Fault prediction in information systems

Love Walden
Abstract

Fault detection is a key component to minimizing service unavailability. Fault detection is generally handled by a monitoring system. This project investigates the possibility of extending an existing monitoring system to alert based on anomalous patterns in time series.

The project was broken up into two areas. The first area conducted an investigation whether it is possible to alert based on anomalous patterns in time series. A hypothesis was formed as follows; forecasting models cannot be used to detect anomalous patterns in time series. The investigation used case studies to disprove the hypothesis. Each case study used a forecasting model to measure the number of false, missed and correctly predicted alarms to determine if the hypothesis was disproved.

The second area created a design for the extension. An initial design of the system was created. The design was implemented and evaluated to find improvements. The outcome was then used to create a general design.

The results from the investigation disproved the hypothesis. The report also presents a general software design for an anomaly detection system.

Keywords

microservice; forecasting; time series; metrics; anomaly detection; software design;
Abstract

Feldetektering är en nyckelkomponent för att minimera nedtid i mjukvarutjänster. Feldetektering hanteras vanligtvis av ett övervakningssystem. Detta projekt undersöker möjligheten att utöka ett befintligt övervakningssystem till att kunna skicka ut larm baserat på avvikande mönster i tidserier.


Resultaten från undersökningen motbevisade hypotesen. Rapporten presenterar även en allmän mjukvarudesign för ett anomalitetsdetekteringssystem.

Nyckelord

Mjukvarudesign; Prognosmodell; tidserie; avvikande mönster; övervakningssystem; larm;
# Table of Contents

1 Introduction................................................................................................. 4  
1.1 Background.............................................................................................. 4  
1.2 Problem...................................................................................................... 4  
1.3 Purpose....................................................................................................... 5  
1.4 Goal............................................................................................................ 5  
1.4.1 Benefits, Ethics and Sustainability............................................................ 5  
1.5 Methodology / Methods........................................................................... 6  
1.6 Stakeholder................................................................................................. 7  
1.7 Delimitations.............................................................................................. 7  
1.8 Outline........................................................................................................ 7  

2 Software design and fault prediction theory.......................................... 8  
2.1 Microservices............................................................................................ 8  
2.1.1 Microservice communication................................................................. 9  
2.2 Forecasting................................................................................................. 9  
2.2.1 Time series patterns................................................................................ 10  
2.3 Rolling mean.............................................................................................. 10  
2.4 Facebook prophet...................................................................................... 11  
2.5 Prometheus................................................................................................. 11  
2.5.1 Prometheus API...................................................................................... 12  
2.6 Related work.............................................................................................. 12  

3 Project approach......................................................................................... 14  
3.1 Investigation.............................................................................................. 14  
3.1.1 Case study design.................................................................................... 15  
3.1.2 Reliability and internal validity.............................................................. 15  
3.1.3 Choice of models.................................................................................... 16  
3.2 Software design.......................................................................................... 16  
3.3 Project methods.......................................................................................... 17  
3.3.1 Project prioritization............................................................................... 17  

4 Time series fault prediction.................................................................... 19  
4.1 Case study design...................................................................................... 19  
4.2 Training sets.............................................................................................. 19  
4.3 Test sets...................................................................................................... 22  
4.4 Prediction intervals................................................................................. 24  
4.5 Anomaly detection.................................................................................... 26  
4.6 Data set preprocessing.............................................................................. 27  
4.7 Implementation.......................................................................................... 29  
4.7.1 Rolling mean........................................................................................... 29  
4.7.2 Facebook prophet.................................................................................... 30  

5 Software design............................................................................................ 31  
5.1 Design process.......................................................................................... 31  
5.2 Initial design.............................................................................................. 32  
5.3 Anomaly detection service........................................................................ 33  
5.4 Retrieval service....................................................................................... 35  
5.5 Test result.................................................................................................. 35  
5.6 Design refinements.................................................................................... 36
6 Results............................................................................................................38
6.1 Measurements.........................................................................................38
6.2 Software design.......................................................................................40
7 Discussion..................................................................................................43
7.1 Fault prediction.........................................................................................43
7.1.1 Validity..................................................................................................44
7.2 Development method..............................................................................45
7.3 Future work..............................................................................................45
7.4 Conclusion...............................................................................................46
References.....................................................................................................48
Appendix A.....................................................................................................1
Appendix B.....................................................................................................1
1 Introduction

There have been a large influx of software applications over the last decade. As of 2018 we can view bank statements, buy groceries, order taxis online etc. As a consumer, there is an expectation that these services always work. When a service is unavailable it generally results in lost revenue. Hence, it is desirable to minimize the amount of service unavailability. When service unavailability occurs, the length in time it is unavailable can be determined by several factors. One factor to determine the length of unavailability is the time to detect a fault has occurred. Another factor is the time to isolate and repair the fault etc. This project focuses on detecting faults. To detect a fault has occurred, applications and systems are monitored. Monitoring applications and systems can be carried out through verifying the application or system is still available. For example, one way to verify the application is available is to invoke a request. If the application fails to respond to the request, the application can be inferred as unavailable. The verification is referred to as a check. If the check fails, a notification is sent out, generally for human intervention. The notification is referred to as an alarm [12]. Implementing checks and sending out alarms is generally automated by a monitoring system.

1.1 Background

An application may record information about itself. Such as the total number of requests it has handled, referred to as a metric. A metric within this project is a value associated with an identifier. The identifier is a string which identifies the metric. The application may also publish the metrics it has recorded for collection. A monitoring system may then collect the metrics in intervals, and record at what time the metrics were collected. The collected metrics then form a sequence of values ordered by time, referred to as a time series [5].

Metrics allows rules to be defined. A rule is generally a condition or a threshold. A condition indicates an expression is either true or false. For example, if an application is up, it may report a metric with identifier “application_up” and value 1. The condition could then be defined as “application_up” equals 1 to indicate the application is up. A threshold indicates that a metric must be either above or below a predefined value, otherwise the threshold is breached. For example, a threshold could be configured to specify central processing unit (CPU) utilization must be below 95%. If CPU utilization is above 95%, then the threshold is breached. If a condition is not met, or a threshold is breached, the corresponding rule is then broken. If a rule is broken, an alarm is sent out by the monitoring system.

Metrics are collected in intervals. Since the time is recorded of when a metric was collected, time series are formed. Time series allows rules to be extended to include a condition of time [12]. For example, if CPU utilization is above 95% for 5 minutes then an alarm is sent out.

1.2 Problem
There are a few problems with rules. Firstly, if a value continuously fluctuates between a limit defined by a threshold, an alert will not be sent as the condition of time is not met. Secondly, when a rule is broken it is often an indication that a fault has happened. It would be advantageous to report before the fault takes place or detect it early. One solution is to put conservative thresholds, or shorter timing conditions. However, this might lead to alarms being sent out which does not require intervention, referred to as a false alarm. Lastly, thresholds and conditions are statically configured and generally approximations of what is believed to indicate downtime. If the approximation changes, the corresponding rule will have to be updated. Hence, it is desirable to automatically adapt and update the approximations over time.

By graphing and observing historical metrics, some general patterns emerge. Thus, mathematical model should be able to make predictions of what values are likely to be encountered at future points in time. When an encountered value diverges significantly from a predicted value, an outlier occurred [7]. A set of outliers then forms what is referred to as an anomaly. An anomaly might be an indication of an underlying fault, or that a fault is about to occur. Hence, the problem statement is:

How can a monitoring system be extended to alert based on anomalous metrics?

1.3 Purpose

The purpose of this project is to provide a foundation for a system which can send out alarms by detecting anomalies in time series. The foundation should be general and serve as a basis for future development. Furthermore, this project conducts a rudimentary investigation of whether mathematical models can be used to detect anomalies in time series. As such, the report could serve as a basis for a more extensive project of time series anomaly detection.

1.4 Goal

The goal of this project is to create a software design for a system which can automatically detect faults in time series. The goal involves two parts. The first part is to investigate whether detecting faults in time series is possible and feasible. The second part is to create a general design which extends existing monitoring systems.

1.4.1 Benefits, Ethics and Sustainability

The United Nation have 17 sustainable development goals which are targeted to be achieved by 2030 [40]. Anomaly detection could potentially be utilized in two of the goals, clean water and clean energy. In the case of clean water goal, target 6.3 is to improve water quality by reducing pollution and release of hazardous chemicals and materials [41]. Water values could continuously be collected. Because these values would form a time series, anomaly detection could be applied to detect anomalies in water values which are affected by pollution and chemicals or material. Alarms could then be sent out if one of these values are affected for intervention. In the case of clean energy, target 7.3 specifies to double the rate of improvement in energy efficiency [42]. The
United States Data Center energy efficiency usage report estimates data center power consumption in the United States to be 73 billion kWh in 2020 [43]. One contributing factor to power consumption according to the report is inactive servers. An inactive server represents obsolete or unused servers which consume electricity [43]. One reason for inactive servers is the concept of hardware redundancy, where an application run on multiple physical servers to reduce the risk of downtime. In some environments, these servers are unused until one of the active servers breaks. If forecasting methods can predict when a fault is about to occur or detect it early, the fault could potentially be avoided. Given that the fault can be avoided, it stands to reason that redundancy may be reduced by having inactive servers powered off until a fault is about to happen. For instance, inactive servers could be powered off and then be powered on when a fault is predicted to occur.

There were a few ethical concerns regarding this project. Firstly, time series presented in this project come from the stakeholders production systems. These time series may give indication of how well the stakeholder is doing. One example is the number of visitors observed over an extended period of time, whereby trends can be inferred based on the pattern. Material used by the project was thus treated with confidentiality and any confidential material should have written consent before being presented [1]. No time series presented in this project was without prior consent from the stakeholder. Furthermore, the service from which the time series come from will be kept secret.

Secondly, a collected time series could potentially be linked to personal data. Personal data is protected by the General Data Protection Regulation within the EU [51]. No time series collected and used within this project could be tied to personal data. However, extra consideration went into this point as processing personal data is unlawful without consent under article 6 [52].

1.5 Methodology / Methods

To answer the problem statement, the project was broken down into two parts. The first part investigated the possibility of alerting based on anomalous patterns in time series. If the first part of the project was unsuccessful, then the monitoring system can’t be extended. The second part concerned creating the design of the system. The goal of this project is to create multiple artifacts. The artifacts together make up a software design extending a monitoring system. A qualitative research method partly concerns development of artifacts [1]. Hence, this project used a qualitative research method to develop the software design.

The first part of the project formed the following theory; forecasting models can be used to detect when anomalous patterns occur in time series. This project only attempts to strengthen this theory. To strengthen the theory an hypothesis was formed as follows; forecasting models cannot be used to detect anomalous patterns in time series. An applied research method is used to answer a specific question. In applied research, a set of circumstances is examined and the results are related to a situation [1]. To disprove the hypothesis an applied research method was used. In a deductive approach an
existing theory is tested to verify or disprove a hypothesis [1]. The first part of 
the project concerns forming a hypothesis and disproving it. Hence, a 
deductive approach was used in the first part. 
The second part of the project was to create a design which extended a 
monitoring system. In an inductive approach a theory is formed based of 
observations [22]. An inductive approach can also be used to develop an 
artifact [1]. To create the design an inductive approach was used. A literature 
review was carried out. Based on the literature review the design was created 
and evaluated. 
An abductive approach combines an inductive and a deductive approach to 
form a conclusion based on an incomplete set of evidence [1]. The project was 
split into two parts to reach a conclusion. In the first part a deductive 
approach was used. In the second part an inductive approach was used. As the 
project combines the two approaches to reach the conclusion, an abductive 
approach was used. 

1.6 Stakeholder 
Pricerunner offers a free price-comparison site for consumers. It is Swedens 
largest price-comparison site with over 1 648 000 products and 5600 stores. 
Everyday, 121 000 000 prices are updated [6]. As of 2016, ownership of the 
company changed. A new project was started to re-invent the website. Since 
then, the IT environment has grown rapidly, which in turn has led to a need of 
detecting faults quickly within the environment. Pricerunner now seeks to 
extend the existing monitoring system to alert based on anomalies. 

1.7 Delimitations 
This project will not present methods of isolating causes to anomalies 
reported. Rather, it will focus on finding and reporting them. Futhermore, this 
project does not seek to dive deeply into how certain models work. Instead it 
will present the models used with a high level overview and resulting 
outcomes from case studies performed. 
Multiple metric values from multiple time series could be used as inputs to 
models. However, this thesis only use one time series at a time as input to a 
chosen model. 
The extension of the monitoring system should fit into the stakeholders IT 
architecture. The implication of this delimitation is that a specific 
architectural style has to be used. This architectural style is discussed in 
section 2.1. 

1.8 Outline 
Chapter 1 gives an introductory overview of the degree project. Chapter 2 
presents background theory of the project. Chapter 3 presents the research 
and project methods used in the degree project. Chapter 4 presents an 
investigation conducted the disprove the hypothesis. Chapter 5 presents the 
development of the software design. Chapter 6 presents the results of the 
project. Chapter 7 discusses the results, describes future work and concludes 
the thesis.
2 Software design and fault prediction theory

This chapter presents the theoretical parts behind the project. Section 2.1 presents the software architectural style microservices. Section 2.2 presents general forecasting concepts. Section 2.3 presents the forecasting model rolling mean. Section 2.4 presents the forecasting model Facebook prophet. Section 2.5 presents the monitoring system Prometheus. Section 2.6 presents related work.

2.1 Microservices

Microservices are a software architectural style [45]. In a microservice architecture, a system is broken up into components [15]. A component is referred to as a microservice. A microservice should be small, independent and replaceable. To achieve this, there are two prerequisites which need to be met by a microservice, loose coupling and high cohesion. Loose coupling is when a microservice have few dependencies to other microservices [16]. A change in one microservice should not result in a required change in other microservices. If this condition is fulfilled, a service is both independent and replaceable. High cohesion indicates related behavior should grouped. If high cohesion is fulfilled, in order to change behavior only one place needs to be changed [9].

A microservice should have a clear area of responsibility. Each microservice should then collaborate to achieve a common goal. Note, there should never be a circular dependency between two microservices as this is an indication of tight coupling. To collaborate, a microservice provide an application programming interface(API) for other microservices, and may use other microservices APIs [9]. An API consists of a set of functions, which have a set of input parameters and return values. The purpose of an API is to separate a client from implementation [17]. A client which uses an API then depends on the functions, input parameters and return values provided. To build an abstraction, this will be referred to as a contract between a client and implementer of an API. If a contract is broken, meaning a function or parameter is changed within an API, a client consuming the API must also change.

Microservices provide contracts and uses contracts from other microservices. If a contract is broken, all microservices depending on the contract have to be changed. Hence, if a contract is broken and multiple microservices depend on the contract, the condition of a change only affecting the microservice itself is broken. A solution is to version the API [9]. API versioning means a contract is valid for a specific version of the API. Instead of breaking the contract, a new contract is created for a new version. The old contract can then be supported until all microservices depending on the specific version of the contract have migrated to a newer version. Hence, a microservice may support multiple versions of an API, and clients will not have to change implementation until an API version is no longer supported [9].
2.1.1 Microservice communication

Microservices collaborate by communicating over a network. This project focuses on communication through the Hypertext Transfer Protocol (HTTP). HTTP is a message driven protocol, which consists of requests and responses. A client issues a request, to which a server issues a response [2]. Each HTTP message is either a request or response [3].

A request partly consists of a method and a Uniform Resource Identifier (URI), it may also contain a body which consists of arbitrary data. A resource is defined as the target of an HTTP request [3]. The URI is a string which identifies which resource the request is intended for. The method indicates which method to apply on the resource [2]. The URI and method together forms an abstraction defined as a route. A route is a mapping from the URI and method to a function to execute on the server [19]. There are various HTTP methods available, this project used the methods GET, POST and PUT. GET is used to retrieve the state of a resource. POST is used to create a new resource. PUT is used to update the state of a resource [8].

A response partly consists of a status line, status code and may include a body [2]. The status code indicates if the request was successful or not. The body contain arbitrary data.

Requests and responses may contain a body consisting of arbitrary data. The data may be of a specific format. The data format used in this project was JavaScript Object Notation (JSON) [4].

2.2 Forecasting

Forecasting concerns predicting the future [7]. One common application is weather prediction, as for example, the likelihood of rain tomorrow can be estimated by using historical data. This project specifically focuses on time series forecasting. A time series forecast uses a historical time series to predict future values of the time series [7]. The technique to produce a forecast is referred to as a model. The historical time series used to create the forecast is referred to as the training set. The training sets are time series consisting of datapoints. A datapoint is a point in time associated to a value. Time is denoted by $t$, the value at a time $t$ is then denoted by $y_t$. A model produces a forecast based on a time series up until a time $t$. The forecast produced is another time series consisting of predicted datapoints in the future from $t$. The data points in the forecast are denoted by $\hat{y}_t$, which means the predicted value of $y$ at time $t$ [18]. To measure how accurate the forecast was, a set of data points from the future of time $t$ can be used. This set of datapoints is referred to as a test set.

Predicting future values of a time series generally carry uncertainty. To indicate how certain a model is of a predicted value, the model can produce a prediction interval. The prediction interval indicates a future value is within the range of the interval with a specified probability [7]. Within this project, a future value which falls outside a prediction interval is referred to as an outlier. An anomaly is then a set of outliers.

Both the forecast and the prediction intervals depend on the training set. If the training set contain anomalies, the forecast might become inaccurate and the prediction interval wide. To make the forecast more accurate and the
prediction intervals narrower, data points from the training set may be removed. This technique is referred to as preprocessing. More generally preprocessing is utilized when removing, adding or transforming data points in the training set [18]. This project focused on two preprocessing techniques. The first preprocessing technique removes any data point which is n standard deviations from the mean of the training set. The first technique is referred to as \( n^* \sigma \). The second technique is to remove isolated data points, as these are more likely to be outliers. To detect isolated data points, the algorithm isolation forest was used, for details see [36]. The library scikit-learn [37] provides an implementation for isolation forest [38]. One parameter were explicitly used in this project, the contamination parameter. The contamination parameter indicates the proportion of outliers in the data set [38].

2.2.1 Time series patterns

A time series may follow certain patterns. A time series is said to have a seasonal pattern if it is affected by time, such as hour of the day, weekends etc [7]. Seasonal pattern tend to occur in metrics affected by users. As for example, the number of logged in users at a specific hour tend to variate depending on the time of the day. A time series may also have a trend. A trend indicates there is an increase or decrease over time in the time series. A time series is said to be stationary if it does not depend on time, meaning it does not have a trend or seasonality [7]. The stationary pattern generally shows up in metrics such as error rates.

2.3 Rolling mean

A time window refers to consecutive values within a time series over a period of time [31]. As for example, a time window could be of size 10, the time window then would include 10 consecutive values observed up until a time \( t \). A rolling mean model is the average of a time window [31]. The window is updated as time moves forward, hence the mean is also updated. To predict \( \hat{y}_t \) using a rolling mean model, the mean of the window at time \( t-1 \) is used, defined as \( \frac{1}{n} \sum_{i \in T} y_i = \hat{y}_t \). Where \( T \) is the set of values within the time window at time \( t-1 \) and \( n \) is the number of values in \( T \). To construct a prediction interval for the parameter \( \hat{y}_t \), the standard deviation of the time window may be used, defined as \( \sqrt{\frac{1}{n} \sum_{i \in T} (y_i - \hat{y}_i)^2} = \sigma_t \). However, using only one standard deviation yields a narrow prediction interval. The 68-95-99 rule states 68% of all observations from a normal distribution lie within one standard devation of the mean, 95% within two and 99% within three standard deviations The 68-95-99 rule rests on the assumption the observations are outcomes from a normal distribution [49]. To clarify, the time series used within this project aren’t necessarily observations from a normal distribution. The 68-95-99 rule is used as a rough guideline of the width of a prediction interval produced by a rolling mean model. The library
Pandas [26] provide an implementation for the rolling mean model which was used by the project [48].

2.4 Facebook prophet

Facebook prophet (fbprophet) is a forecasting library. Fbprophet takes into account both trend and seasonality when calculating forecasts [27]. Fbprophet also produce prediction intervals for a forecast. To produce a forecast by fbprophet, firstly a Prophet object must be created. The Prophet object take an optional parameter “interval_width” which controls the width of the prediction intervals produced.

Secondly, a fit method can be called with the training set as an argument. The training set should be in a tabular data structure, referred to as a dataframe. The dataframe is provided by the Pandas library [50]. The fit method expects the dataframe to contain two columns, a datestamp (ds) and a y column. The y column represents the value observed at a specific time. The datestamp column should be of the date format YYYY-MM-DD HH:MM:SS [30].

Lastly, after the model has been fitted, the model can produce forecasts. To produce a forecast, a method called predict is called. The predict method expects another dataframe as an argument. This dataframe should contain a column called ds. The ds column should contain timestamps for which the model produces the forecast [30]. To create the dataframe used for the predict method, fbprophet provides a helper function called make_future_dataframe [30]. The make_future_dataframe function take a periods and a frequency argument. The period specifies how many timestamps to create in the ds column. The frequency argument specifies the intervals between the timestamps in the ds column. For example, the make_future_dataframe method could be called with a periods argument of 120 and frequency argument of 30 seconds. The method will then return a dataframe containing 120 timestamps with an interval of 30 seconds between each timestamp. If the predict method is called with this dataframe, it will produce a forecast for the next hour.

The forecast returned from the predict method is yet another dataframe. The returned dataframe contain the forecast, prediction intervals and a timestamp. The forecasted values are present in the column labeled yhat. The upper and lower prediction intervals are present in the columns labeled yhat_upper and yhat_lower respectively [30]. The timestamp specifies which time the predicted values are valid for.

2.5 Prometheus

Prometheus is an open source monitoring system. Its main purpose is to collect and store metrics. In addition, Prometheus exposes collected metrics through a query language and run rules for alerting based on collected metrics [10].

An application may expose its metrics through an hypertext transfer protocol (HTTP) endpoint. Prometheus collects metrics from the endpoint in time intervals. The endpoint must return metrics according to an exposition format [11][12]. The collected metrics are then stored in a time series
database. The time series database stores a value is with a timestamp, identified by a metric name and a set of labels [13].

Prometheus contain four different metric types. Only two of them will be used in this thesis, counter and gauge. A counter is a numerical increasing value, it may stay the same, increase or reset between scrapes. An example is number of request handled by some application. A Gauge is a numerical fluctuating value, it may either go up, down or stay the same in between scrapes. An example is memory usage.

Prometheus offers a numerous functions based on a query language (PromQL) [12]. Of these functions, rate is of particular interest. Rate calculates an average increase per second of a time series based on a range vector [12][14]. A range vector contains values within a time interval defined. Hence, it may be used to display an average request per second over a time interval.

Rules may be written as PromQL expressions, which prometheus will evaluate continuously [12]. Once a rule is broken (e.g a condition is met), an alert is sent out. Alerts should only be raised if there needs to be human intervention. Thus, one property is that false alarms should be avoided. Another property is alarms should always be sent out when there needs to be intervention.

2.5.1 Prometheus API

PromQL is accessible through an HTTP API which Prometheus exposes. The project used two functions provided by the API, query and range queries. The query function returns the value of an input query at a specific point in time. The function takes a query parameter which is a Prometheus expression. The function also takes a time parameter which specifies a point in time the query should return a result from. If the query function is called without a time parameter, the value from the current time of the Prometheus system is returned [46]. The query parameter is useful to extract the current value of an expression.

Range queries returns metric values over a range of time. The function takes a query parameter which specifies a Prometheus expression. The function takes a start and end timestamp, which is the range of time values should be returned from. The function also takes a step parameter which specifies the resolution of the returned values [46]. Range queries are useful to extract training data sets from prometheus.

Values to a corresponding query are returned from the Prometheus API with unix timestamps. A unix timestamp is time represented in seconds from the reference date 1970 January 1st 00:00 UTC [47]. Unix timestamps are represented by a number. In order to apply methods in the fbprophet library, the unix timestamps have to be converted to date timestamps.

2.6 Related work

As the project was split into two parts, an investigation and a software design of the system. Related work can either fall into the category of the investigation, a system for anomaly detection or both. No related work which falls in both categories were found.
Skyline is an anomaly detection system [29]. Skyline receives metrics from the monitoring daemons StatsD or Graphite. Skyline also have support for other extensions, however, these have to be developed. Skyline is no longer actively maintained. Banshee is another anomaly detection system [33]. However, Banshee is tightly coupled to StatsD which does not fit into Pricerunners architecture.

A similar, but more extensive investigation of time series anomaly detection was conducted by Shipmon et al. [34]. They used different models to predict future values and took another approach to inferring anomalies than presented in this thesis. Resolution indicates how many seconds a data point covers. The data sets used by Shipmon et al. differs in resolution from the data sets used in this thesis. However, the measurements taken in the investigation presented in this project drew inspiration from the one conducted by Shipmon et al.
3 Project approach

To answer the problem statement of how an existing monitoring system can be extended to alert based on anomalous metrics, the project was divided into two parts. The first part conducted an investigation to determine whether it is possible to alert based on anomalous metrics. If it isn’t possible to alert based on anomalous metrics, the monitoring system cannot be extended, and hence the second part of the project wouldn’t be necessary. To conduct the investigation an applied research method was used. The second part of the project was to create a software design of the extension to the monitoring system. The software design included three artifacts, an overview of the whole system, a state diagram and a system sequence diagram. To create these artifacts, a case study was conducted where an extension to the monitoring system Prometheus was developed. The second part relates to how the monitoring system can be extended.

3.1 Investigation

The investigation sought to answer if forecasting models could be used to detect anomalous patterns in time series. If forecasting models couldn’t be used to detect anomalous patterns, then the monitoring system couldn’t be extended to alert based on anomalous patterns. To answer if forecasting models could be used, a hypothesis was formed as follows; forecasting models cannot be used to detect anomalous patterns in time series. A successful investigation would contradict the hypothesis. A contradiction was chosen due to two reasons. The first reason was, this project investigates a subset of time series patterns. Proving the opposite hypothesis, forecasting models can be used to detect anomalies, would only generate a partial result for the investigated time series. The second reason was, the purpose of this project is to be a basis for future development and research of anomaly detection systems. Hence, the project seeks to strengthen the theory that forecasting models can be used to detect anomalies in time series. A contradiction makes a more general statement and encourages more extensive research into the area. Due to the two listed reasons, a contradiction of the hypothesis was chosen.

To disprove the hypothesis, only one instance where a forecasting model was used to detect anomalies had to be shown. An applied research method was used to disprove the hypothesis. An applied research method can be used to answer a specific question where a set of circumstances and results relate to a particular situation [1]. The circumstances was historical time series under two different situations. The first situation involved time series where at least one fault had occurred. The second situation involved a time series where no faults had occurred. Results from the first situation was used to disprove the hypothesis. Results from the second situation was used as validation of the results from the first situation. Desirable results were to only raise alarms whenever there was a fault occurring.
3.1.1 Case study design

A case study is a strategy where an empirical investigation of a phenomenon is conducted by using multiple sources of evidence [1]. To generate results which could disprove the hypothesis a case study approach was used. As the scope of the project was limited, an empirical approach to obtaining the results was chosen.

To produce the results, two commonly and one rare occurring time series patterns were chosen. The time series consisted of collected metrics by Prometheus. The time series were collected in 30 second intervals. Each time series was divided into two sets, a training set and a test set. Each training set consisted of at least 14 days of data. Each test set consisted of 24 hours of data. Each test set began right after the corresponding training set. For example, if the training set had its last entry Friday 23:59:30, the test set would have its first entry Saturday 00:00:00. Hence, the test set was up to 24 hours of data in the future from the training set. The time series were investigated under two scenarios. The first scenario consisted of a test set without any faults occurring. The second scenario consisted of a test set where one or multiple faults occurred or were simulated.

Three types of measurement values were taken in each case study. The first type was a true alarm. The test set contained predefined time spans where if an alarm was raised, it would count as a true alarm. The second type was a missed alarm. If an alarm was not raised during one of the predefined time spans, it would count as a missed alarm. The third type was a false alarm. If an alarm was raised outside one of the predefined time spans, it would count as a false alarm. The hypothesis was considered disproved if the following conditions were met for at least one of the investigated time series under both scenarios;

- Raised alarms during all predefined time spans.
- Did not raise any false alarms
- Did not fail to raise any alarms during one of the predefined time spans

3.1.2 Reliability and internal validity

Reliability refers to the consistency of the results of the project. Reliability also refers to the extent the outcomes of the project can be replicated by others [39]. In other words, can the study be repeated and obtain the same results? [44]. A specifically important point is if the results are consistent with the data collected. Reliability is said to be increased if an experiment is repeated and yields the same results [39]. To ensure the results were consistent with the data collected, each case study were conducted five times. The results from the case studies were considered consistent if the model missed the same amount of alarms, and generated the same amount false alarms in each case study. The total amount of alarms raised were allowed to vary as this metric is not of importance to disprove the hypothesis.

Internal validity refers to if the results match reality [44]. More generally, internal validity poses the question whether a measurement measures the correct thing [39]. For example, a thermostat may consistently measure
boiling water to 25 degrees Celsius. The results are reliable as the results are consistent. However, the results are not valid as the incorrect temperature is consistently reported [44].

In terms of this project, the correct thing to measure is if a model can correctly predict when a fault has occurred. As mentioned in the preceding section, time spans of when a model should alert was set up. These time spans were when actual faults had occurred or simulated. In the seasonal time series, the fault was simulated. Hence, the time span was between when the simulated fault began and ended. In the stationary time series, the specified time spans were cross referenced against a monitoring system. A time span was marked as a fault if the monitoring system raised an alarm. The beginning of a marked time span was when the time series first began to diverge from the pattern of the time series. The end of a marked time span was when the time series returned to the normal pattern of the time series. As for the seasonal time series, the fault was simulated and hence there is complete control over the time span. Furthermore, the seasonal time series was crosschecked against the monitoring system to ensure no alarm had been raised.

3.1.3 Choice of models

The investigation was considered successful if anomalies could be detected by a model in at least two of the three chosen time series. The priority was thus to find one or more models which could detect anomalies in at least two out of the three chosen time series. To choose candidate models, a literature review of existing libraries and models was carried out. A candidate model had to provide a prediction of at least one time step into the future. The model also had to provide a prediction interval interval for each prediction it produced. The candidate models were prioritized based on how much configuration was necessary to implement it. Less configuration indicates less variables in the corresponding API model. Less variables in the API model indicates less domain knowledge needed in order to use the API, which is desirable from a usability perspective.

3.2 Software design

The software design related to how an existing monitoring system can be extended. An inductive approach can be used to develop an artifact. In an inductive approach, data is collected through qualitative methods and then analyzed to reach a conclusion [1]. The software design concerns creating multiple artifacts, and thus, an inductive approach was chosen. The goal was to create three artifacts. The first artifact was an overview diagram of the whole system. The second artifact were API models for each microservice. The last artifact was state diagrams of the communication between the microservices. To create the artifacts, a case study where an extension to Prometheus was developed. After the case study was completed, the extension was generalized to extend any monitoring system.

An iterative development method consists of mini-projects called iterations. An iteration consists of requirement analysis, design, implementation and testing [21]. The case study used an iterative development method. The case
The study consisted of one and a half iterations. During the first iteration an initial design was created from requirement analysis. The design was implemented and then tested. The tests was used to reveal additional requirements for the next iteration. The tests consisted of a separate restart of each component in the initial design. After a restart of a component, the system should continue operating. A successful test should reveal any flaws of the initial design which does not allow the system to continue operating after a restart of any component.

The half iteration consisted of another requirement analysis and design phase. The requirement analysis was based on the outcome of the test conducted in the first iteration. The design was then revised based on the findings in the requirement analysis phase.

An iterative development method was chosen to minimize the risk of not delivering a product. The outcome of each iteration should be a partial system [21]. The partial system can be viewed as one of the artifacts to be developed by the project. If the full one and a half iterations weren’t completed, there would have been at least a partial result. The partial result could then have been used for future development.

### 3.3 Project methods

To carry out the project, the process framework scrum was used as a basis. In scrum, work is time-boxed into one month or less, referred to as a sprint. Each sprint has a specific goal, referred to as the sprint goal. During a sprint, no changes affecting the sprint goal can be made [20]. A sprint is created through a meeting called sprint planning. The product backlog is a prioritized list of features, requirements etc. [24]. In the sprint planning, tasks are created from the product backlog. Tasks created are then part of the sprint backlog. The sprint backlog consists of tasks which are specifically worked on during the sprint.

The project used the scrum framework with one deviation, sprint goals may be changed. During the course of some sprints priority may change, which requires flexibility. Scrum was chosen as Pricerunner uses scrum. When the project was due to be handed over, it could continue as any other project at Pricerunner. Furthermore, scrum fits well with the iterative development method used.

To handover all code developed in this project, git was used. Git is a version control system. Version control means changes to a set of files are recorded [23]. An update to a set of files is referred to as a commit. A commit includes a message. Commit messages within this project used a general structure guideline. The structure consists of a subject line with a summary of changes, no longer than 50 characters, and a body containing a motivation of why a change was made [23].

### 3.3.1 Project prioritization

To prioritize the project, the MoSCoW method was used. MoSCoW is a method for prioritizing requirements based on a requirements specification. MoSCoW classifies requirements into four categories, must, should, could and won’t. Must specifies a requirement must be fulfilled. Should also specifies
requirements which must be fulfilled, but may be postponed until later. Could specifies requirements which are nice to have, but aren’t required for the project. Won’t specifies requirements which won’t be fulfilled during the project, but are relevant for the future [25].

The minimum result for a successful project was one investigation case study which disproved the hypothesis and a design which extended Prometheus. Table 1 describes the MoSCoW goals set up in the project.

<table>
<thead>
<tr>
<th>Must</th>
<th>Should</th>
<th>Could</th>
</tr>
</thead>
<tbody>
<tr>
<td>• One successful investigation</td>
<td>• Two successful investigations</td>
<td>• All investigations successful</td>
</tr>
<tr>
<td>• One iteration of development completed</td>
<td>• Additional half iteration completed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Data set pre processing</td>
<td></td>
</tr>
</tbody>
</table>

*Table 1: MoSCoW goals*

In the must column, one successful investigation case study and one iteration of development of the system was put. A stronger result of the project would be two successful investigation case study outcomes and a half iteration to refine the initial design. Hence, the should column included two successful case studies and a half iteration to refine the design. Furthermore, preprocessing of the training sets were included in the should column. The could column included successful outcomes from all three investigation case studies.
4 Time series fault prediction

This chapter presents the case studies conducted to disprove the hypothesis forecasting models cannot be used to detect anomalies in time series. The models used to conduct the case studies were rolling mean and Facebook Prophet (fbprophet). Each model has a dedicated section which briefly describes how the model was implemented.

4.1 Case study design

Each case study was made up by a model, a training set and test set. A training set consisted of historical metrics. The training set contained at least 14 days of metrics. The metrics were collected in 30 second intervals by Prometheus. Any potential missing samples were not taken into account when conducting the measurements. The amount of metrics used from the training set depended on the model used. A test set consisted of 24 hours of metrics in the future from the training set. Each test set began right after the corresponding training set ended. For example, if the training set had its last entry Friday 23:59:30, the test set would have its first entry Saturday 00:00:00.

The test sets consisted of two different scenarios. The first scenario was a test set of 24 hours without a fault occurring. The second scenario was a test set of 24 hours with at least one fault occurring. If no fault had occurred in the test set, a fault was simulated and is explicitly stated. These scenarios were then used to take measurements.

There were three types of measurements taken during a scenario. The first type was an alarm which was raised during a predefined time interval, referred to as a true alarm. The second type was an alarm which was raised outside any of the predefined time intervals, referred to as a false alarm. The third type was when an alarm wasn't raised during a predefined time interval, referred to as a missed alarm.

The measurements taken was used to disprove the hypothesis set up by the project. The hypothesis stated that forecasting models cannot be used to detect anomalous patterns in time series. The hypothesis was considered disproved if the following conditions were met for at least one of the investigated time series under both scenarios

- Raised alarms during all predefined time spans.
- Did not raise any false alarms
- Did not fail to raise any alarms during one of the predefined time spans

4.2 Training sets

The project used three training sets. Of the three training sets one was seasonal and two were stationary. The training sets had at least 14 days of data available. However, the amount of data used to train the models varied based on both the model and the case study. The test sets contain 24 hours of data and begins right after the corresponding training set ends. The seasonal time series had no real faults and therefore a fault was simulated in the test set.
Because the fault was simulated in the seasonal test set the same training set was used for both scenarios. The stationary time series used two different training sets as the tests sets were of different dates. In one of the test sets for the stationary time series multiple faults had occurred. In the other test set for the stationary time series no faults had occurred.

Figure 1 shows the training set used for the seasonal time series. The training set contained 14 days of data.

![Figure 1: Training set seasonal time series, 14 days of data. Image created by the author.](image)

The x-axis displays the date of the month followed by the time of day. The y-axis displays the requests per second observed at a specific point in time. As can be seen in the figure, the time series had a repeating seasonal pattern in 24-hour cycles. Furthermore, the training set contained outliers, where the requests per second spiked. For example, after 07:00:00 in the figure, there are two spikes which do not fit the pattern. In the first spike the value increased from 2.5 to 4.5 briefly. In the second spike the value increased from 3 to 6. The two spikes should be regarded as outliers rather than anomalies, as no fault occurred during the timespan. Because the training set contains some outliers but no anomalies, preprocessing may not be necessary.

Figure 2 shows the training set used for the stationary time series where no fault had occurred. The training set contained 14 days of data.
The x-axis display the date of the month followed by the time of day. The y-axis display the errors per second observed at a specific point in time. As can be seen in the figure, the time series is stationary around 0 with variation. There are spikes present in the training set. The spikes should be regarded as outliers. However, preprocessing may not be necessary for the training set.

Figure 3 shows the training set used for the stationary time series where a fault had occurred. The training set contained 14 days of data.
The x-axis display the date of the month followed by the time of day. The y-axis display the errors per second observed at a specific point in time. The training data is similar to the data in figure 2. However, after 13 00:00 in figure 3 there are a few things worth noting.

Firstly, the training set contain anomalies. The first anomaly is just before 15 00:00 in the figure, there are two spikes which last for an extended period of time. There are similar spikes after 15 00:00 but of shorter time spans. The spikes after 15 00:00 should be regarded as anomalies as well.

Secondly, after 15 00:00 the time series begin an increasing trend. In this case, there was a bug in a service where it reported the incorrect response code, which in turn caused the increasing trend. Hence, the increasing trend should not be regarded as a fault, but should be regarded as an anomaly. However, it does not affect the results as the reporting error is present in the test set as well. As the training set contain anomalies, preprocessing might yield better results.

4.3 Test sets

The project used four test sets to disprove the hypothesis. Two of the test sets were seasonal and two were stationary. The seasonal time series did not have any fault occurring and hence no anomaly present in the test set. As no fault had occurred, an anomaly was instead simulated. Each test set started just after the corresponding training set ended and contained 24 hours of data.

Figure 4 shows the test set of the seasonal time series were no fault had occurred.

![Figure 4: Test set seasonal time series, 24 hours of data. Image created by the author.](image)

The x-axis display the time of day. The y-axis display the requests per second observed at a specific point in time. The test set is another cycle of the seasonal time series. No alarm should be raised during this time period.

Figure 5 shows the test set of the seasonal time series were an anomaly is simulated. The anomaly was simulated by dividing the y-value by 2 and
adding a random number between -0.25 and 0.25 to each time point in a 2-hour interval.

![Seasonal Time Series with Anomaly](image)

**Figure 5:** Test set seasonal time series with anomaly, 24 hours of data. *Image created by the author.*

The x-axis display the time of day. The y-axis display the requests per second observed at a specific point in time. The anomaly is marked in red. As can be seen in the figure, the anomaly is a sharp drop in the metric value. At least one alarm should be raised during the time span of the red area. The anomaly was simulated with the following intention; the time series is an aggregate of metric values reported by two instances running the same service. If one of the instances fail and begin losing requests, a drop of around half should be expected as the requests are shared between the instances.

Figure 6 displays the stationary time series during a day without any fault occurring.

![Stationary Time Series without Anomaly](image)

**Figure 6:** Test set stationary time series without anomaly, 24 hours of data. *Image created by the author.*

The x-axis display the time of day. The y-axis display the errors per second observed at a specific point in time. The values are stationary around 0 with small variations. No alarms should be raised during this time period.

Figure 7 display the same time series during a day with multiple faults occurring.
The x-axis display the time of day. The y-axis display the errors per second observed at a specific point in time. The figure have five time intervals marked in red. During these time intervals faults had occurred. As can be seen in the figure, there are sharp increases in the observed values during the intervals marked in red.

The general theme shared between the anomalies in the test sets are either a sharp increase or decrease in values. During normal operation, the time series tend to follow reoccurring patterns. Hence, a model which detects sharp increases or decreases should be suitable to detect anomalies. However, how can a model determine whether there is a sharp increase or decrease in the metric value which wasn’t expected?

4.4 Prediction intervals

The preceding section raised a question which relates closely to determining whether an anomaly occurred or not. From the test sets it is evident that during a fault, there is generally a sharp increase or decrease in the metric value. Hence, to detect a fault these sharp increases or decreases should be noticed. To determine whether a sharp increase or decrease occurred, the first step is to forecast future values for the time series. There were two forecasting models used in the project, rolling mean and Facebook prophet (fbprophet). The rolling mean model forecasts the next value by calculating the mean of a set of values. The set of values span over a period of time, referred to as a time window. The fbprophet model forecasts a specified period of time. However, as forecasts concerns predicting the future, there is some amount of uncertainty. How much uncertainty a forecast contain can be described by constructing intervals around a predicted value, referred to as a prediction interval. The prediction interval indicates a future value is within the range of the interval with a specified probability. To construct a prediction interval for the rolling mean model, the standard deviation of the time window was used. Fbprophet provides prediction intervals out of the box for each predicted value $\hat{y}_t$. 

Figure 7: Test set stationary time series with anomalies, 24 hours of data. Image created by the author.
Figure 8 displays a forecast of 24-hours on the seasonal time series with an anomaly present using the fbprophet model. The prediction interval width was set to 0.99 and 7 days of training data was used.

The x-axis display the time of day. The red lines displays the prediction interval. The orange line displays the predicted values. The blue line displays the metric values. As can be seen in the figure, the prediction interval begin close to the predicted value, but over time grows wider. Because the prediction interval grows wider, the anomaly between 11:00 and 13:00 does not fall outside the prediction interval, and hence it won’t be noticed. To counteract the prediction interval growing wider, the forecast may be renewed. From the figure, it looked as if a 4-hour forecast will provide accurate prediction intervals. Figure 9 displays the same model under the same conditions, but with the forecasting being renewed every 4 hours.

Figure 8: 24 hour forecast fbprophet. Image created by the author.

Figure 9: Prophet forecast with renewed prediction intervals. Image created by the author.
The x-axis display the time of day. The red lines display the prediction interval. The orange line displays the predicted values. The blue line displays the metric values. As can be seen in the figure, the prediction interval is tighter when producing a forecast over 4 hours than 24 hours. In addition, the anomaly between 11:00 and 13:00 falls outside the prediction interval. However, there are also other values falling outside the prediction interval, as for example around 16:00. As no fault occurred during the time span, sending alarms when a value falls outside the prediction interval isn’t sufficient. This raises another problem; how can faults be detected without generating false alarms?

4.5 Anomaly detection

The preceding section raised the problem of how faults can be detected without generating false alarms. If false alarms are generated in addition to detecting the faults occurring, the hypothesis wasn’t considered disproved. In this section two approaches are presented where one of the approaches was used to conduct the case studies. In addition, the future work section presents another approach which is similar to the second approach presented in this section.

The first approach is to define a threshold, as for example the upper limit of a prediction interval. If observed metric values are above the threshold n times in a row, then send out an alarm. However, this approach inherits some of the problems with rules. For instance, if the observed metric values fluctuate between the threshold value, an alarm will not be sent out. Furthermore, the approach does not take into account how much larger or smaller the observed metric value is than the upper or lower threshold value. For example, an observed metric value which is five times larger than the upper threshold value might be a stronger indication of a fault occurring, rather than an observed metric value which is slightly larger than the upper threshold value.

The second approach takes into account how far from the closest threshold an observed metric value was. The second approach was also the approach used to conduct the case studies. The approach was to develop a scoring function. The scoring function update a score, referred to as the anomaly score, after each observed metric. If the observed metric value is above upper threshold or below lower threshold, the anomaly score is increased in relation to the closest threshold value. If the observed metric value is between the upper and lower threshold, the anomaly score is decreased in relation to the closest threshold value. To determine whether an anomaly had occurred, a predefined value is used, referred to as the anomaly score limit. If the anomaly score is above the anomaly score limit, an alarm was raised.

The scoring function was defined as follows;

Let \( t_l(t) \) and \( t_u(t) \) be the lower and upper threshold function at time \( t \) respectively. Let \( y_t \) be an observation at time \( t \). Let \( s(y_t) \) be the scoring function, then
The anomaly score is then updated as follows:

\[ a_s(t) = \begin{cases} 
  a_s(t-1) + s(y_t), & y_t > tu(t) \\
  a_s(t-1) \cdot s(y_t), & tu(t) \leq y_t \leq tu(t) \\
  a_s(t-1) + s(y_t), & y_t < tl(t) 
\end{cases} \]

When the observed metric is between the threshold, the anomaly score is multiplied by the scoring function. Generally, the score will be less than 1 if this relation is true, and hence the anomaly score will shrink. When the observed metric is smaller than the lower threshold, or larger than the upper threshold, the scoring function is added to the anomaly score. Hence, the anomaly score increase additively, and decrease multiplicatively. The multiplicative decrease rest on the assumption that the numerator is smaller than the denominator in the scoring function.

The score will increase based on how far from the threshold the observed metric was. Hence, a sharp increase or decrease in value of the observed metric which breaches a threshold will be reflected by the scoring function. In turn, the sharp increase or decrease will be quickly noticed, assuming the values are somewhat small. If the observed metric is within a predicted range, it will decrease multiplicatively based on how far from the closest threshold it was. Assuming the numerator and denominator is approximately equal in the scoring function, the anomaly score will approximately be multiplied by 1 and added by 1 if an observed metrics fluctuate closely between the limit of a threshold. Hence, metrics which fluctuate between the limit of a threshold should eventually be noticed.

4.6 Data set preprocessing

The prediction intervals produced by fbprophet are sensitive to outliers. Hence, it may be desirable to use preprocessing on the training sets to remove outliers. Measurements made for the prophet model were taken with two
preprocessing techniques and without any modifications done to the training sets.

The first technique removed all observations more than 1.96 standard deviations from the mean of the whole training set. The number 1.96 relates to the 68-95-99 rule. The 68-95-99 rule states 95% of all observations from a normal distribution are approximately two standard deviations from the mean. The exact number for 95% is 1.96 based on the standard normal table. Note, the time series are not necessarily outcomes from a normal distribution. Hence, the 68-95-99 rule is applied as a general guideline. The first technique will be referred to as $1.96\sigma$. Figure 10 displays the stationary training set after preprocessing with the $1.96\sigma$ technique.

![Figure 10: Stationary training set preprocessed by $1.96\sigma$. Image created by the author.](image)

The x-axis display the day and time of day. The y-axis display the observed errors per second. As can be seen in the image, the sharp increases have been removed. The maximum value is now around 3 instead of around 30 as in figure 3.

The second technique was to remove outliers by isolation forest, an anomaly detection algorithm. The library scikit-learn provides an implementation for the isolation forest which was used for the measurements. One parameter to the implementation was explicitly set. The contamination parameter, which indicates the proportion of outliers in the data set, was set to 0.05. There was also one modification to the x-values in the training set. The training set was obtained from Prometheus, hence the x-values in the training set are represented by Unix timestamps. The x-values in the training set were converted to the remainder after division by 86400 (one day in seconds). The idea was to group each point based on the time of the day the point was observed. The assumption was as follows; each day a value observed is approximately equal to the value observed at the same time during the previous day. Because the main idea behind the isolation forest algorithm is to isolate anomalous points, it might be desirable to group points in this fashion.
Figure 11 displays the stationary training set after preprocessing with isolation forest.

![Image of stationary training set preprocessed by isolation forest](image_url)

Figure 11: Stationary training set preprocessed by isolation forest. Image created by the author.

The x-axis displays the day and time of day. The y-axis displays the observed metric value. As can be seen in the image, the sharp increases have been removed. The maximum value is now around 2 instead of around 30 as in figure 3.

As no fault had occurred in the seasonal requests time series, a fault was simulated. To simulate the fault in the seasonal time series, values within a time interval were divided by 2 and added with a random uniformly generated number between -0.25 and 0.25 to introduce some noise.

### 4.7 Implementation

Each time series were tested five times for the fbprophet and pre-processing technique to ensure consistency of the results. The rolling mean model was tested two times per time series to ensure the same result was observed. The case study was divided into two phases, training and test phase. During the training phase, the training set was first preprocessed if applicable. The training set was then fed to the chosen model to make a forecast. During the test phase the test set was processed and compared to the forecast. If a metric value from the test set was within the prediction interval, it was appended to the training set. If a metric value from the test set breached a threshold, and the anomaly score was above half of the anomaly score limit, the metric value was not appended to the training set. The training sets were appended with the processed test set metrics as this simulates how the system would work in a real-life setting. Furthermore, the forecast will lose accuracy over time and should be updated. Lastly, the anomaly score limit was set to 10 and .
4.7.1 Rolling mean

To train the rolling mean model, seven days of training data was used. Time window sizes, referred to as $n$, of 480, 960 and 1440 were used, this represents four, eight and twelve hours of data respectively. The rolling mean model was first run through the seven days of training data. Upon completion, the trained model was run through the test set. The library pandas was used to compute the rolling mean and standard deviation. The threshold functions were then defined as follows

\[ tu(t) = \mu_{t-1} + 3 \cdot \sigma_{t-1} \]
\[ tl(t) = \mu_{t-1} - 3 \cdot \sigma_{t-1} \]

Where $T$ is the training set and $n$ is the number of elements in $T$. Three standard deviations were chosen based on the guideline from the 68-95-99 rule to represent approximately 99% certainty. The rolling mean model forecasted one time step into the future and hence was updated after each metric from the test set was processed.

4.7.2 Facebook prophet

To train the fbprophet model, 7 and 14 days of training data was used. Because both the training and test set consisted of Unix timestamps, each timestamp was converted to a date format of YYYY-MM-DD HH:MM:SS. The confidence interval parameter passed to the fbprophet model was set to 0.99. The models were then used to forecast four and eight hours respectively in 30 second intervals. A data point from the test set was matched to the closest forecasting point by time distance. The time distance was at most 15 seconds. If no forecasting point matched the 15 second condition, the forecast was updated. The updated forecast was produced with the updated training set. Note, the training set contained at most 7 or 14 days of data. Hence, once an observation was added, the first element in the training data was removed if the training set contained more than 7 or 14 days of data. The threshold functions were defined by the $\hat{y}, lower$ and $\hat{y}, upper$ parameters provided by fbprophet.
5 Software design

This section presents the development process of the project. An initial design was first created. The initial design used Prometheus as data source. The initial design was then implemented and tested. The purpose of tests was to detect additional requirements and potential drawbacks of the initial design. The outcome of the tests were then used to create a general design. The general design is presented in the results section and is based on a microservice architecture. There were two microservices designed, the anomaly detection service and the retrieval service. Section 5.1 presents the design process. Section 5.2 presents the initial design. Section 5.3 presents the anomaly detection service. Section 5.4 presents the retrieval service. Section 5.5 presents the test results. Section 5.6 presents a discuss which includes the improvements made to the initial design.

5.1 Design process

The design process was broken down into three stages. Firstly, requirements were defined and analyzed. Secondly, area of responsibilities were clearly defined based on the requirements. Lastly, each area of responsibility was analyzed to determine boundaries between services.

The initial system specification was as follows:

The system should detect and report anomalies within time series. To detect anomalies, models must first be trained with historical metrics from Prometheus. After the model has been trained, new metrics must be passed in to detect anomalies. When an anomaly is detected, a notification should be sent out.

Three requirements can be defined;
- The system must build models with training data and detect anomalies in new observed metrics.
- The system must be able to retrieve historical and new metrics continuously from Prometheus.
- The system must be able to notify when it detects an anomaly.

From these requirements, three areas of responsibilities can be defined. The three areas are retrieval of metrics, detecting anomalies and notifying when anomalies occur. Notifying in this case refers to the decision of when to send out an alert. Services for sending notifications were already available at Pricerunner, and hence, a service for sending notifications was not implemented.

Detecting and notifying when an anomaly occurs is closely related to one another. The service detects whether an anomaly occurs. If an anomaly occurs it should send out a notification and keep state of notifications sent out for a specific time series. Likewise, when a time series leaves a state of anomaly, the service should send out a notification that the alarm has cleared. Hence, detection and notifying behavior should be grouped into one service. To change the behavior of when to notify, only one service has to be changed.
Retrieving metrics from Prometheus and anomaly detection could potentially be grouped into the same service. Grouping both retrieval and anomaly detection into the same service would simplify the logic of the service. The logic is simplified as the service has complete access to its own state. For example, if a time series is missing, the service has knowledge of this fact and can directly add the missing time series. Contrary, if retrieval and anomaly detection are split into two separate services, they have to communicate about the state. If the anomaly detection service is missing a time series, it has to tell the service responsible for retrieval. Hence, the application logic is made more complex. However, a service which is responsible for both retrieval and anomaly detection will be tightly coupled to the Prometheus API. If the Prometheus API changes, the Prometheus API client have to be updated. In turn, the combined service have to be redeployed to update the Prometheus API client. Hence, retrieval of metrics and anomaly detection was split into two services. The first service was the retrieval service, the second service was the anomaly detection service. In the first iteration, the service for anomaly detection exposed an API. The API contained methods to add time series to be monitored and adding new observations in a monitored time series. The retrieval service consumed both Prometheus and the anomaly detection service API. The benefit with this approach is that the anomaly detection service is decoupled from the metric data source. Hence, the anomaly detection service is generalized. The retrieval service is however tightly coupled with the data source.

5.2 Initial design

Based on the design process, two services were created, the retrieval and the anomaly detection service. Figure 5 presents an overview of the initial design.

![Initial design](image.png)

*Figure 12: Initial design. Image created by the author.*

An arrow in the figure indicates a dependency. The source of the arrow is a component which depends on another component in the destination of the arrow. In the figure the retrieval service depends on both Prometheus and the anomaly detection service. Both Prometheus and the anomaly detection service expose APIs which are used by the retrieval service.

The first dependency of the retrieval service was Prometheus. The retrieval service continuously retrieve time series data from Prometheus. The retrieval service could either be pre-configured by a configuration file or expose an API. The API should allow time series to be added to a list of metrics to retrieve
from Prometheus or to be deleted from the same list. In this project, the retrieval service used a configuration file. A configuration file was chosen as the retrieval service holds no state other than the time series to retrieve. To update the time series to retrieve, the configuration file was updated and the retrieval service was restarted.

The second dependency was to the anomaly detection service. After time series data was obtained from Prometheus, the time series data was converted to the API model of the anomaly detection service. The API model of the anomaly service is described in the next section. Hence, the retrieval service has tight coupling to both Prometheus and the anomaly detection service. If either Prometheus or the anomaly detection service changes their API model, the retrieval service have to be updated as well. However, the anomaly detection service is decoupled and does not depend on any specific data source.

5.3 Anomaly detection service

The anomaly detection service was responsible for detecting and notifying of anomalies. This project used time series to detect anomalies. First a forecast was produced based on historical data from a time series by using a forecasting model. New observations from the time series were then compared to the forecast in order to determine if an anomaly had occurred. To detect anomalies, the anomaly detection service exposed two API functions during the first iteration.

The first function was to add a new time series to monitor. Table 2 describes the function of adding a new time series to monitor.

<table>
<thead>
<tr>
<th>Route</th>
<th>/v1/model POST</th>
</tr>
</thead>
</table>
| Body parameters | `{  
  "model": "string"  
  "windowSize": number (optional)  
  "forecastPeriods": number (optional)  
  "timeSeries": "string"  
  "sampleInterval": number (optional)  
  "trainingSetSize": number (optional)  
  "trainingSet": [number]  
  "trainingSetIndex": [number]  
}` |
| Response | Status code: 201, the resource was successfully created.  
Status code: 400, the request was invalid. For example, a required field is missing. Sample body response: `{“message”: “model must be specified”}`  
Status code: 409, the specified time series already exists |

Table 2: API model for adding a new time series

The Route column specifies the Uniform Resource Identifier (URI) and HTTP method. The body parameters specifies the API model. The body parameters
are set in the HTTP body. The body parameters specifies the configuration of the time series. The model parameter specifies which model should be used (e.g. rolling mean etc.). The windowSize specifies how large the window should be for a rolling mean model. The forecastPeriods parameter specifies how many periods into the future the model should predict before updating itself. The timeSeries parameter is a unique identifier. The sampleInterval specifies how frequent samples are taken in seconds. The trainingSetSize specifies how many samples the training set should contain when updating future models. The trainingSet parameter is an array with metric values. The trainingSetIndex is an array of Unix timestamps. The timestamps represent the time of which the metric at the same index in the trainingSet array was observed. The response specifies the possible HTTP responses.

The second function was to add new observations to an existing monitored time series. Table 3 describes the function of adding new observations to a time series.

Route | /v1/model/observe
--|--
PUT

Body parameters

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
| "timeSeries": "string"
| "values": [number]
| "timeObserved": [number] |

Response

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
| Status code: 202, the observations was added for processing.
| Status code: 400, the request was invalid. As for example, the arrays values and timeObserved were of different lengths. Sample body response:
| {"message": "input arrays must be of equal length"} |
| Status code: 404, the specified timeSeries does not exist |

Table 3: API model for adding new observations

The Route column specifies the URI and HTTP method. The body parameters specifies the API model. The body parameters are set in the HTTP body. The timeSeries parameter indicates to which time series the observation belong. The values parameter is an array of metric values. The timeObserved parameter is an array of times the metric values were observed. The response specifies the possible HTTP responses.

During the first iteration the anomaly detection service kept the training set of each time series in memory. Each observation added to a time series were appended to the training set. If the size of the training set exceeded the trainingSetSize parameter, the oldest value were removed from the training set until the actual size of the training set and the parameter were equal. This approach was initially used as the forecasts had to be renewed. Section 4.4 describes the reasoning behind the forecast renewal for the fbprophet model. The rolling mean model had to be updated after each observation.
5.4 Retrieval service

In the project, the retrieval service was responsible for retrieving time series from Prometheus. The time series was converted to the API model of the anomaly detection service. The anomaly detection service was then updated with the converted time series. The purpose of the retrieval service was to decouple the anomaly detection service from Prometheus. In the project, the retrieval service used a configuration file which specified which time series should be monitored.

In the first iteration, upon start up, the retrieval read the contents of the configuration file. The configuration file also included which forecasting model to use with the time series. It then retrieved time series data from each of the configured time series and converted them to the API model of the /v1/model route exposed by the anomaly detection service. The retrieval service then called the /v1/model route exposed by the anomaly detection service with the converted time series. After the time series was added for monitoring, the retrieval service would then in intervals collect single observations for each configured time series from Prometheus. Each observation was converted to the API model of the /v1/model/observe route exposed by the anomaly detection service. The retrieval service then called the /v1/model/observe route to add the observation. Figure 13 describes the initial state diagram.

![Figure 13: Initial state diagram. Image created by the author.](image)

The retrieval service starts of by initializing, this includes loading the configuration. It then add each forecasting model to the anomaly service. Lastly, it continuously adds observations. The retrieval service was initially naive as it didn’t take into the account response code from the anomaly service. Improvements were left for the next half iteration of the project.

5.5 Test result

The last part of the first iteration consisted of a test where each component was restarted independently. A restart of each component was chosen as it simulates two scenarios. The first scenario is when a component crash and is later started again. The second scenario is of a deployment where one of the services is restarted to update the code. The whole system should continue operating after either a crash or a new deployment.

The retrieval service could be restarted without causing the system to stop. The retrieval service had a configuration file with time series to retrieve and
update the anomaly detection service with. The whole system was able to continue operating as the state of the retrieval service was stored in the configuration file. Hence, when the retrieval service was restarted, the state was restored.

The anomaly detection service could not be restarted without the whole system stopping. When the application was restarted, the state about which time series were monitored and the corresponding forecasting models was lost. As all time series were lost, 404 responses were returned to the retrieval service. Hence, the retrieval service got stuck on adding observations to a time series which did not exist in the anomaly detection service.

5.6 Design refinements

The project completed one and a half iterations. During the half iteration, the initial design was improved based on the test results and gained experience. The test results raised one problem with the initial design. The problem was the anomaly detection service could not be restarted without the whole system stopping. The anomaly detection service lost state about which time series were monitored after a restart because this state was not persisted. Hence, the retrieval service attempted to add observations to time series which did not exist in the anomaly detection service. To solve this problem, the retrieval service should attempt to create a new time series when obtaining a 404 status code response from the anomaly detection service. The retrieval service should first fetch the time series data from Prometheus, and then add the new time series to the anomaly detection service. If the anomaly detection service responds to this request with a 400 or 409 status code, the process attempting to add the time series should exit in the retrieval service. A 400 response indicates the configuration of the time series is incorrect. A 409 response indicates there is a race condition as another process added the time series.

The test results also raised one potential improvement with the initial design. After the anomaly detection service was restarted, it lost all predictions made for each time series. The improvement would then be to persist all predictions and the anomaly score of the corresponding time series to a database. The anomaly service could then be restarted and continue operating as the state is persisted in the database. As for the rolling mean model, the training set should be saved instead. Because the rolling mean model only makes a prediction of one time step ahead, saving the prediction itself is of little use. The training set would have to be obtained from Prometheus after each observation and then the mean and standard deviation can be calculated. Due to this reason, persisting the training set is preferred when using the rolling mean model. Another benefit of this approach is that there could be multiple instances of the anomaly detection service, and hence the anomaly detection service could scale horizontally. A drawback is increased complexity as another component, the database, is added to the design. Hence, the database addition is left as an optional component in the final design.
There were two additional improvements of the initial design which were not based on the test results. The first improvement was based on the concept of lag between the current time and the predictions. For example, the rolling mean model makes a prediction for the next time step. If there is a gap of time between the predicted time step and the input observation, the prediction might be inaccurate. If there is a sufficient gap of time between the current observation and the predictions, the anomaly service API should return an error. If the gap is not large enough, the anomaly detection service should update the predictions. The retrieval service should then react to the error by attempting to update the forecasting model with a new training set. The model route in the anomaly service was updated to include an HTTP PUT method which was used to update the predictions. In addition, another state in the state diagram was added to the retrieval service.

The second improvement was the API model of the anomaly detection service. The route for adding forecasting models was too broad. Arguments which are not used by some forecasting models may be provided. For example, when adding a rolling mean model, only windowSize and trainingSetSize are applicable of the optional parameters. To decrease the amount of API parameters to the route for adding forecasting models, the route was split up into separate routes for each forecasting model. Hence, the updated API model consists of one route for the rolling mean and one route for the fbprophet model. A specific route per model increases cohesion. In order to add another model to the anomaly detection service, a new route is added rather than editing the existing parameters for a specific route. The updated API model is presented in appendix A.
6 Results

This chapter presents the results from the time series fault predictions and software design. Section 6.1 presents the resulting measurements from chapter 4. Section 6.2 presents the software design based on the development process from chapter 5.

6.1 Measurements

Results from each measurement are attached in appendix B. This section presents the outcome for each investigated time series. As each model was measured five times, the average of each measured alarm type for the specific model and implementation is presented. For precise results of each measurement, the reader is referred to appendix B.

Table 4 summarizes the results of the stationary time series without anomalies present.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training data</th>
<th>Preprocesesing</th>
<th>False alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>1.96σ</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>Isolation forest</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>1.96σ</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>Isolation forest</td>
<td>0</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>12 hours</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>8 hours</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>4 hours</td>
<td>N/A</td>
<td>0</td>
</tr>
</tbody>
</table>

*Table 4: Measurement results, stationary time series without anomalies* 

The model column defines the model used. The training data column describes the amount of training data used in time format. The preprocessing column describes the preprocessing method applied on the training data set. The false alarms column is a count of the amount of alarms raised outside of the predefined time intervals.

Table 5 summarizes the results of the stationary time series with anomalies present.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training data length</th>
<th>Preprocesesing</th>
<th>True alarms</th>
<th>False alarms</th>
<th>Missed alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>None</td>
<td>7,8</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>1,96σ</td>
<td>17,2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

38
<table>
<thead>
<tr>
<th>Model</th>
<th>Training data</th>
<th>Preprocessing</th>
<th>False alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>1,96σ</td>
<td>3,8</td>
</tr>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>Isolation forest</td>
<td>4,6</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>1,96σ</td>
<td>0,4</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>Isolation forest</td>
<td>1,2</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>12 hours</td>
<td>N/A</td>
<td>1</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>8 hours</td>
<td>N/A</td>
<td>1</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>4 hours</td>
<td>N/A</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5: Measurement results, stationary time series with anomalies

The model column defines the model used. The training data length column describes the amount of training data used in time format. The preprocessing column describes the preprocessing method applied on the training data set. The true alarms column is a count of the amount of alarms raised during one of the predefined time intervals. The false alarms column is a count of the amount of alarms raised outside of the predefined time intervals. The missed alarms column is a count of how many of the predefined time intervals the model did not raise any alarms during.

Table 6 summarizes the results of the seasonal request per second time series without an anomaly present.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training data</th>
<th>Preprocessing</th>
<th>False alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>1,96σ</td>
<td>3,8</td>
</tr>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>Isolation forest</td>
<td>4,6</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>1,96σ</td>
<td>0,4</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>Isolation forest</td>
<td>1,2</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>12 hours</td>
<td>N/A</td>
<td>1</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>8 hours</td>
<td>N/A</td>
<td>1</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>4 hours</td>
<td>N/A</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6: Measurement results, seasonal time series without anomalies

The model column defines the model used. The training data length column describes the amount of training data used in time format. The preprocessing column describes the preprocessing method applied on the training data set. The false alarms column is a count of the amount of alarms raised outside of
the predefined time intervals. Only false alarms are applicable as the time series is used as validation.

Table 7 summarizes the results of the seasonal request per second time series with an anomaly present.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training data</th>
<th>Preprocessing</th>
<th>True alarms</th>
<th>False alarms</th>
<th>Missed alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>None</td>
<td>1</td>
<td>3.6</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>1,96σ</td>
<td>1</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>14 days</td>
<td>Isolation forest</td>
<td>1</td>
<td>4.8</td>
<td></td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>None</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>1,96σ</td>
<td>1</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>Prophet</td>
<td>7 days</td>
<td>Isolation forest</td>
<td>1</td>
<td>1.2</td>
<td>0</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>12 hours</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>8 hours</td>
<td>N/A</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Rolling mean</td>
<td>4 hours</td>
<td>N/A</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 7: Measurement results, seasonal time series with anomaly

The model column defines the model used. The training data length column describes the amount of training data used in time format. The preprocessing column describes the preprocessing method applied on the training data set. The true alarms column is a count of the amount of alarms raised during one of the predefined time intervals. The false alarms column is a count of the amount of alarms raised outside of the predefined time intervals. The missed alarms column is a count of how many of the predefined time intervals the model did not raise any alarms during.

6.2 Software design

This section presents the resulting software design which is based on the artifacts created by the case study in chapter 5. It presents a component diagram of the whole system. It presents a state diagram of the retrieval service. The API of the anomaly service is presented in appendix A. Figure 14 displays the design of the whole system.
An arrow in the figure indicates a dependency. The source of the arrow is a component which depends on another component in the destination of the arrow. The metric data source collects metrics from an application. The retrieval service reads all metrics to poll from the configuration data source upon startup. The configuration data source in the project was a configuration file. The retrieval service polls the configured metrics from the metric data source. The metrics are then converted by the retrieval service to the API model of the anomaly service and sent to the anomaly detection service. The anomaly service may include a data source where the training sets or predictions, depending on the model, are stored. However, this is an optional extension.

Figure 15 displays the state diagram of the communication from the retrieval service to the anomaly detection service.
The retrieval service begins in the initialize state where it loads configuration from the configuration data source. It then moves to the add observation state where it will continue adding observations until it either encounters a status code 404 or 500 from the anomaly service. If a 404 status code is encountered, it moves to the add time series state where the time series is added to the anomaly service. If a status code 400 or 409 is encountered, the thread should exit as this is an indication of misconfiguration or a race condition. If status code 201 is returned, it moves back to the add observation state. If status code 500 is encountered in the add observation state, it should move to the update time series state. In the update time series state it updates the existing time series in the anomaly service. If status code 400 is encountered, the thread should exit as this is an indication of misconfiguration and thus, it can not continue. If status code 201 is encountered, the state should return to add observation.
7 Discussion

This chapter discusses the results presented in chapter 6. Section 7.1 discusses fault prediction. Section 7.2 discusses the development method. Section 7.3 describes future work. Section 7.4 concludes the project.

7.1 Fault prediction

The investigation conducted in section 4 sought to disprove the hypothesis forecasting models cannot be used to detect anomalous patterns in time series. A model was said to accurately predict anomalies if it raised an alarm during all predefined time intervals without raising any false alarms. Accurate prediction was a necessary condition to consider the hypothesis disproved. Furthermore, the model should have provided consistent results for the amount of missed and false alarms to improve reliability.

The rolling mean model could detect anomalies in one out of the three time series investigated. However, it could only predict anomalies accurately in one case. The rolling mean model with a window size of twelve hours could accurately predict all anomalies in the stationary time series. In contrast, while processing the stationary time series, the prediction intervals grew as more anomalous metrics were added to the training set. Metrics far from the mean increased the standard standard deviation. Hence, the rolling mean model performs worse overtime if anomalous metrics are added to the training set. Furthermore, the rolling mean model did not work at all for seasonal time series. The rolling mean model disproves the hypothesis in one case. However, the evidence was considered too weak.

Facebook Prophet (fbprophet) could detect anomalies in both time series investigated. Fbprophet could predict anomalies accurately depending on the training data. Fbprophet predicted anomalies accurately in the stationary time series if the training set was preprocessed. The stationary time series contained anomalies, and hence the predictions were improved when some of the anomalies were removed. However, fbprophet did not predict anomalies accurately if the seasonal time series was preprocessed. The seasonal time series training data set did not contain anomalies. As there were no anomalies present, only outliers were removed which led to tighter prediction intervals. Tighter prediction intervals in turn led to the false alarms. The fbprophet model disproves the hypothesis. However, preprocessing may have to be applied depending on the training data set.

Preprocessing via $1.96\sigma$ gave better results than isolation forest. Isolation forest in most cases removed more data points than $1.96\sigma$. Because more data points were removed, fbprophet produced tighter prediction intervals after preprocessing with isolation forest than $1.96\sigma$. Tighter prediction intervals have a consequence which affected the results. As the prediction interval tend towards the predicted value, the anomaly score will tend towards being multiplied by 1 with the scoring function used. Hence, the anomaly score will decrease slowly, which may in turn lead to false alarms. A solution might be to
multiply by a constant rather than the fraction of the closest prediction interval and the predicted value. However, refinements to the scoring function is left as future work.

The results had some variation in the amount of true alarms raised. However, the amount of missed and false alarms were consistent for all measurements were preprocessing was applied. The amount of missed and false alarms varied for one measurement when preprocessing wasn't applied. Hence, the results are considered reliable.

7.1.1 Validity

Internal validity refers to whether the correct thing is being measured. In section 3.1.2 a few factors which increased internal validity were described. For example, the faults in the stationary time series were crosschecked with the alarms raised from a monitoring system. In this case, the alarms raised from the monitoring system were based on checks. A check in this case was a request. An alarm was raised if two requests failed within a 30 second interval from two different countries. However, there are a few problems which decrease internal validity.

Firstly, the checks are not immune to raising false alarms. The main problem is this project have not clearly defined what a fault is. For instance, there might be a temporary problem which caused the two requests to time out. However, two requests timing out is not necessarily an indication of a fault occurring. The requests might time out due to problems with the client making the request.

Secondly, the checks might not always raise alarms when there is a fault occurring. For example, 50% of the requests might result in errors. Generally, this would be a strong indication of a fault occurring. However, the checks might always be in the 50% of the requests which succeed. In turn, no alarms would be raised as the checks would pass.

Thirdly, Prometheus was used as the only data source used in this project. To increase internal validity, multiple systems which collect metrics could be used. The case studies should then use the same metrics collected by multiple sources and obtain the same results to increase internal validity.

Lastly, the time spans set up where an alarm raised would be counted towards a true alarm might have been too long. For instance, a model could raise an alarms towards the end of an anomaly and still have it count as a true alarm. In practice this would have little use as the system would have raised the alarm towards the end of a fault. Hence, the time spans could have been shorter.

External validity refers to if the results can be generalized. There are at least two factors which decrease external validity. Firstly, time series with a resolution of 30 seconds were used. No other resolution was tested. Preferably the results should generalized to time series of any resolution. To increase external validity, time series of different resolutions should have been used to train the models. In addition, measurements where the training set a have different resolution than the test set should also be conducted. For example,
the training set could have a resolution of five minutes and the test set a resolution of 30 seconds.

Secondly, the scoring function developed is limited. The scoring function requires specifically that a predicted value and a prediction interval is provided. This limits the amount of models which could be used to detect anomalies. For example, the isolation forest algorithm can’t be used with the scoring function developed. The isolation forest algorithm does not provide a predicted value nor a prediction interval.

7.2 Development method

The project managed to complete one and a half iterations of development. One iteration was put into the must column in the MoSCow prioritization. The half iteration to refine the design was put in the should column. As both the must and could goals were completed the iterative development method was an appropriate choice. Even if only a half iteration was completed and the must goal wasn’t met, there would have been some results to present. Hence, the iterative development method was an overall good choice for the project.

The testing done during development was very limited. The only tests done was to restart the services separately and to take measurements. No unit tests were written as these were replaced by the measurements. Because testing was limited, external validity is decreased. External validity is decreased as there might be defects within the design which haven’t been taken into account. Two examples are given. Firstly, the option of preprocessing training sets have not been taken into account. Suggestively, this feature should be implemented by the anomaly detection service. The anomaly detection service should then add another field for the models. The field indicates if the training set should be preprocessed and by which method.

Secondly, the API model for fbprophet does not take into account the resolution of the time series. This is a serious defect as the assumption that all time series will have the same resolution is incorrect. A solution is to add another field to the API model of fbprophet to include the resolution of the time series. Due to the second point, external validity is decreased as the API model can only handle time series one resolution.

Another testing scenario to consider is to turn off one of the services for an extended period of time, for example 24 hours. After the 24 hours have passed, then start the turned off service again and see if the system continues to work.

The results presented in section 6.2 provides a basis for future development of an extension to a monitoring system. The extension provide capabilities to alert on anomalous patterns in time series. Future development could continue using iterative development or another method. Furthermore, the anomaly detection does not depend on a specific monitoring system. In fact, it does not depend on a monitoring system at all. Hence, the anomaly detection service is generalized. On the other hand, the retrieval service will always require specific implementation depending on the data source used.

7.3 Future work
This project investigated a limited set of time series and models. A more extensive report could include more models and time series. Furthermore, the time series could be of longer time spans and of different resolutions. A more extensive report could also use multiple time series as input to predict another time series.

The presented scoring function does not take into account how large the values are. For instance, if the predicted value is 10000, the upper threshold 12000 and the observed value 14000, the anomaly score will increase by approximately 1.17. As the observed value is 2000 units larger than the upper threshold, it might be seen as a sharp increase. However, this isn’t reflected by the anomaly score increase. Another approach is to compute the anomaly score is by distances from the predicted value. For example, the distance formula is defined as
\[ d(x_1, x_2) = |x_1 - x_2| \]

Then the scoring function can be defined as
\[ s(y_t) = \frac{c_1 + d(y_t, \hat{y}_t)}{c_2 + d(|tu(t)|, \hat{y}_t)} \] for \( y_t \geq \hat{y}_t \) \& \( d(tu(t), \hat{y}_t) + c_2 \neq 0 \)

\[ s(y_t) = \frac{c_1 + d(tu(t), \hat{y}_t)}{c_2 + d(y_t, \hat{y}_t)} \] for \( y_t < \hat{y}_t \) \& \( c_2 + d(y_t, \hat{y}_t) \neq 0 \)

The scoring function will update the score relative to the distance from \( \hat{y}_t \). In addition, the scoring function assumes the prediction interval is symmetrical. In the provided example the score now becomes \( \frac{4000}{2000} = 2 \) which might be a more correct increase as it reflects the distance of the observed value to the predicted value. Future work could compare this scoring function to the presented function by taking similar measurements and comparing the outcomes.

In section 1.4.1 the concept of hardware redundancy was introduced and how it could be reduced if faults could be predicted. The ability to predict faults ahead of time was never explored within this project. Future work could compare the time to detect a fault by anomaly detection versus regular checks and rules.

7.4 Conclusion

This project presented an investigation which disproves the following hypothesis; forecasting models cannot be used to detect anomalies in time series. Because the hypothesis was disproved, the following theory is strengthened; forecasting models can be used to detect when anomalous patterns occur in time series. However, the theory remains unproved. The project also presented a software design which extends a monitoring system. Hence, the problem statement can be answered as follows: A monitoring system can be extended by the design presented in 6.2. The Facebook Prophet forecasting model can be used in some cases to detect faults in a time series as shown by the results in section 6.1. To detect faults preprocessing may have to
be applied on the training sets. Preprocessing should be applied when the training sets contain anomalous data points as these lead to inaccurate predictions. However, the ability to detect faults relies on the unproved theory which claim forecasting models can be used to detect when anomalous patterns occur in time series.
References


Appendix A

This appendix presents the API models as part of the results from section 6.2. Table 8 presents the API model for creating a new time series prediction using the fbprophet forecasting model. The route row describes the URI and method combination. The body parameters describes the parameters the API expects. The response row describes the possible responses from the API.

<table>
<thead>
<tr>
<th>Route</th>
<th>/v1/model/fbprophet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PUT, updates an existing time series</td>
</tr>
<tr>
<td></td>
<td>POST, creates a new time series</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Body parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;timeSeries&quot;: &quot;string&quot;</td>
</tr>
<tr>
<td>&quot;forecastPeriods&quot;: number (optional)</td>
</tr>
<tr>
<td>&quot;sampleInterval&quot;: number (optional)</td>
</tr>
<tr>
<td>&quot;trainingSetSize&quot;: number (optional)</td>
</tr>
<tr>
<td>&quot;trainingSet&quot;: [number]</td>
</tr>
<tr>
<td>&quot;trainingSetIndex&quot;: [number]</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status code: 201, the time series was successfully created.</td>
</tr>
<tr>
<td>Status code: 400, the request was invalid. As for example, a required field is missing. Sample body response:</td>
</tr>
<tr>
<td>{“message”: “timeSeries must be specified”}</td>
</tr>
<tr>
<td>Status code: 409, returned on POST requests when the specified time series already exists</td>
</tr>
</tbody>
</table>

Table 8: API model for creating a new time series using the prophet model

The timeSeries parameter specifies a unique id which the predictions will be stored under. The forecastPeriods specifies how far into the future the model should make a prediction. The sampleInterval specifies the resolution of the time series and indicates the interval of which the predictions will be made. The trainingSetSize parameter specifies the amount of data to use when updating the predictions. The trainingSet parameter is the training set used to train the model. The trainingSetIndex is an array with time stamps of which the training set corresponds to. The trainingSetIndex and the trainingSet must be of the same size.

Table 9 presents the API model for creating a new time series prediction using the rolling mean forecasting model. The route row describes the URI and method combination. The body parameters describes the parameters the API expects. The response row describes the possible responses from the API.

<table>
<thead>
<tr>
<th>Route</th>
<th>/v1/model/rolling_mean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PUT, updates an existing time series</td>
</tr>
<tr>
<td></td>
<td>POST, creates a new time series</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Body parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;timeSeries&quot;: &quot;string&quot;</td>
</tr>
<tr>
<td>&quot;windowSize&quot;: number (optional)</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
Table 9: API model for creating a new time series using the rolling mean model

The API model is similar to the fbprophet API model. The key difference, other than some values which are omitted, is the windowSize parameter. The windowSize parameter specifies how large the rolling mean window should be.

Table 10 describes the API call to obtain all timeSeries which is present in the server.

Table 10: API function to obtain all time series known to the server

The returned values is a list of the time series present in the server.

Table 11 describes the API model to add a new observation to a time series.

Table 11: API model for adding an new observation
The :id part of the URI in the route is the time series id specified by the

timeSeries argument when creating or updating a model. The values

parameter is a list of values. The timeObserved parameter is a list of time

stamps of which the values were observed.
Appendix B

This appendix present the outcomes of all measurements taken. In the time series without anomalies, all alarms raised were considered false alarms. Hence, only false alarms were measured in the time series without anomalies. In the time series with anomalies the amount of false, true and missed alarms were measured.

**Stationary time series without anomalies**

Run x: Number of false alarms raised

**Prophet 7 days no preprocess**
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

**Prophet 7 days 1.96 sigma preprocess ca. 200 outliers removed**
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

**Prophet 7 days isolation forest preprocess ca. 1000 outliers removed**
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

**Prophet 14 days no preprocess**
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

**Prophet 14 days 1.96sigma preprocess ca. 500 outliers removed**
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

**Prophet 14 days isolation forest preprocess ca. 2000 outliers removed**
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

Rolling mean 4 hour window size
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

Rolling mean 8 hour window size
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

Rolling mean 12 hour window size
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

Stationary time series with anomalies present
Run x incidates the measurement. The first number indicates the amount of true alarms. The second number indicates the amount of missed alarms. The third number indicates the amount of false alarms. The structure is as follows;
Run x: True alarms, Missed Alarms, False alarms

No pre-process prophet 14 days of training data
Run 1: 8, 0, 0
Run 2: 9, 0, 0
Run 3: 6, 1, 0
Run 4: 8, 0, 0
Run 5: 8, 0, 0

Prophet 14 days training data 1.96 sigma pre-process ca. 1350 data points removed from the training set
Run 1: 17, 0, 0
Run 2: 18, 0, 0
Run 3: 17, 0, 0
Run 4: 17, 0, 0
Run 5: 17, 0, 0

Isolation forest (0.05 contamination) pre-process prophet 14 days training data ca. 2000 data points removed from the training set.
Run 1: 15, 0, 1
Run 2: 15, 0, 1
Run 3: 15, 0, 1
Run 4: 15, 0, 1
Run 5: 14, 0, 1

No pre-process prophet 7 days training data
Run 1: 8, 2, 0
Run 2: 9, 1, 0
Run 3: 13, 1, 0
Run 4: 8, 1, 0
Run 5: 9, 1, 0

1.96 sigma pre-process prophet 7 days training data ca 1200 data points removed from the training set.
Run 1: 15, 0, 0
Run 2: 15, 0, 0
Run 3: 15, 0, 0
Run 4: 15, 0, 0
Run 5: 16, 0, 0

Isolation forest (0.05 contamination) pre-process prophet 7 days training data ca 1000 outliers removed
Run 1: 12, 0, 0
Run 2: 15, 0, 0
Run 3: 13, 0, 0
Run 4: 13, 0, 0
Run 5: 13, 0, 0

Rolling mean 4 hour window size
Run 1: 14, 1, 0
Run 2: 14, 1, 0
Run 3: 14, 1, 0
Run 4: 14, 1, 0
Run 5: 14, 1, 0

Rolling mean 8 hour window size
Run 1: 14, 1, 0
Run 2: 14, 1, 0
Run 3: 14, 1, 0
Run 4: 14, 1, 0
Run 5: 14, 1, 0

Rolling mean 12 hours training data
Run 1: 18, 0, 0
Run 2: 18, 0, 0
Run 3: 18, 0, 0
Run 4: 18, 0, 0
Run 5: 18, 0, 0

**Seasonal time series with anomaly**

Run x indicates the measurement. The first number indicates the amount of true alarms. The second number indicates the amount of missed alarms. The third number indicates the amount of false alarms. The structure is as follows: Run x: true alarms, missed alarms, false alarms

Prophet 7 days no pre-process
Run 1: 1, 0, 0
Run 2: 1, 0, 0
Run 3: 1, 0, 0
Run 4: 1, 0, 0
Run 5: 1, 0, 0

Prophet 7 days 1.96sigma pre-process ca. 400 outliers removed from the training set
Run 1: 1, 0, 0
Run 2: 1, 0, 1
Run 3: 1, 0, 1
Run 4: 1, 0, 1
Run 5: 1, 0, 1

Prophet 7 days training data isolation forest pre-process ca. 1000 outliers removed from the training set
Run 1: 1, 0, 2
Run 2: 1, 0, 1
Run 3: 1, 0, 1
Run 4: 1, 0, 1
Run 5: 1, 0, 1

Prophet 14 days no pre-process
Run 1: 1, 0, 4
Run 2: 1, 0, 4
Run 3: 1, 0, 4
Run 4: 1, 0, 3
Run 5: 1, 0, 3

Prophet 14 days training data 1.96sigma pre-process ca. 543 outliers removed
Run 1: 1, 0, 0
Run 2: 1, 0, 0
Run 3: 1, 0, 0
Run 4: 1, 0, 1
Run 5: 1, 0, 0

Prophet 14 days training data isolation forest pre-process ca. 2000 outliers removed
Run 1: 1, 0, 4
Run 2: 1, 0, 6
Run 3: 1, 0, 5
Run 4: 1, 0, 4
Run 5: 1, 0, 5

Rolling mean 4 hour window size
Run 1: 0, 1, 6
Run 2: 0, 1, 6
Run 3: 0, 1, 6
Run 4: 0, 1, 6
Run 5: 0, 1, 6

Rolling mean 8 hour window size
Run 1: 0, 1, 2
Run 2: 0, 1, 2
Run 3: 0, 1, 2
Run 4: 0, 1, 2
Run 5: 0, 1, 2

Rolling mean 12 hour window size
Run 1: 0, 1, 0
Run 2: 0, 1, 0
Run 3: 0, 1, 0
Run 4: 0, 1, 0
Run 5: 0, 1, 0

**Seasonal time series without anomaly**
Run x: False alarms raised

Prophet 7 days no pre-process
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

Prophet 7 days 1.96sigma pre-process ca. 400 outliers removed
Run 1: 1
Run 2: 1
Run 3: 0
Run 4: 0
Run 5: 0

Prophet 7 days isolation forest ca. 1000 outliers removed
Run 1: 1
Run 2: 1
Run 3: 2
Run 4: 1
Run 5: 1

Prophet 14 days no pre-process
Run 1: 0
Run 2: 0
Run 3: 0
Run 4: 0
Run 5: 0

Prophet 14 days 1.96sigma pre-process ca. 550 outliers removed
Run 1: 3
Run 2: 4
Run 3: 4
Run 4: 5
Run 5: 3

Prophet 14 days isolation forest ca. 2000 outliers removed
Run 1: 4
Run 2: 7
Run 3: 5
Run 4: 4
Run 5: 3

Rolling mean 4 hour window size
Run 1: 2
Run 2: 2
Run 3: 2
Run 4: 2
Run 5: 2

Rolling mean 8 hour window size
Run 1: 1
Run 2: 1
Run 3: 1
Run 4: 1
Run 5: 1

Rolling mean 12 hour window size
Run 1: 1
Run 2: 1
Run 3: 1
Run 4: 1
Run 5: 1