.

Subsequently, we discretise the data using EQF and three bins (representing low, medium and high values). For MMSE, the following bins are generated:

- a – $[1, 27]$;
- b – $[28, 29]$;
- c – $30$.

Doing so produces the data represented in Fig. 7.


Fig. 7. Result of discretising the data using EQF and three bins.

Following that, we proceed to clean the data by removing outliers. Through learning a tDBN using log-likelihood as the scoring criterion, a Markov lag of one and two parents – represented in Fig. 8 –, it is possible to plot the outlierness of each transition and subject in a histogram – Fig. 9.

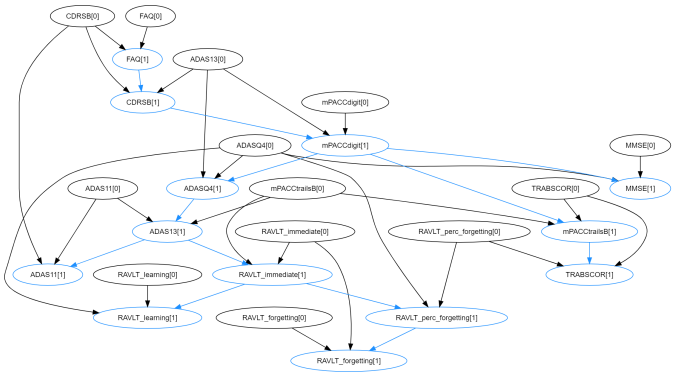



Fig. 8. Network learnt for outlier detection using log-likelihood as the scoring criterion, a Markov lag of one and two parents.

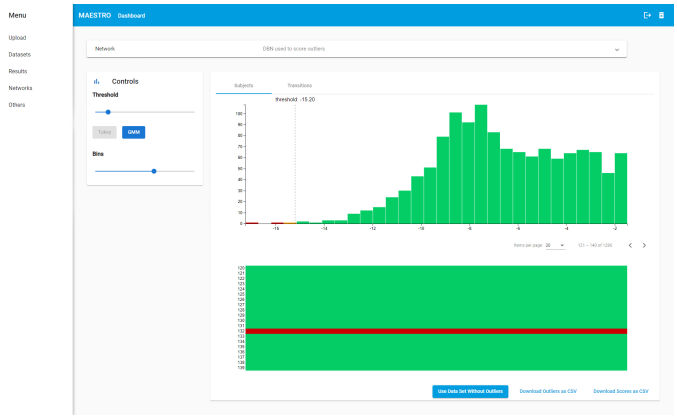



Fig. 9. Result of outlier detection using log-likelihood as the scoring criterion, a Markov lag of one and two parents. The threshold is set to Tukey's.

Having removed the three outlier subjects using Tukey's threshold, we now analyse the remaining patients, starting with training an sdtDBN model for inference. Since this method accepts static attributes, we selected the three variables described in Table II.

TABLE II
SELECTED STATIC FEATURES, THEIR CLASSES AND PERCENTAGES OF MISSING VALUES (%M).

| Feature | Classes | %m |
|---|---|---|
| PTGENDER | {Male, Female} | 0.0 |
| PTETHCAT[a] | {Not Hisp/Latino, Hisp/Latino} | 0.5 |
| PTMARRY[b] | {Married, Divorced, Widowed, Never married} | 0.3 |

[a] Race.
[b] Initial marital status.

Adding these observations, we may train the sdtDBN using a log-likelihood scoring criterion, a unitary Markov lag, one dynamic parent, two static ones, and no restrictions. However, knowing this package cannot process static files with subjects that do not exist in the dynamic input, we added a new one to induce an error. As seen in Fig. 10, this is gracefully caught and presented to the user.


Fig. 10. Error handling in the application.

When using the original observations, the result is successful. Again, this network may be viewed through an expandable card, where it is also possible to download it – Fig. 11.

Knowing the MMSE value for subject 6 at the seventh time point is 22 (falling into bin a according to this discretisation), we may test the model's prediction by feeding it this subject's observations and requesting the most probable inference. The model makes an accurate prediction, as can be seen in Fig. 12.

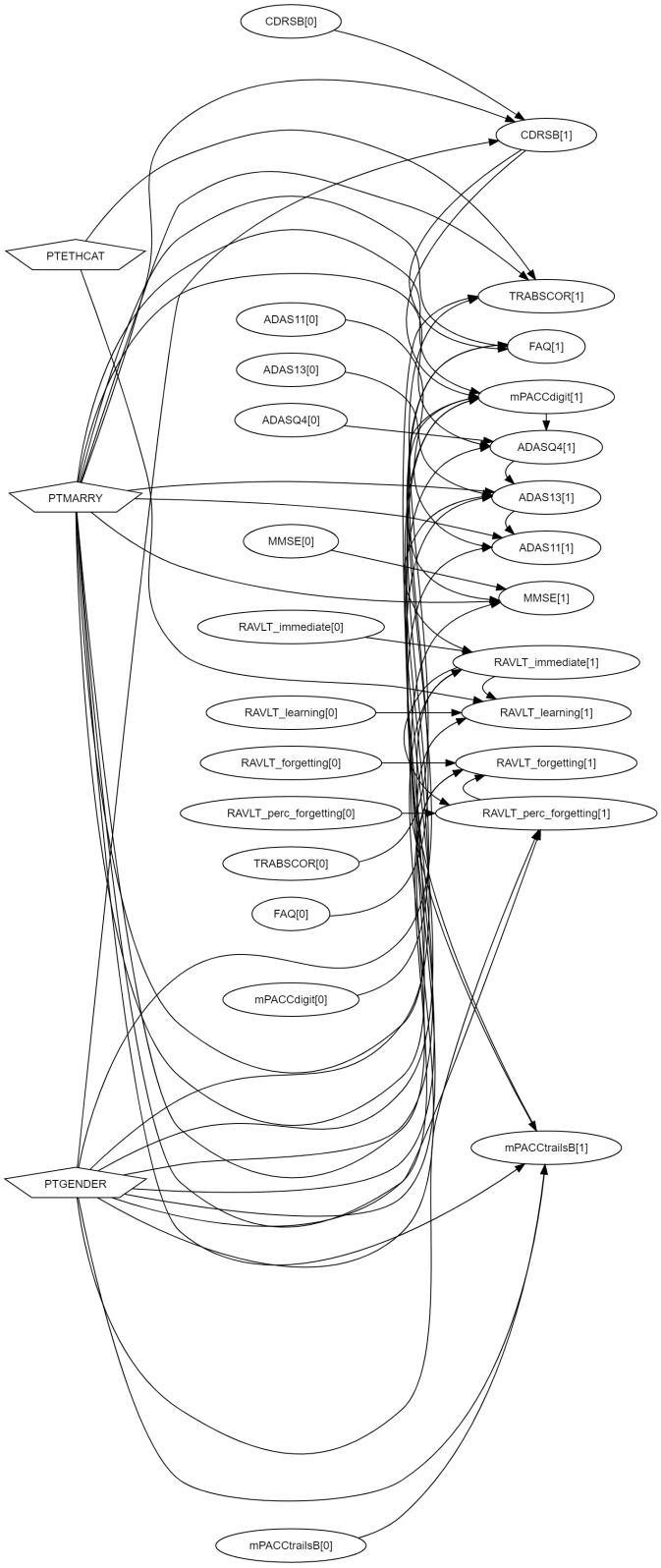



Fig. 11. DBN learnt using log-likelihood as the scoring criterion, a unitary Markov lag, one dynamic parent and two static ones.

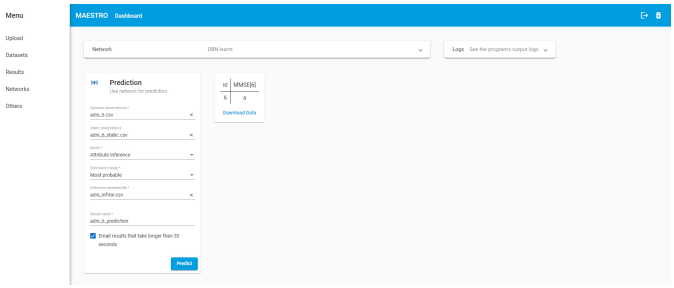



Fig. 12. Most probable MMSE for subject 6 at the seventh time point.

To further understand this result, we computed this attribute's distribution of probabilities for the requested subject and time point – Fig. 13. It is clear that the likelihood heavily inclines towards the prediction. However, there is some fluctuation on these probabilities across multiple requests, since the intermediate attributes are predicted using random sampling according to each node's probability distribution.


Fig. 13. MMSE's predicted probability distribution for subject 6 at the seventh time point.

Finally, we may group the TS into clusters. Since there are three possible diagnoses – clinically normal, mild cognitive impairment and dementia –, we try to fit three tDBNs using an intra-slice in-degree of one, two parents and a unitary Markov lag. After handling the request, MAESTRO presents the results as in Fig. 14. It is also possible to download both the network as text and the scores or clustering results as CSV files. Furthermore, every network representation of each cluster is presented in an individual tab.

As in every other network displayed by the application, hovering over a node will show its conditional probabilities table. Moreover, as in most of the other DBN-related functionalities, it is possible to view the application's logs, which correspond to the information that the command-line executable usually prints to the standard output.

This might lead to the assumption that errors are only caught when the packages quit successfully since the workers are retrieving the standard output. As such, it should be clarified that, even though the error shown in this section is a controlled one, any exception that causes the packages to quit unexpectedly is also caught. However, these may be too verbose if not appropriately handled by the program.

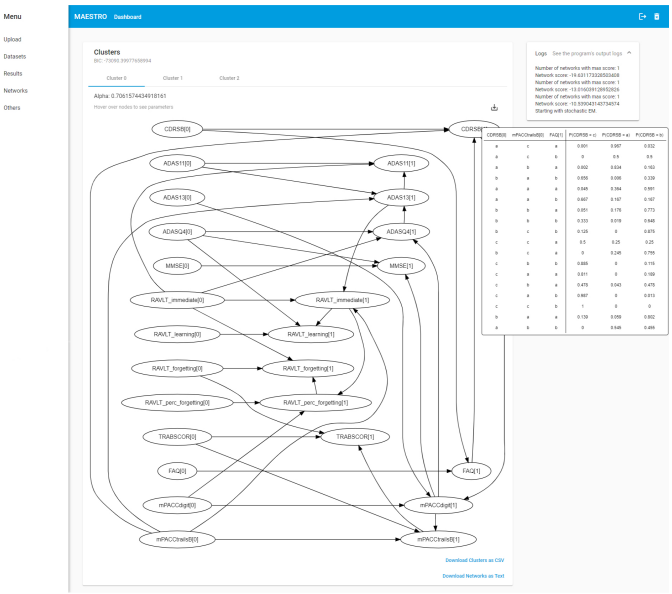



Fig. 14. Result of finding three clusters using tDBNs with a Markov lag of one, two parents and an intra-slice in-degree of one.

## IV. Stress Testing

### A. Vertical scalability

One of the fundamental requirements for this website was ensuring it was capable of handling every long-running analysis within an acceptable time frame, which is tightly related to the system's vertical scalability: if we are able to augment the processing nodes' computational resources, the execution time will decrease. To assess this metric in the cloud architecture, we chose the `learnDBM` package for its higher resource demand and used the `combinedDataset.csv`, available in the package's webpage [17] and consisting of 2,000 time series with five attributes and 10 time steps each, produced by two distinct DBNs.

The two networks were trained with a Markov lag of two, a unitary intra-slice in-degree, two parents, and multithreading. The test was conducted by launching three distinct EC2 instances – m5.large, m5.xlarge and m5.2xlarge –, establishing a connection to each of them, and executing a Python script which sequentially started and timed ten child processes that performed the aforementioned task. Since the purpose was to evaluate the hardware impact on execution time, we opted not to include any network-bound operations in the measurements and solely timed the package execution. The results are plotted in Fig. 15.

As expected, the most powerful VM – the m5.2xlarge with 8 virtual processors and 32 gibibytes of memory – was substantially faster than the one with the least resources – the m5.large with 2 virtual processors and 8 gibibytes of memory. The vertical scalability is thus warranted, since the developer may easily choose to specify a more robust instance to be used for subsequent requests whenever these are taking too long.

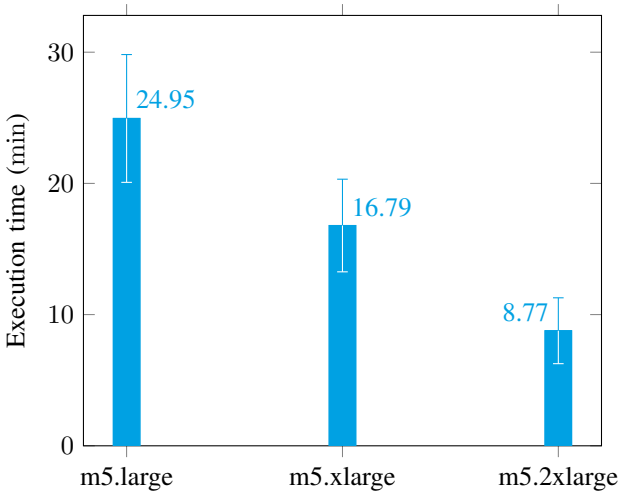



Fig. 15. Execution time when clustering the `combinedDataset.csv` for ten times in three instances with a Markov lag of two, a unitary intra-slice in-degree, two parents, two clusters and multithreading, using three distinct EC2 instance types.

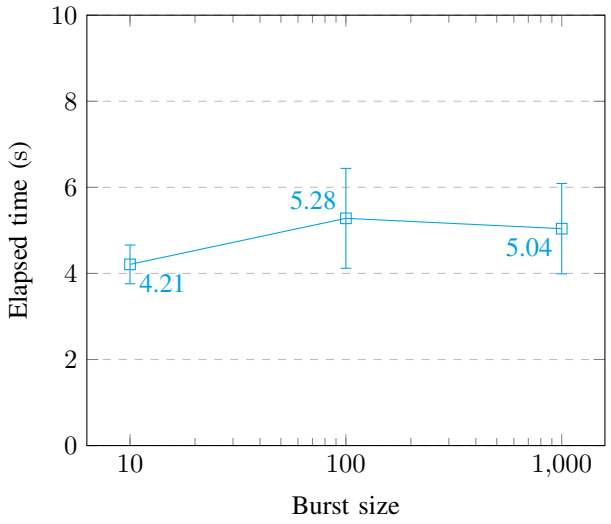



Fig. 16. Average elapsed time as a function of the burst size when clustering the `combinedDataset.csv` with a Markov lag of one, a unitary intra-slice in-degree, one parent and two clusters.

Furthermore, it is worth mentioning that using extra computing resources may not be significantly more expensive, since the instances are terminated after the requests complete, thus having no idle cost. For example, even if the m5.xlarge – with 4 virtual processors and 16 gibibytes of memory – costs $\$\,0.222$ per hour and the m5.2xlarge costs $\$\,0.444$ when hosted in London [43], their average price in this test would not be significantly different: while the former would take $\$\,0.062$ to execute the process in over 16 minutes, the latter would cost $\$\,0.065$ to accomplish it in almost half the time. As such, it might even be economically beneficial to invoke VMs that execute the tasks in a more timely manner: a win-win situation for the user and the organisation.

### *B. Horizontal scalability and concurrency*

Another major objective was scaling horizontally, which would allow the users to concurrently access the application and make analysis requests without performance degradation. Even though, for the cloud implementation, invoking a Lambda function and launching a new EC2 instance per request should suffice this constraint, we decided to conduct a test to assure this characteristic. This was accomplished by sending bursts of 10, 100 and 1,000 requests and measuring their execution times. The requests consisted in finding two clusters in the `combinedDataset.csv`, using a Markov lag of one, a unitary intra-slice in-degree and one parent. Fig. 16 graphically represents the results.

From this experiment, it is clear that the application maintains its performance despite the load and that the users will not notice whenever the servers are under pressure. As for the slightly lower duration of the smaller burst, it can be explained by the lower number of requests which is not enough to represent the amplitude of values: in fact, it falls inside the confidence interval of the remaining sizes.

## V. Discussion

After demonstrating and appropriately stress testing the application, we should go through the pre-established requirements and confirm their assurance. Since the development comprised both a cloud and a local version, let us first address the prerequisites that the two address equally, and later focus on those that differ between implementations.

First off, as Section III demonstrates, QoS is assured as the website is user friendly and extends the capabilities of the original packages with its visualisation tools, while still being transparent as the user has no perception of the multiple components involved and their interactions. What it does not depict is the notification and authentication services, both of which have been implemented, with the former sending an email upon either correct or unexpected completion of long tasks, and the latter resorting to the commonly used JWTs.

Additionally, the architecture was designed to effortlessly accommodate new packages through the inclusion of their source codes and two simple interface files per functionality, as long as they can be executed in a Linux system with Python and Java installed. If that is not the case, augmenting the environment is just as simple, as it solely requires updating a Dockerfile and rebuilding the worker's container. Besides, the users may rest assured that erroneous executions will not disturb the tool's correct execution, but will be caught by the workers and properly displayed to the practitioner, as exemplified in Section III through a deliberate error.

Diving into implementation-specific characteristics, we can state that two of the most critical predefined objectives – accepting data sets of any size and handling both short- and long-running tasks – are accomplished by executing the analyses asynchronously and, for the cloud architecture, by uploading directly to S3 and launching a Lambda and an EC2 instance in parallel.

Furthermore, since MAESTRO is focused on clinical data interpretation, security was also a fundamental concern. By distributing a local implementation where data never has to leave the premises, we give practitioners the option to improve confidentiality if they opt not to trust AWS security guarantees, which rely on AES-256 encryption [44]. Notwithstanding, it is up to each organisation to then implement HTTPS connections and secure the servers themselves from unwanted agents. However, this should not be a concern, given that the communication with the cloud back-end uses HTTPS and that the S3 storage, the DynamoDB tables and the EC2 instances are all encrypted, with every processing instance being isolated from the others. This last assertion is quite essential, as it means that in the eventuality of a hacker finding and accessing the physical server handling his request, he is still unable to access anyone else's data.

As for scalability, both of its dimensions are satisfied: the two solutions support not only adding more nodes by using a message queue and a microservices architecture but also effortlessly enhancing the existing ones by changing the instance type in the cloud architecture (as shown in Section IV-A) or merely upgrading the servers in the local version. Moreover, concurrency is achieved in the cloud, much like the horizontal scalability, by launching one Lambda function and one EC2 instance per processing request (without performance deterioration, as proven in Section IV-B). On-premises, it is up to the organisation to provide enough computing power, but since Docker Compose – an orchestration tool – was used, increasing the number of workers could not be easier, so assuring local concurrency is only a matter of having the right hardware. Besides processing, using NoSQL databases in both implementations further guarantees horizontal scalability, but now in the realm of data storage.

Finally, longevity may be evaluated by consulting web applications such as WIMP [45] or RISP [46]. When trying to access these tools' websites, the user will realise they are no longer available, demonstrating they have not withstood the test of time. While the local platform's longevity can only be assured by the organisations that adopt it, by using low-priced AWS services and free GitHub Pages instead of hosting our own cluster of servers, we are confident the publicly available version will remain online without much maintenance or restructuring.

## VI. Conclusion

In this dissertation, a language-agnostic microservice architecture for both short- and long-running web applications was devised so that practitioners no longer have to download and locally execute data analysis software. The proposed approach satisfies all of the prerequisites, including being secure, scalable and transparent while still managing to efficiently complete tasks of any duration.

This design can be adapted to incorporate any demanding data analysis command-line program and was adopted in this thesis to implement MAESTRO [7], an MTS interpretation tool that makes use of DBNs to impute data, detect outliers, cluster TS and predict their progression. By deploying this application using AWS and merging the use of serverless functions with auto-scaling VMs, the application showed positive results regarding both concurrency and scalability.

Besides the publicly available web tool, an on-premises implementation of the introduced architecture was also developed, and its source code was made available in a GitHub repository [47]. This version can be extended with any scriptable package and is aimed at any analyst or organisation that needs this flexibility or the added layer of security that only on-premises solutions can guarantee.

With this project, the lack of user-friendly applications that merge machine learning and biomedicine should be closer to fulfilment, as every practitioner and organisation can now both use state-of-the-art DBN-based data analysis software online, and effortlessly host their own tools.

## VII. Future Work

Bearing in mind the ease of vertically scaling the workers underlined in Section IV-A, it is possible to further improve the cloud implementation by choosing the EC2 instance type to launch according to the expected execution time of each incoming request. To accomplish this, the Lambda function that launches the container may consult a table that maps the input parameters and data size to an instance and the elapsed time. This table may be dynamically built by the workers when they conclude each task. Doing so will improve the application's QoS and optimise its costs.

With the advancements in homomorphic encryption, the packages could also be re-written to allow operating on top of encrypted data, which would require the front-end code to encrypt the data before uploading. However, this is only advised whenever this technology matures to the point where the added time complexity becomes negligible.

As for the on-premises implementation, the task queue may be replaced by another that allows distributing each request to the worker that is processing fewer messages. In fact, the fair dispatch used by RabbitMQ is only equitable to a certain degree: since some tasks are exponentially more demanding than the others, some workers might get all of the most expensive ones. Delivering each message to the worker with the smallest number of unacknowledged messages will mitigate this issue. However, in case some workers are considerably less resourceful than the rest, this load balancer would not be fair as well, as these would be starved out while the others are still capable of higher loads. Solving this is a matter of limiting the number of unacknowledged messages in the more limited servers.

The monitoring and logging service of the on-premises version also has room for improvement. Even if the Docker logs are parsable, they still lack the usage and performance metrics that tools such as Amazon CloudWatch combine in a user-friendly manner. There are multiple services – the ELK stack [48] or Prometheus [49] and Grafana [50] for instance – that can be easily integrated using Docker Compose, providing a dashboard that centralises the entire system's information.

Lastly, this private dashboard could also allow to more easily upload the files regarding new packages. In this way, the tool administrators would not have to access each worker nor use command-line instructions to update the Docker Compose file and update them. Creating a fan-out queue that carries the new packages (or instructions to remove old ones) to every worker or programmatically updating the Docker images and reloading the services are just two of the methods that can help to combine this feature.

## REFERENCES


[1] L. V. Ho, D. Ledbetter, M. Aczon, and R. Wetzel, "The dependence of machine learning on electronic medical record quality," in *AMIA Annual Symposium Proceedings*, vol. 2017. American Medical Informatics Association, 2017, p. 883.

[2] J. T. Schwartz, M. Gao, E. A. Geng, K. S. Mody, C. M. Mikhail, and S. K. Cho, "Applications of machine learning using electronic medical records in spine surgery," *Neurospine*, vol. 16, no. 4, p. 643, 2019.

[3] M. Ghassemi, L. A. Celi, and D. J. Stone, "State of the art review: the data revolution in critical care," *Critical Care*, vol. 19, no. 1, pp. 1–9, 2015.

[4] D. S. Watson, J. Krutzinna, I. N. Bruce, C. E. Griffiths, I. B. McInnes, M. R. Barnes, and L. Floridi, "Clinical applications of machine learning algorithms: beyond the black box," *Bmj*, vol. 364, 2019.

[5] J. L. Serras, "Outlier detection for multivariate time series," Master's thesis, Instituto Superior Técnico, 11 2018.

[6] S. Arcadinho, "Model-based Learning in Multivariate Time Series," Master's thesis, Instituto Superior Técnico, 11 2018.

[7] V. Candeias. MAESTRO. Accessed: 5 December 2020. [Online]. Available: https://vascocandeias.github.io/maestro/

[8] J. Serras. METEOR. Accessed: 6 December 2020. [Online]. Available: https://jorgeserras.shinyapps.io/outlierdetection/

[9] SPSS Software — IBM. Accessed: 6 December 2020. [Online]. Available: https://www.ibm.com/analytics/spss-statistics-software

[10] Weka 3 - Data Mining with Open Source Machine Learning Software in Java. Accessed: 6 December 2020. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/

[11] N. Friedman, K. Murphy, and S. Russell, "Learning the structure of dynamic probabilistic networks," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, p. 139–147.

[12] J. L. Monteiro, S. Vinga, and A. M. Carvalho, "Polynomial-time algorithm for learning optimal tree-augmented dynamic Bayesian networks," in *UAI*, 2015, pp. 622–631.

[13] M. Sousa and A. M. Carvalho, "Learning Consistent Tree-Augmented Dynamic Bayesian Networks," in *International Conference on Machine Learning, Optimization, and Data Science*. Springer, 2018, pp. 179–190.

[14] ——, "Polynomial-Time Algorithm for Learning Optimal BFS-Consistent Dynamic Bayesian Networks," *Entropy*, vol. 20, no. 4, p. 274, 2018.

[15] T. Leão. sdtDBN — tDBN inference and learning with static parents. Accessed: 6 December 2020. [Online]. Available: https://ttlion.github.io/sdtDBN/

[16] S. Arcadinho. learnDBN — Dynamic Bayesian Network learning. Accessed: 8 December 2020. [Online]. Available: https://ssamdav.github.io/learnDBN/

[17] ——. learnDBM — Dynamic Bayesian Multinet learning. Accessed: 8 December 2020. [Online]. Available: https://ssamdav.github.io/learnDBM/

[18] R. T. Fielding and R. N. Taylor, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[19] GitHub Pages — Websites for you and your projects, hosted directly from your GitHub repository. Just edit, push, and your changes are live. Accessed: 21 November 2020. [Online]. Available: https://pages.github.com/

[20] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, 2012, pp. 423–430.

[21] S. I. Abrita, M. Sarker, F. Abrar, and M. A. Adnan, "Benchmarking VM Startup Time in the Cloud," in *International Symposium on Benchmarking, Measuring and Optimization*. Springer, 2018, pp. 53–64.

[22] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *Proceedings of the 2018 USENIX Annual Technical Conference, USENIX ATC 2018*, 2018, pp. 133–145. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/wang-liang

[23] H. Martins, F. Araujo, and P. R. da Cunha, "Benchmarking Serverless Computing Platforms," *Journal of Grid Computing*, 2020.

[24] Cloud Object Storage — Store & Retrieve Data Anywhere — Amazon Simple Storage Service (S3). Accessed: 25 November 2020. [Online]. Available: https://aws.amazon.com/s3/

[25] Amazon API Gateway — API Management — Amazon Web Services. Accessed: 12 November 2020. [Online]. Available: https://aws.amazon.com/api-gateway/

[26] Build Mobile & Web Apps Fast — AWS Amplify — Amazon Web Services. Accessed: 12 November 2020. [Online]. Available: https://aws.amazon.com/amplify/

[27] Amazon Cognito - Simple and Secure User Sign Up & Sign In — Amazon Web Services (AWS). Accessed: 12 November 2020. [Online]. Available: https://aws.amazon.com/cognito/

[28] AWS Identity & Access Management - Amazon Web Services. Accessed: 20 November 2020. [Online]. Available: https://aws.amazon.com/iam/

[29] AWS Lambda – Serverless Compute - Amazon Web Services. Accessed: 12 November 2020. [Online]. Available: https://aws.amazon.com/lambda/

[30] Amazon EC2. Accessed: 12 November 2020. [Online]. Available: https://aws.amazon.com/ec2/

[31] I. Müller, R. F. Bruno, A. Klimovic, G. Alonso, J. Wilkes, and E. Sedlar, "Serverless Clusters: The Missing Piece for Interactive Batch Applications?" in *10th Workshop on Systems for Post-Moore Architectures (SPMA 2020)*, 2020. [Online]. Available: https://doi.org/10.3929/ethz-b-000405616

[32] T. James, *The Docker Book: Containerization Is the New Virtualization*. James Turnbull, 2019.

[33] R. Smith, *Docker Orchestration*. Packt Publishing Ltd, 2017.

[34] K. Relan, "Deploying Flask Applications," in *Building REST APIs with Flask*. Apress, 2019, pp. 159–182.

[35] T. Butler, *NGINX Cookbook*. Packt Publishing Ltd, 2017.

[36] J. Kreibich, *Using SQLite*. O'Reilly Media, Inc., 2010.

[37] S. Boschi and G. Santomaggio, *RabbitMQ Cookbook*. Packt Publishing Ltd, 2013.

[38] J. Cook, "The Dockerfile," in *Docker for Data Science*. Apress, 2017, pp. 81–101.

[39] K. Banker, *MongoDB in Action*. Manning Publications Co., 2011.

[40] S. Bradshaw, E. Brazil, and K. Chodorow, *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 2019.

[41] ADNI — Alzheimer's Disease Neuroimaging Initiative. Accessed: 10 December 2020. [Online]. Available: http://adni.loni.usc.edu/

[42] J. Mota, "Discovery of temporal patterns from multivariate time series data to support the classification of dementia profiles," Master's thesis, Instituto Superior Técnico, 2019.

[43] EC2 On-Demand Instance Pricing – Amazon Web Services. Accessed: 2 December 2020. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/

[44] K. Beer and R. Holland, "Encrypting Data at Rest," *White Paper of amazon web services*, 2014.

[45] D. Urda, J. L. Subirats, P. J. García-Laencina, L. Franco, J. L. Sancho-Gómez, and J. M. Jerez, "WIMP: Web server tool for missing data imputation," *Computer Methods and Programs in Biomedicine*, vol. 108, no. 3, pp. 1247–1254, 12 2012.

[46] J. Tong, P. Jiang, and Z. h. Lu, "RISP: A web-based server for prediction of RNA-binding sites in proteins," *Computer Methods and Programs in Biomedicine*, vol. 90, no. 2, pp. 148–153, 5 2008.

[47] V. Candeias. On-prem MAESTRO back-end. Accessed: 17 December 2020. [Online]. Available: https://github.com/vascocandeias/maestro-backend

[48] S. Chhajed, *Learning ELK stack*. Packt Publishing Ltd, 2015.

[49] B. Brazil, *Prometheus: Up & Running*. O'Reilly Media, Inc., 2018.

[50] Grafana: The open observability platform — Grafana Labs. Accessed: 4 December 2020. [Online]. Available: https://grafana.com/