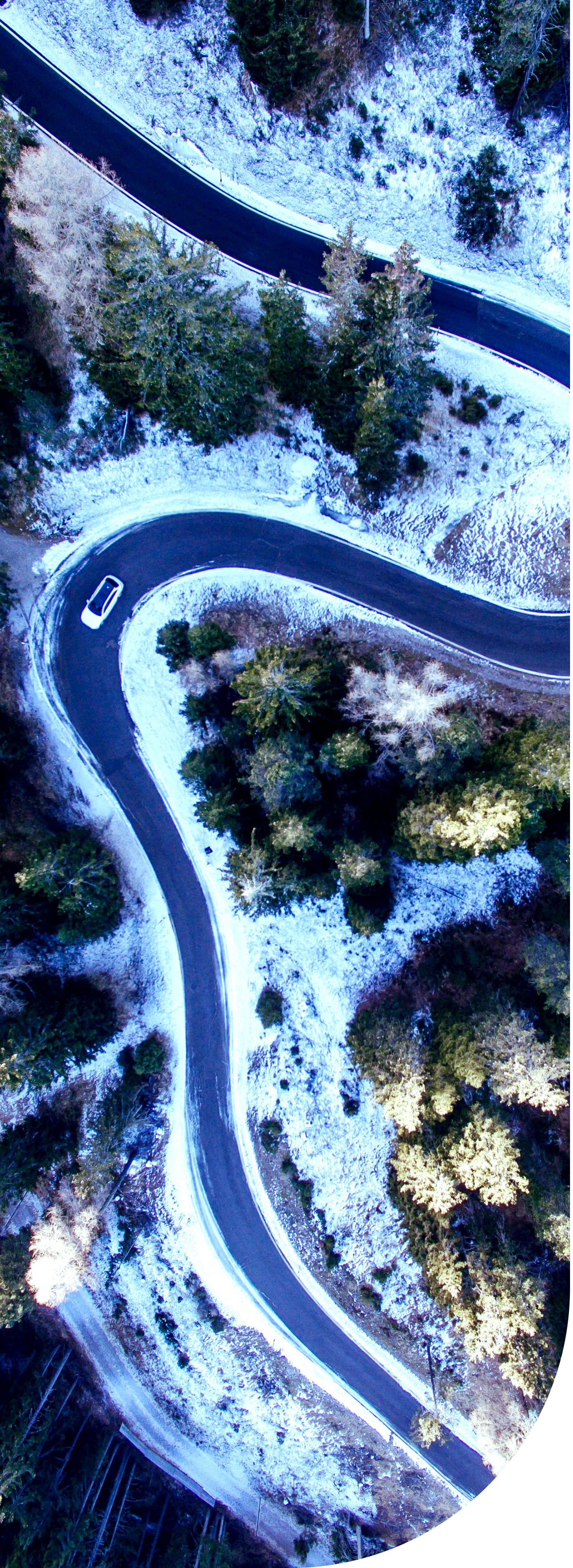

anodot

THE ESSENTIAL GUIDE TO **TIME SERIES FORECASTING**

Part III: A System Architecture Built to Support Autonomous Forecasting

INTRODUCTION

Accurately forecasting growth and demand within a business is one of the most important aspects of corporate planning. Every company needs to know how much revenue it realistically can expect to earn in the future, how much money it can comfortably spend in the coming year, what kind of demand there will be for its products and services in the quarters ahead, and so on. Without reliable forecasts for these metrics, a company would not know how to operate.

“Productizing” the generation of accurate on-demand or continuous forecasts requires a system architecture and many technical components. Anodot describes the architecture that underlies Autonomous Forecast.

In Part I of this white paper series, we talked about the challenges of doing forecasting manually. At best, a cumbersome process can only yield high-level forecasts for one or two critical metrics, and only for a one-time prediction. Even a data science team would build something that is not a product, doing their model training and running a forecast on some servers as a one-time effort. The next time the same forecast is needed, the kludge of a process must be repeated. It’s an inefficient use of resources and a totally ineffective way to conduct frequent or detailed forecasts. The far better approach is to use a fully automated, machine learning-based system that can generate forecasts on demand or even continuously.

Building the underlying infrastructure that can support such a system is no trivial matter. There are many ways that a technical team can approach building a system that essentially “productizes” the generation of accurate forecasts. Anodot’s team of highly skilled and experienced machine learning experts, application developers and DevOps specialists, among others, laid out a system architecture and technical components that have resulted in the SaaS-based turnkey forecasting solution known as Anodot Autonomous Forecast.

This white paper describes that system architecture and its core components, all of which have been built from scratch. Prior to Anodot’s efforts, there was no blueprint for how to build an autonomous forecasting system. We couldn’t go to some app store and pull a forecasting system off the shelf—but now Anodot customers can do just that. They can bring their time series data and relevant events, plug them into Autonomous Forecast, specify what they want to forecast, and get their desired output in the format they desire for consumption (e.g., via a dashboard, in alerts, in reports, etc.). It’s all very turnkey, push-a-button for customers, and getting to that point capped a complex year-long development effort by Anodot, which we describe in the pages below.

AN OVERVIEW OF THE SYSTEM ARCHITECTURE

At a high level, **Figure 1** below shows the blueprint Anodot developed for a machine learning-based forecasting system.

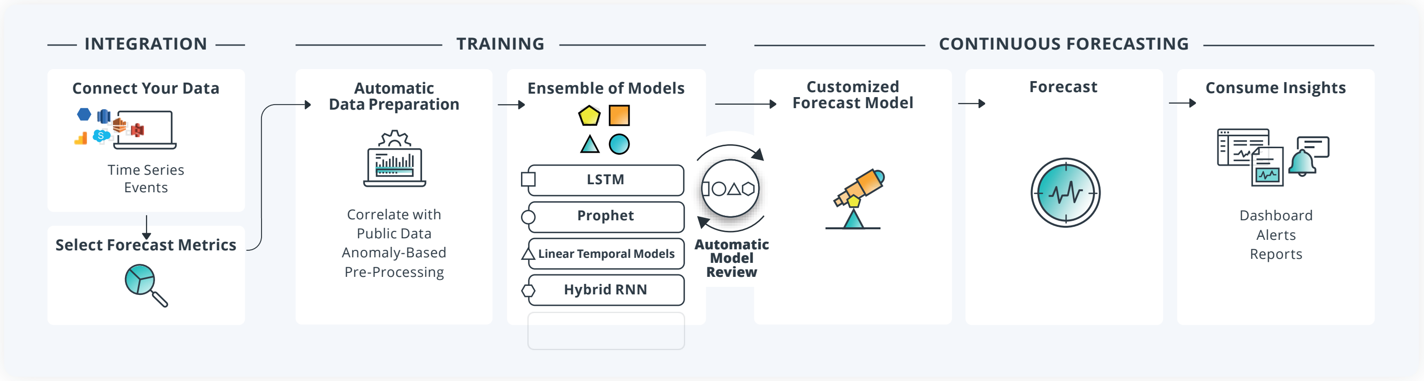

Figure 1. *The blueprint for Anodot's Autonomous Forecast product.*

In general, the processes of this blueprint go something like this:

- **CONNECT THE DATA –** Allow the customer to provide its own internal data sources, plus external data and/or events, if desired. Store this data in a time series database.
- **SELECT THE FORECAST METRICS –** Allow the customer to specify what metrics to forecast, over what time period.
- **DO AUTOMATIC DATA PREPARATION –** Using Anodot's Autonomous Detection solution, pre-process the historical data of the metric to be forecast by looking for anomalies that could affect the forecast. Depending on how many factors have been provided for the datasets, run various algorithms and procedures to weed out factors that are not relevant to the forecast metric.

- **TRAIN THE MACHINE LEARNING MODELS –** Send data and other relevant factors to a variety of algorithms to derive a forecast and then test each of the results for accuracy. Store all of the results and the information about the models for future use.

- **CREATE A CUSTOMIZED MODEL –** Based on the accuracy measurements, select the best models for the use case and create an ensemble from them that yields the custom model that can be used to forecast the metric on demand or continuously. Store this model and all its parameters for future use.

- **REVIEW THE CUSTOMIZED MODEL –** Conduct frequent reviews of the custom model to ensure it is optimized. If the accuracy is slipping, repeat the previous steps to retrain the models and create a new ensemble customized model.

- **MAKE A FORECAST –** Using the customized model and data supplied by the user, create an actual forecast and store the results for future use.

- **CONSUME THE FORECAST INSIGHTS –** Send output of the forecasting process to a dashboard, reports, alerts, or to other information systems.

There isn't a single architecture that everyone agrees on for accomplishing the process outlined above, so in a way, Anodot has created something new. It's how we chose to build our system so that it can ultimately be used by the non-data scientist and the non-developer.

THE THREE MAIN COMPONENTS OF THE FORECASTING SYSTEM

The illustration in **Figure 2** shows the main components of the forecasting system built by Anodot.

Shown on the left-hand side of the illustration, the training module is where we train the forecasting models. On the right-hand side is the forecasting module, which is the part that continuously forecasts the desired metric. The uppermost portion of the illustration shows the persistency layers comprised of the data stores where data, models and other pertinent details are kept.

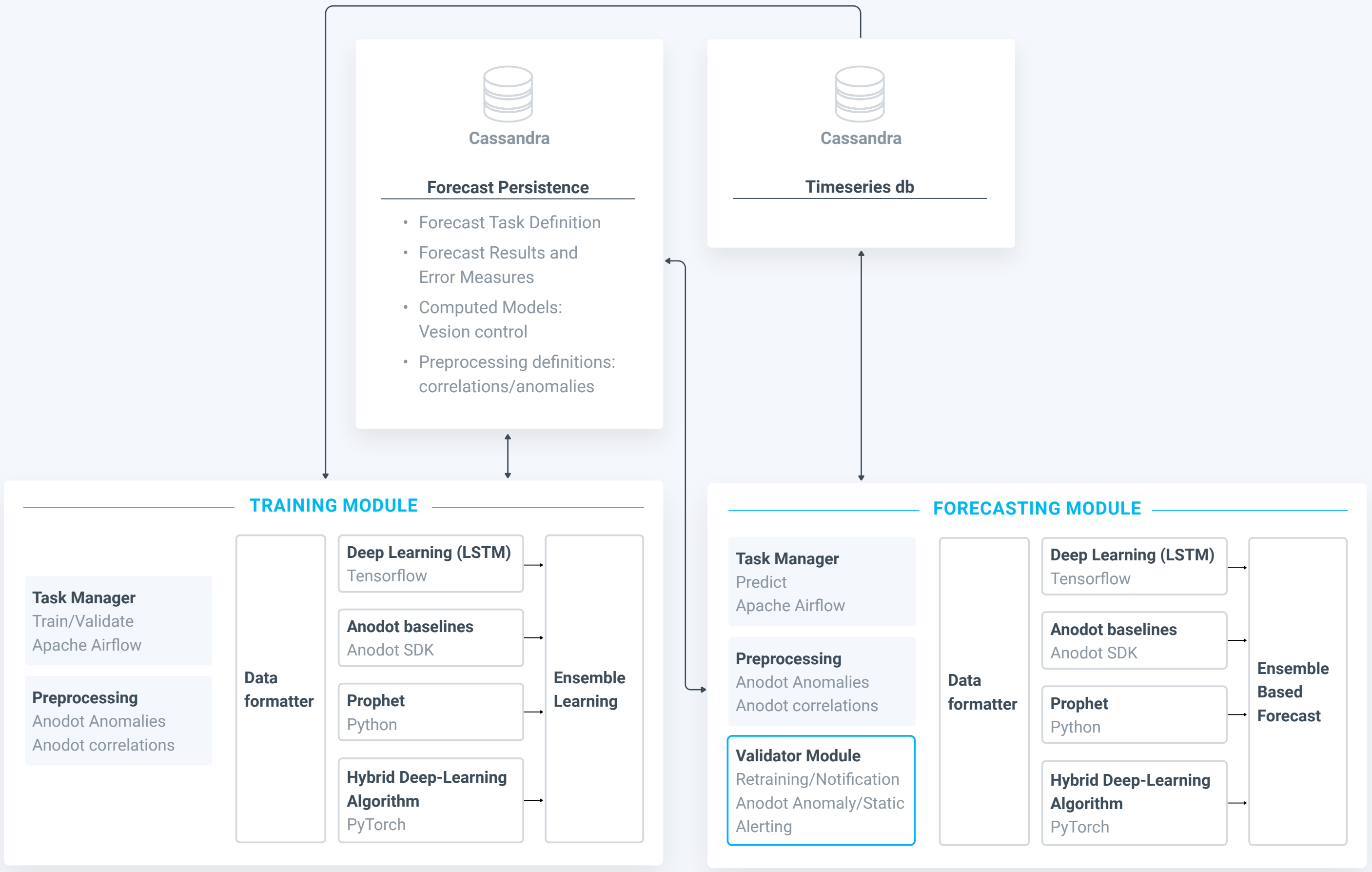

Figure 2. *The main components of the Anodot forecasting system.*

THE DATA STORE: PERSISTENCY FOR DATA, MODELS, DEFINITIONS AND MORE

The data store components of the solution are shown in **Figure 3**.

The first part of the data store (on the right-hand side of the image) collects and stores the actual data that is to be forecast, along with all the supporting data that will help with the forecast. For example, if we are forecasting revenue, all the data for revenue is stored in this time series database. Or, continuing with the eCommerce example used in Part II, if we are forecasting the sales of shoes, we keep historical data on shoe sales, as well as sales of adjacent product such as socks, plus information about special events like back-to-school sale dates and Black Friday sale dates.

The second part of the data store is for persisting the models themselves. When we train a model, we want to persist that description of the model, the forecast tasks and their definition, the results and error messages, and the models themselves with version control around them. We also want to persist any type of definitions we had around correlations and how to handle anomalies.

Anodot has chosen to use the Cassandra database for the data store. This persistence layer is very important in order to keep everything in a way such that it can be retrieved, debugged, perfected and used again. The ability to do this distinguishes a forecasting system from a one-time approach to forecasting where little is preserved for re-use.

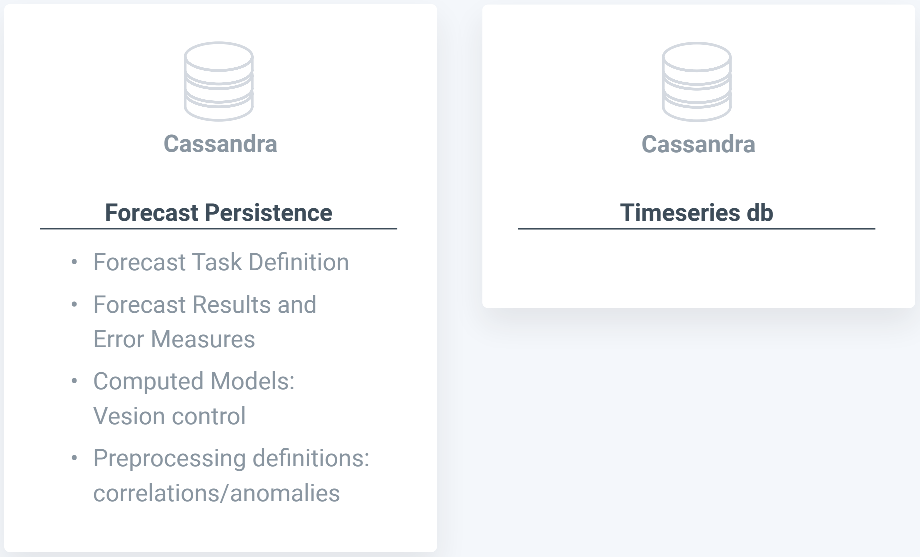
Cassandra
Forecast Persistence
Forecast Task Definition
Forecast Results and Error Measures
Computed Models: Vesion control
Preprocessing definitions: correlations/anomalies
Cassandra
Timeseries db

Figure 3. *The data store components.*

THE TRAINING MODULE

The training module component of the solution is shown in **Figure 4**.

In Part II of this white paper series, we discussed the design principles of our forecasting system. One of those principles involves training multiple forecasting models to see which have the best results. A model that works well for one dataset might not work quite as well for another, so training and testing a variety of models is important to attain the most accurate results possible.

Training a model entails a lot of pre-processing steps. This creates a need for a task manager component or a workflow component that allows us to predefine all the different parts of the workflow of what it takes to train the model itself. Anodot uses Apache Airflow for this task manager component. It allows us to schedule the execution of a wide array of tasks for the pre-processing step, which includes discovering anomalies in the data, and for the correlation step, which prepares the data for the different types of algorithms.

Then the task manager has to orchestrate the training of all the different models. Basically, it sends the data to the models, tells the system to spin up a new server (or set of servers) that will train, for example, an LSTM model with this data. When the LSTM model finishes its task, the task manager stores the data and waits for all the other models to finish their tasks. (In reality, we might train a thousand different types of LSTM models, a hundred different types of baselines, and multiple versions of Prophet and other algorithms.) Once the workflow manager understands that the models have completed their training, it directs the tasks to validate the models, then create an ensemble out of the models by looking at their errors and how well they did on the validation step and deciding how to combine them. The results of all these processes get stored in the data store.

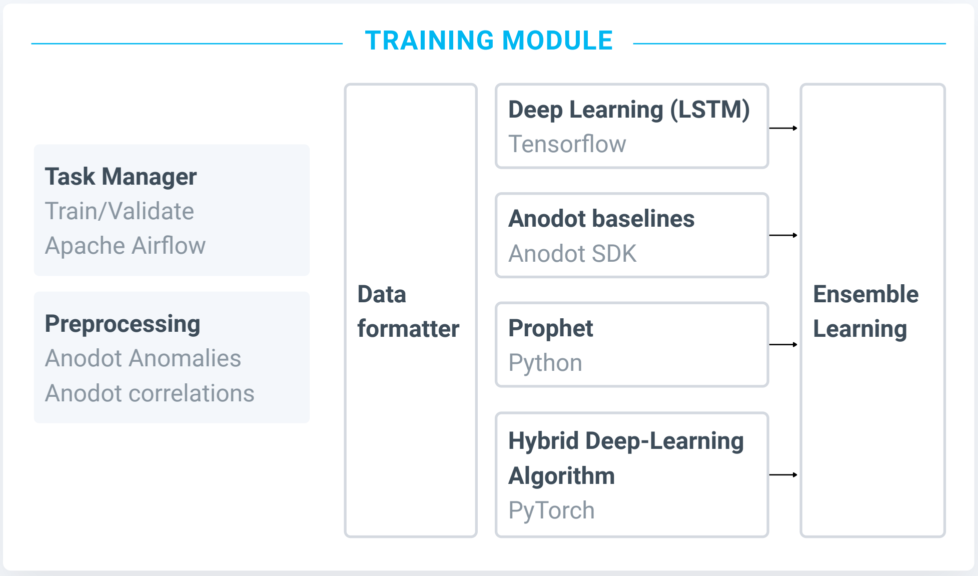

Figure 4. *The training module of the system.*

The task manager directs the orchestration engine Kubernetes to handle the physical aspect of spinning up and down the necessary servers to perform the various tasks. It could be one server or it could be a hundred, depending on the amount of activity for the tasks at hand. It's actually quite a lot of activity to have to orchestrate. Apache Airflow doesn't come "out of the box" knowing how to manage all this. This is where Anodot developers have to think about all these steps and code it into the task manager.

There are different technologies associated with different algorithms. For example, we use a version of LSTM that is implemented in an open source package called TensorFlow. Prophet is in Python, another algorithm is in PyTorch, and Anodot wrote many of our own linear algorithms. It means we have to combine a lot of different technologies coming from different sources, automate the process of spinning them up, running them, taking their output, storing it, then grouping it together with a consistent view. There's a lot of engineering work involved in getting this to work in the click of a button.

The way that data science teams usually work today, if they are given a forecasting task, they write something that's not at the click of a button. It's going to go one time and isn't permanent. Developing a system that preserves the data, the models, and so on for reuse again and again is not trivial at all.

THE FORECASTING MODULE

The forecasting module of the system is shown in **Figure 5**.

The forecasting module is somewhat simpler than the training module. Suppose a customer says, "I want to forecast my revenue, 30 days in advance, for all my countries." That's their definition, out of which comes a lot of extra data – basically the results of what was done on the training data during the training tasks – such as what pre-processing was done, which things we found correlated in the training phase, what anomalies we are taking out, and which things around the data preparation were necessary to run these models. All this gets stored with that forecast task.

Now when we need to forecast, we want to get a new data point for that revenue, tomorrow's data point. This forecasting module has to first figure out the forecasting task related to the data here. We need to load all these models that were chosen in the training phase into memory. If it's an LSTM model and one of the baselines or maybe Prophet or some combination of them, we have to bring up the servers and load it into the appropriate technology that can put this in memory. We do any pre-processing that would pre-define on that data the correlated metrics, the correlated events, and we find the anomalies in this new data. Then we push the new data into all these models and let them forecast. Once the forecast is done, we store the forecast back into the time series database, compute the error that we had from the last time we had data, and store that error into the forecasting persistency. And that's it.

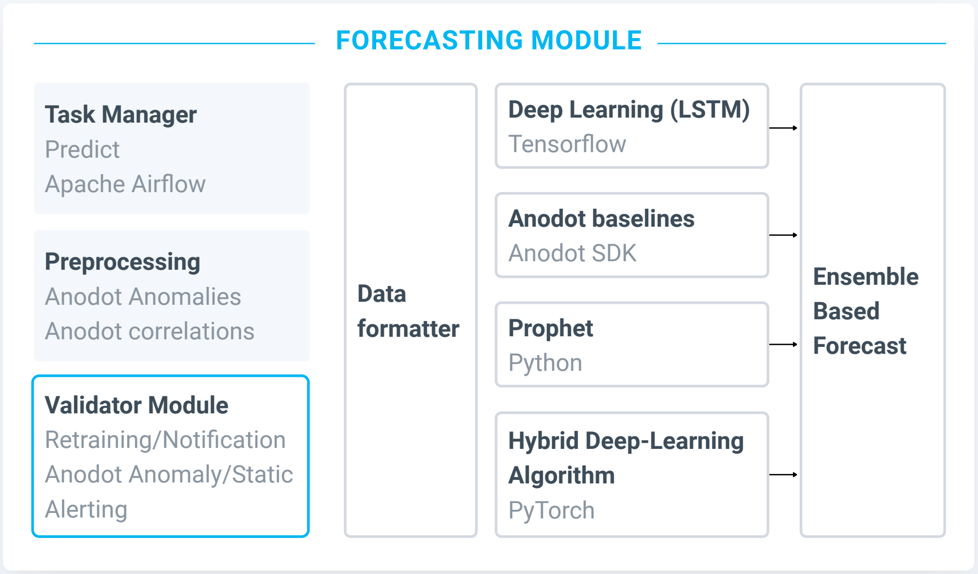

Figure 5. *The forecasting module of the system.*

Once those tasks are finished, we need to tear down all the servers we used for this process if we don't want to keep them running. If the forecast is done daily, we don't need to let these servers run, but If we do the forecasting every minute, then maybe we keep them alive. Tearing down the servers takes much less time than bringing up the relevant servers. For example, it takes a few minutes to bring up a server with TensorFlow that can run an LSTM model, but that server can be killed in seconds. The complexity of these activities is about the same, however.

As the task manager, Apache Airflow is the component that issues the commands to "do step A, then step B, etc." However, we have to write code to do all the steps that are necessary to train a forecast task and to model the steps necessary to actually perform the forecast. And Kubernetes is the piece that actually controls what is needed with the servers. Apache Airflow sends commands to Kubernetes to say, "You know we need a TensorFlow server. Bring me a TensorFlow server." And once Kubernetes does that, then Airflow says, "Okay, install this software on it." And now once that's installed, then Airflow sends the command to that server and says, "Okay, here's the data, send the data to this server, train the model." And once that's done and the process ends, Airflow will say, "Okay, this process ends, run the next process." That's the nature of how this is. The coding part of this effort, by the way, is not easy. It requires data science skills, DevOps skills and generic skills.

WHAT IT TAKES TO BUILD THIS SYSTEM: IN CONSIDERATION OF BUILD VS. BUY

As with any type of software application, there are some enterprise organizations that ask, "Could we build our own automated forecasting system? Is it better to build or buy this type of solution?" It's a fair question and we want to help companies that are considering the "build" option by sharing our experience of what it takes to build a fully automated forecasting system. Of course, this is Anodot's experience for the approach we have taken, and other companies' experiences might be different.

It takes a multi-disciplinary team of experts to design and build this type of system. The team needs data scientists – in our experience, at least three people – that understand series analysis and forecasting. Even though the forecasting algorithms are generally well known, getting them to work correctly with time series data is not trivial. Such algorithms don't work "straight out of the box" and they require some finessing to suit the purpose.

Using linear models takes another set of skills. All the baselines that we've built take a different type of skill than what's needed for working with LSTMs. Prophet is an open source product from Facebook and some people consider it to be easier to use. Nevertheless, building a system such as this takes a data science team that deeply understands time series analysis and that has a working knowledge about all the different types of models that are out there. It takes time to bring up such a team—and it's usually not one person because it's hard to find this level of knowledge and expertise all in one person.

The multi-disciplinary team also includes a software engineering team that helps set up the automation of all these processes, a user interface (UI) team, and a DevOps team that helps automate the coding part of it and the deployment of all these components. The team needs people who can work with the databases, the back-end systems, orchestration, and all the automation aspects.

Anodot takes on the complexity of working with the algorithms and models so that customers – end users – don't have to. The point of our product is to hide all the complexity from users so that no data science skills are needed on their end. Anodot has developed a SaaS product such that all a customer has to do is submit the data and define the task. However, a company developing a forecasting system for internal use may not need to be so fastidious about simplifying the solution such that a data scientist isn't needed to use the product. It's all a matter of preferences for how to use the solution once it's developed.

As for the time to develop this type of system, it's probably a year or more, plus additional time for research preceding the development effort. Also, keep in mind that Anodot had already developed the solution for anomaly detection that is a critical piece of the data pre-processing in the forecasting system. An enterprise building its own forecasting system will need a method for detecting anomalies in the data. (For information on what it takes to develop an anomaly detection system, please see our three-part white paper series on The Ultimate Guide to Building a Machine Learning Anomaly Detection System.)

A more likely scenario is that an enterprise wouldn't build an entire forecasting system such as that described in this series of white papers; instead a company might have a data scientist or maybe a team of them on staff to do one-time forecasts using open source algorithms. We discussed the drawbacks of this approach in Part I: the forecast is generally high level, covering a single metric, not as accurate as it could be, and it is a one-time forecast that is updated infrequently.

On the other hand, buying a solution – or in this case, using Software as a Service – means that the organization can begin forecasting today and not wait months to a year to get a forecast. Moreover, the forecast will be extremely accurate (see all the built-in design considerations outlined in Part II of this white paper series) and the forecast can be generated repeatedly on demand or continuously for better operational control. No data science team is required to use this solution.

SUMMARY

The process of building an autonomous forecasting system is complex and extensive. Anodot's approach splits the system architecture into three components: the data store, the training module and the forecasting module. There probably are other ways to design such a system, but we chose this approach because we believe it's the best way to deliver highly accurate forecasts without requiring the end user company to have a data scientist at the ready.

A skilled multi-disciplinary team is needed to handle different aspects of the solution: data scientists to select and work with the training algorithms, DevOps professionals to design and orchestrate the automated tasks, software engineers to code the complex tasks, and so on. Though companies could build their own forecasting system, it's much more practical to use what has already been built by the expert team at Anodot.

Let Anodot show you how you can achieve highly accurate forecasts for better business decisions.

GET A DEMO

For more information, please contact Anodot:

North America
669-600-3120
info.us@anodot.com

International
+972-9-7718707
info@anodot.com