Help-Center Introduction

Welcome to the Providence Service Help page. This page explains all the important functions and features of the Providence Service.

This page includes the following topics:

- Monitoring Service About
- Monitoring Service Tree
- Administrative Operations
- Monitoring Internal Jobs
- API Documentation
- Postman Links
- Swagger Links

About

Providence Service Summary

The providence service is a custom monitoring solution built specifically to monitor multiple services.
The main purpose of the monitoring service is to provide a single source of information of the platforms current health state. The health state of the platform is calculated bottom-up from the state of the services (B2B-Services and Basic-Services) and their respective components.
The monitoring service contains a structure of the platform: Services are made of actions, which on the other hand are built out of components. Components can be Azure Resources or external services.

Services are green/OK unless an alert on one of their dependencies is triggered, which will then turn their state to WARNING or ERROR.

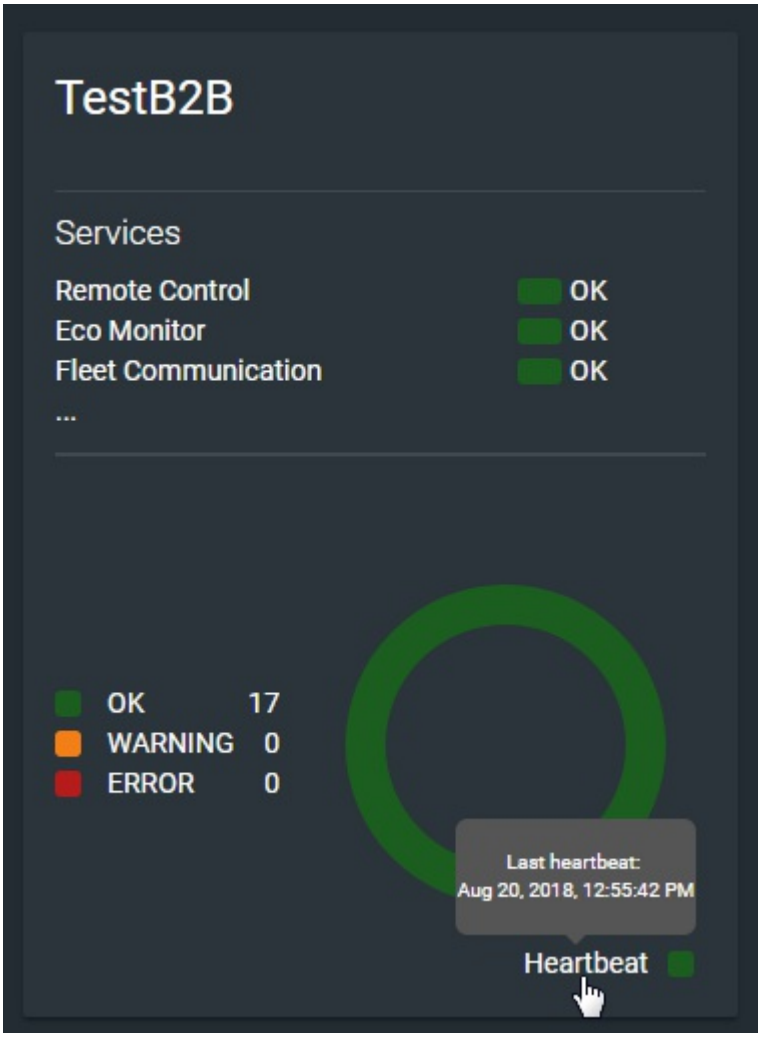
Alerts

Alerts are defined in the platform and when triggered, they change the state of a service's health. The following table shows each state and its implication.

Component's state	Implication	Example
OK	No issues reported/ alerts triggered	-
WARNING	Operations team action required soon to prevent outage of the service	CPU load running continuously at > 80%
ERROR	Action required because a component is failing and auto-recovery was unsuccessful	Watchdog check for microservice failing, Data Factory Pipeline run failing

No Current Data available message

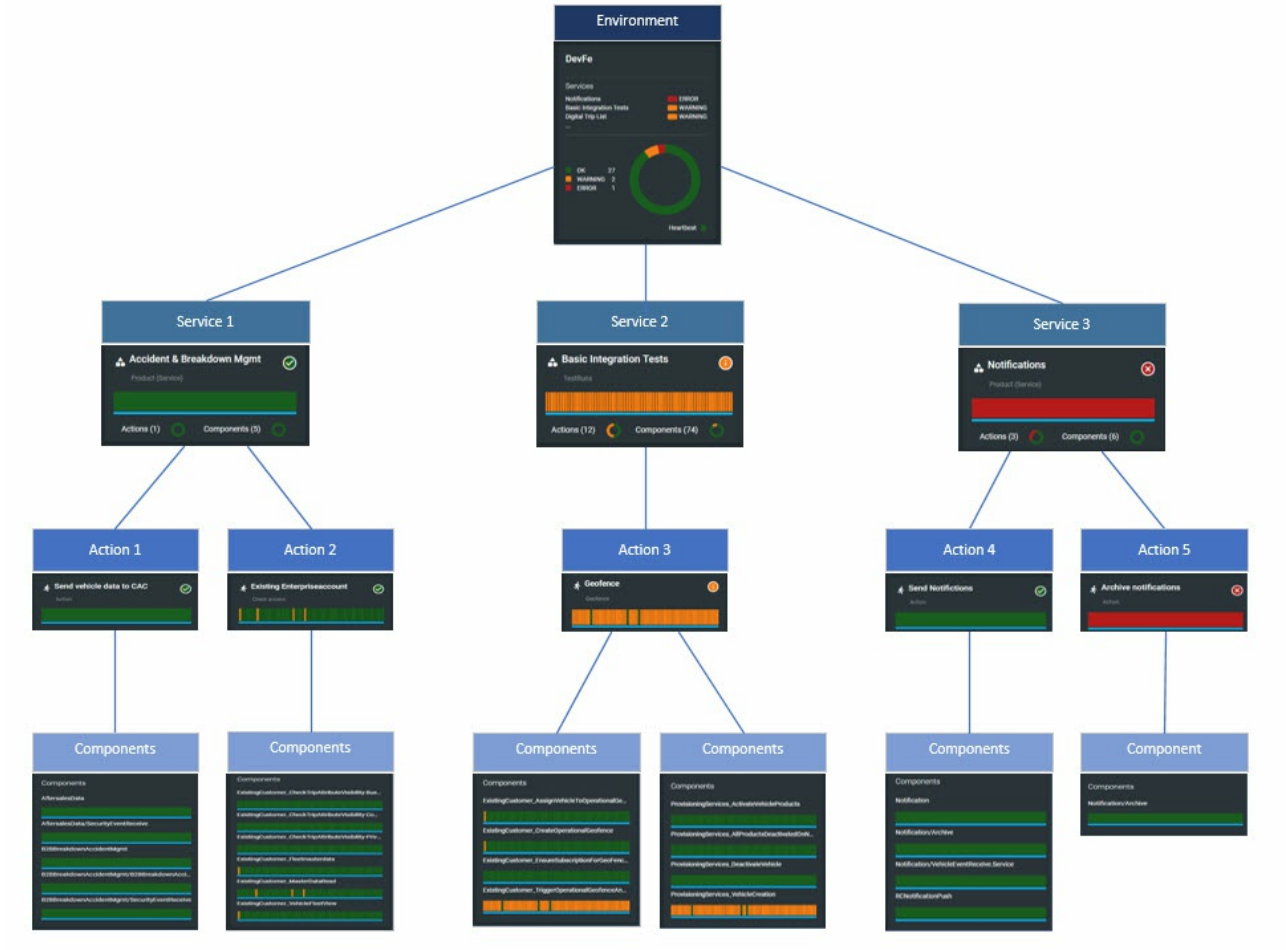
To ensure that the monitoring service is receiving the newest updates of the platform, heartbeat signals are sent periodically from each environment to the service. If no new heartbeats arrive, the connection with the backend might be lost. In this case, data shown in the dashboard might not reflect the current health state of the environments. These are marked with "No current data available".
On the landing page, for each environment the last received heartbeat is shown in the environments tile.


TestB2B
Services
Remote Control
OK
Eco Monitor
OK
Fleet Communication
OK
...
OK 17
WARNING 0
ERROR 0
Last heartbeat:
Aug 20, 2018, 12:55:42 PM
Heartbeat

Providence Service Tree

The Providence service tree is the basis for monitoring the environments.
The service tree is composed of elements which build up a tree with the environment as its root node.

The following diagram shows an example of a service tree:



The root node is always the environment. An environment contains services, which are composed of actions.
Each action has components, which are the leaf nodes of the service tree.

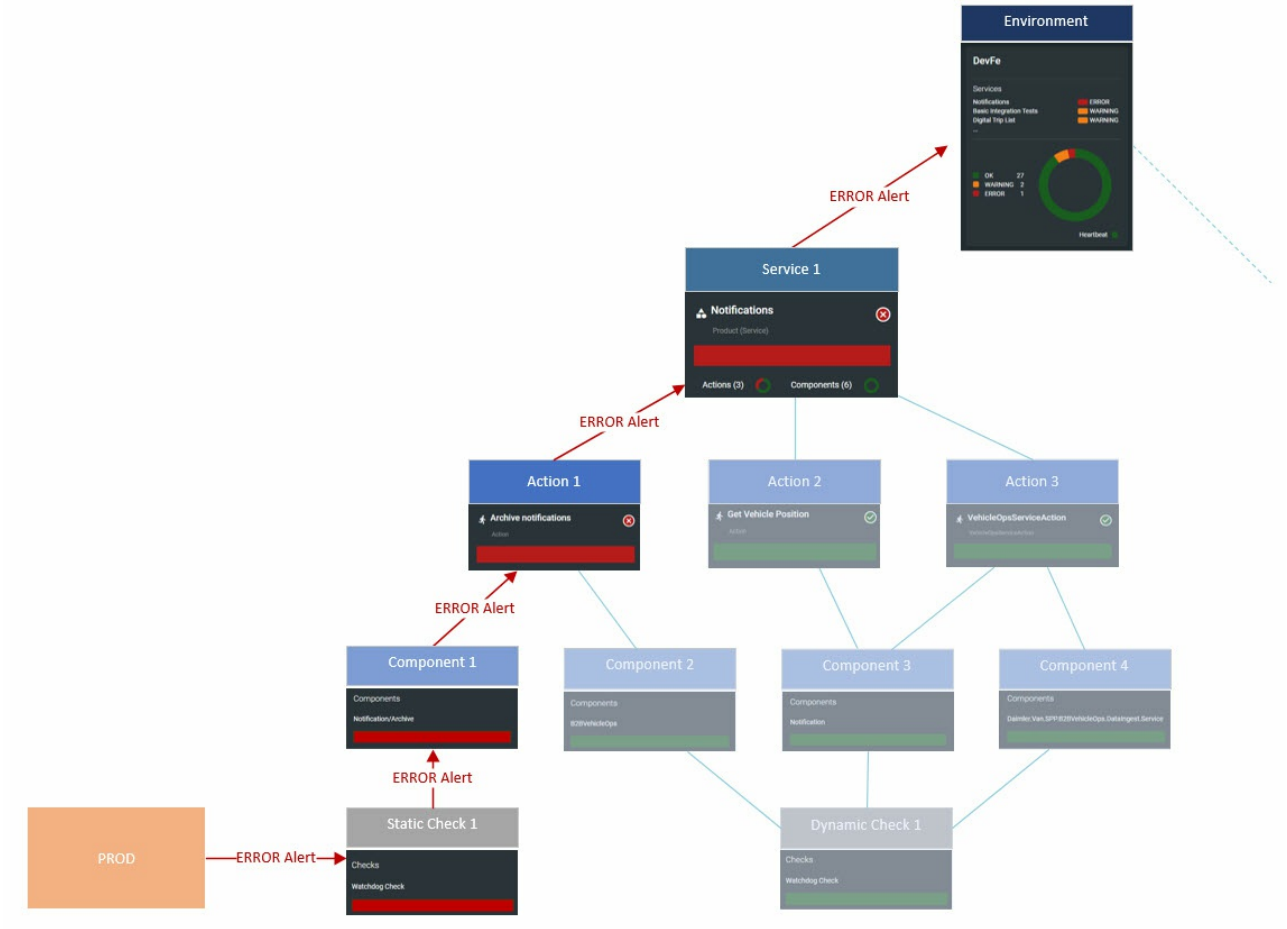
Alerts

Alerts are triggered in each environment. For events to be visible in the Providence dashboard, a check must be created in the Providence service.


Environment
DevFe
Services
Notifications
Basic Integration Tests
Digital Trip List
ERROR
WARNING
WARNING
OK 27
WARNING 2
ERROR 1
Heartbeat
PROD
Alert in the environment
Service 1
Vehicle Operations
Product (Service)
Actions (3)
Components (8)
Action 1
Get Vehicle Position
Action
Action 2
Send Geofence Breaches
Action
Action 3
VehicleOpsServiceAction
Component 1
Components
B2BVehicleOps
Component 2
Components
B2BVehicleOps.DataIngest
Component 3
Components
Notification
Component 4
Components
Daimler.Van.SFP.B2BVehicleOps.DataIngest.Service

State

State in the service tree is propagated bottom-up from the child nodes to their parent node.



Tree and use cases

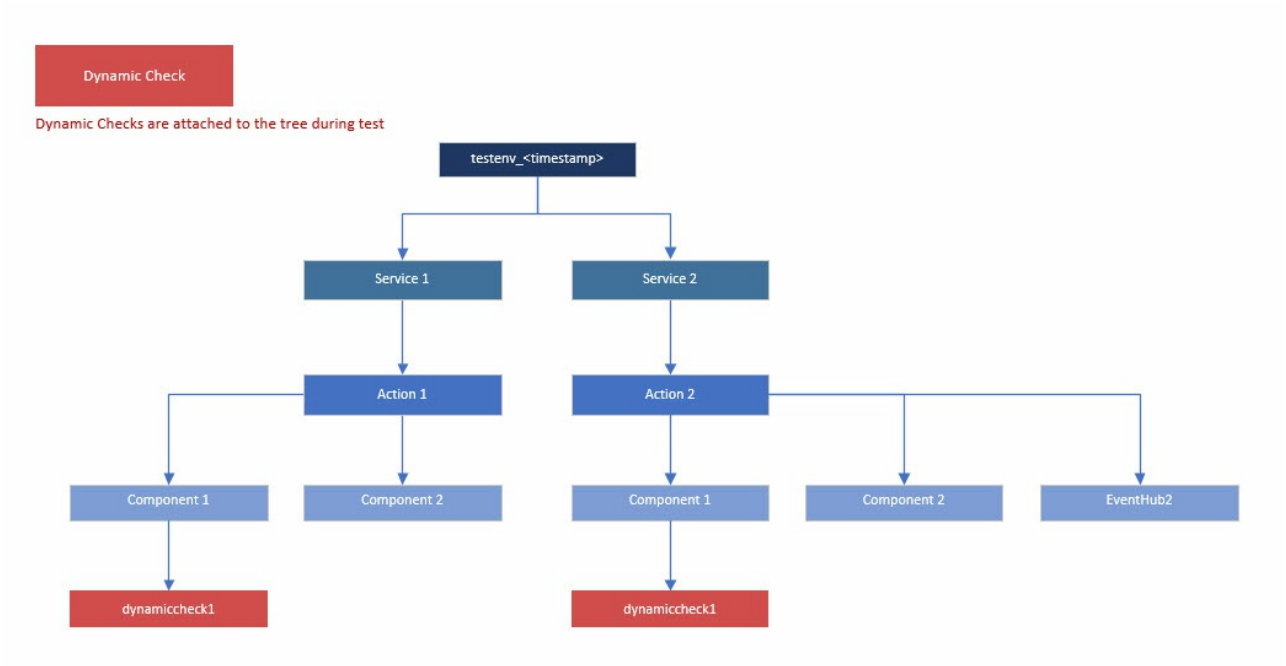
Alerting strategy

You can't fix what you don't know is broken.

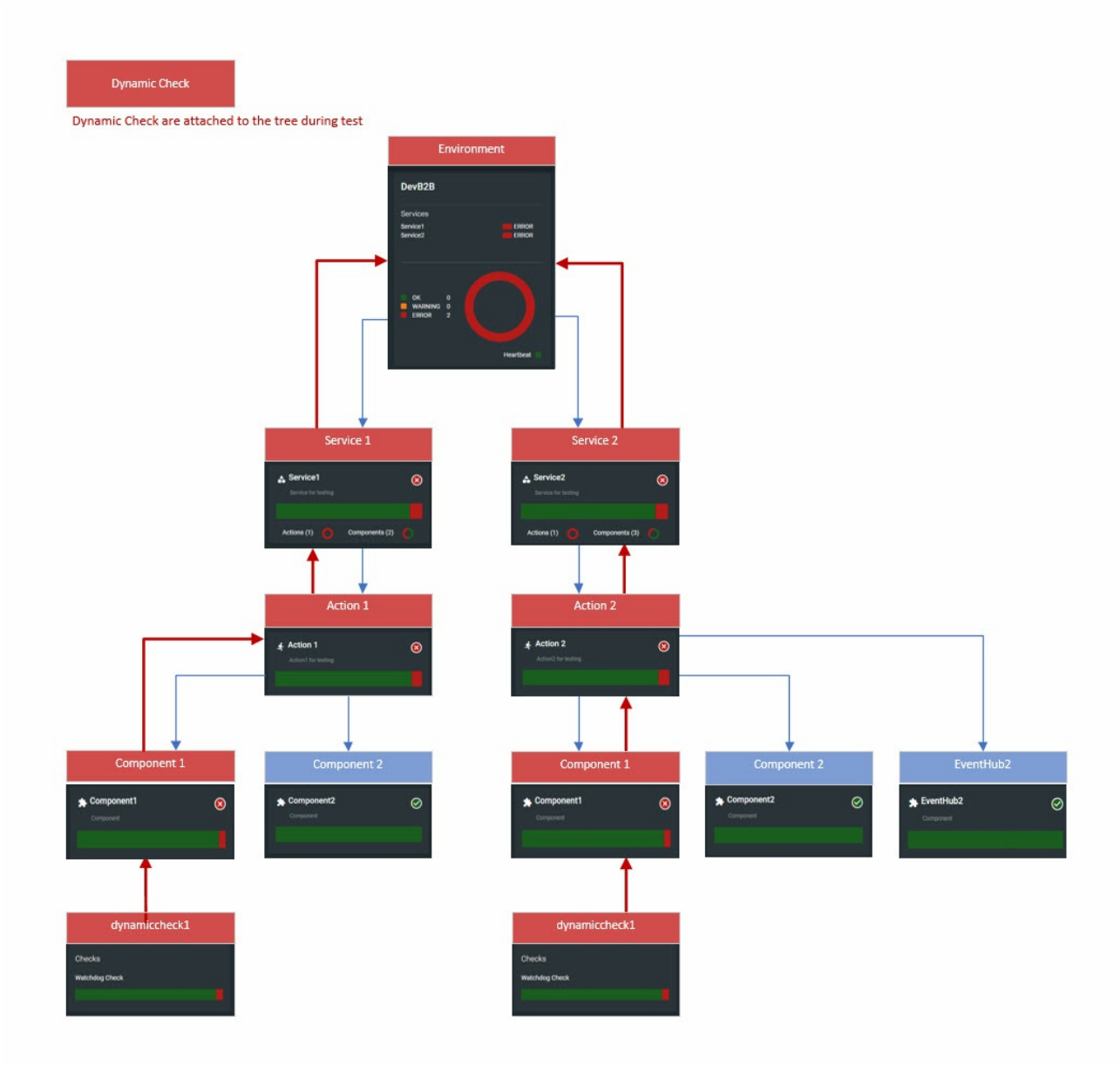
Alerting on what matters is critical, it is underpinned by collecting and measuring the right metrics and logs, as well a monitoring tool that is capable of storing, aggregating, visualizing, analyzing, and initiating an automated response when conditions are met. Improving observability of your services and applications can only be accomplished if you fully understand its composition in order to map/translate that into a detailed monitoring configuration to be applied by the monitoring platform, including the predictable failure states (the symptoms not the cause of the failure) that make sense to alert on.

---

1. Alert for Checks (without AlertName)



For Example Dynamic Check:



Alert

Attribute	Value
CheckId	Watchdog
ElementId	Component1
State	Error

Procedure in background

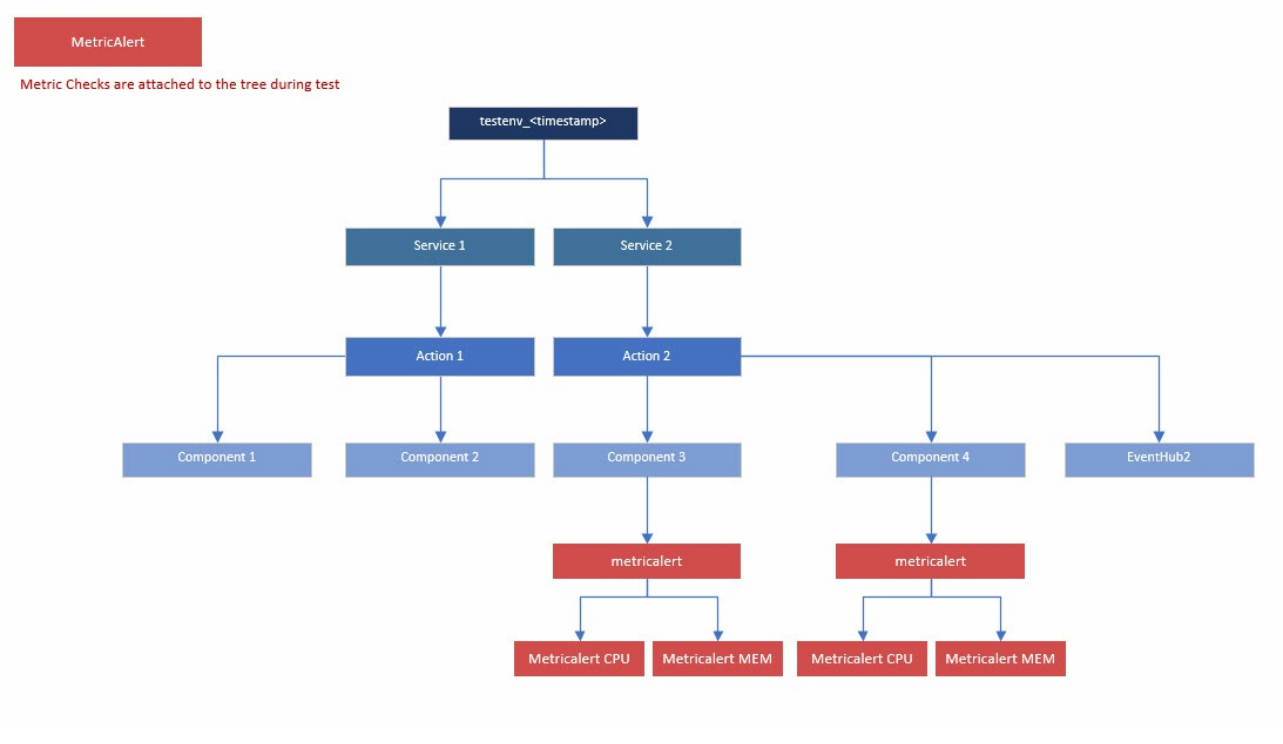
- search for node with ElementId
- attach Check to the node if not existant, else update
- set the state of the Check
- propagate the state transition towards the root of the tree

Statetransitions

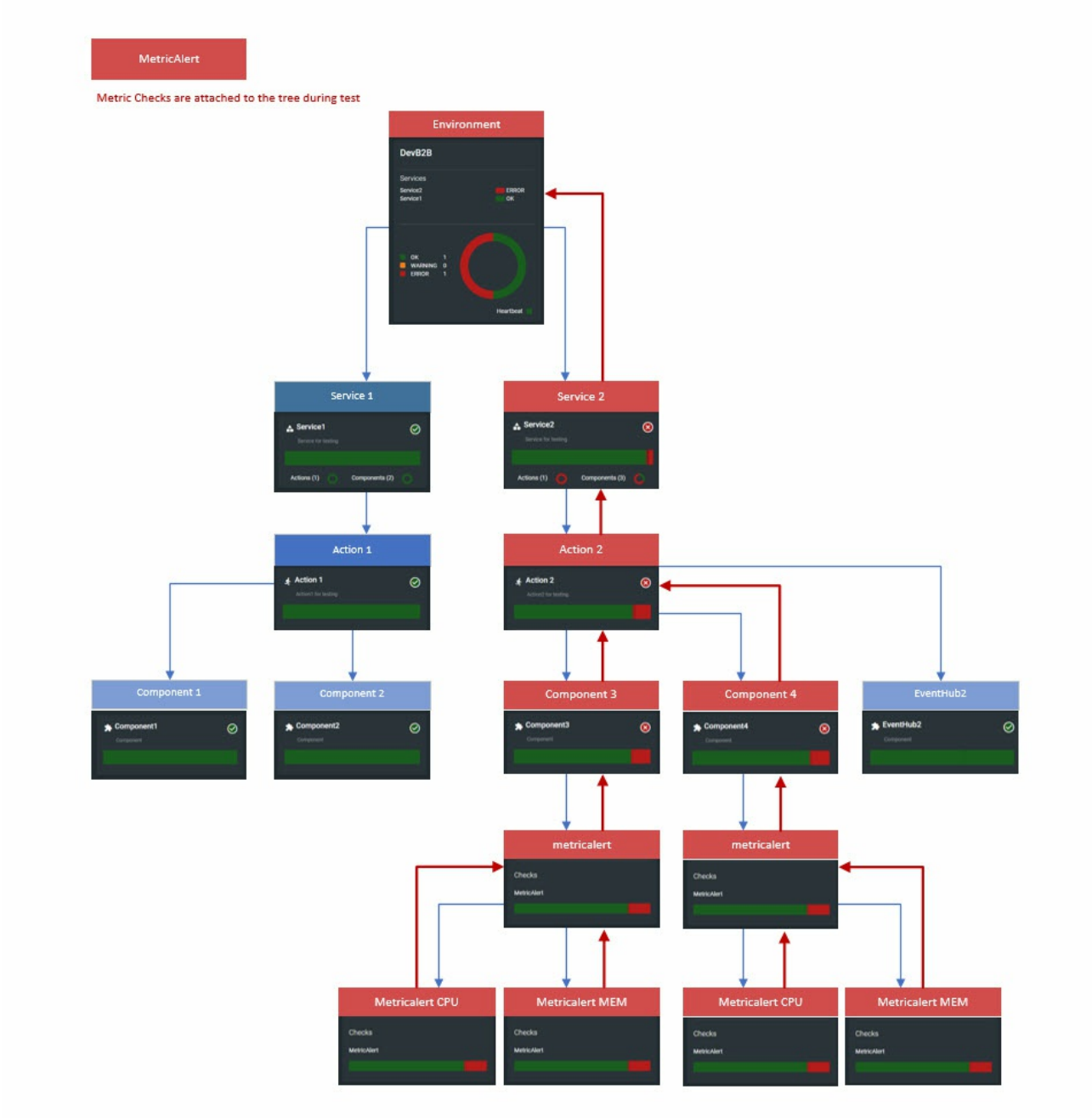
CheckId	ElementId	AlertName	State	Type
Watchdog	Component1	-	Error	Component
-	Action1	-	Error	Action
-	Service1	-	Error	Service
Watchdog	Component1	-	Error	Component
-	Action2	-	Error	Action

-	Service2	-	Error	Service
-	DevB2B	-	Error	Environment

2. Alert for Checks (with AlertName)



For Example Metric Check:



Alert

Attribute	Value 1	Value 2
CheckId	MetricAlert	MetricAlert
ElementId	Component3	Component4
State	Error	Error
AlertName	CPU	MEM

Procedure

- search for node with ElementId
- attach Check to the node if not existant, else update
- set the state of the Check
- propagate the state transition towards the root of the tree
- Attach Check for AlertName

Statetransitions

CheckId	ElementId	AlertName	State	Type
metricalert	Component3	CPU	Error	Component

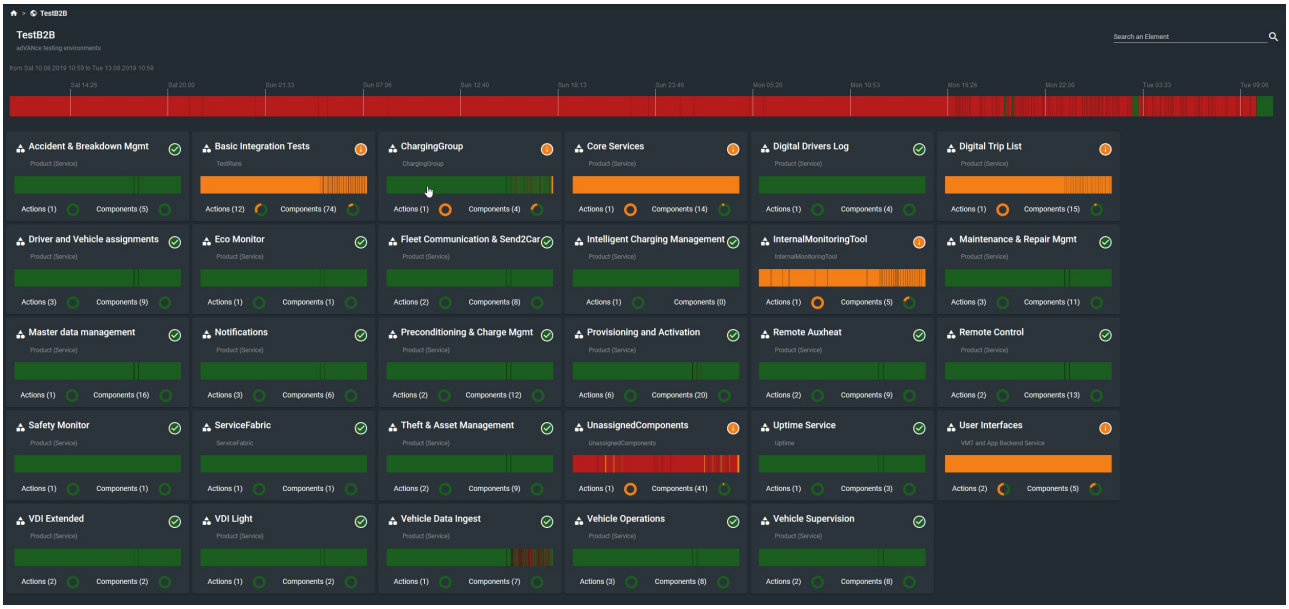
metricalert	Component3	MEM	Error	Component
-	Component3	-	Error	Component
metricalert	Component4	CPU	Error	Component
metricalert	Component4	MEM	Error	Component
-	Component4	-	Error	Component
-	Action2	-	Error	Action
-	Service2	-	Error	Service
-	DevB2B	-	Error	Environment

Graphical User Interface

Dashboard

The dashboard includes information about the current status of the environment.

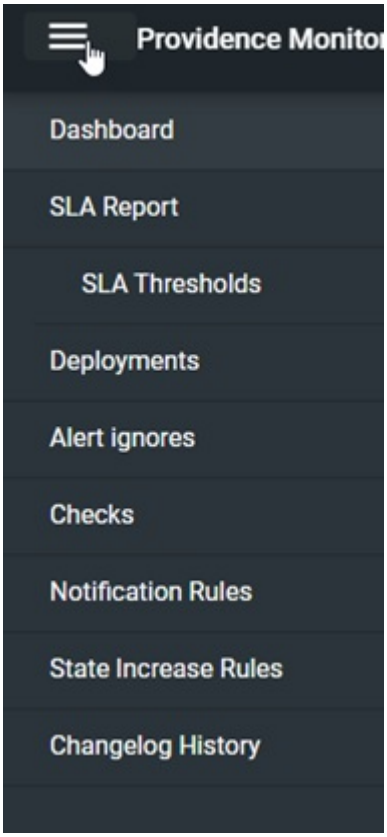
- Timeline of the environment
- Overview of all services monitored: :
    - Service name
    - Status icon (OK, Warning, Error)
    - Service timeline shows the status in color
    - Quantity of actions
    - Quantity of components



Operational Menu

The Operational menu includes the following areas:

- Dashboard
- SLA Report
    - SLA Thresholds
- Deployments
- Alert ignores
- Checks
- Notification Rules
- State Increase Rules
- Changelog History

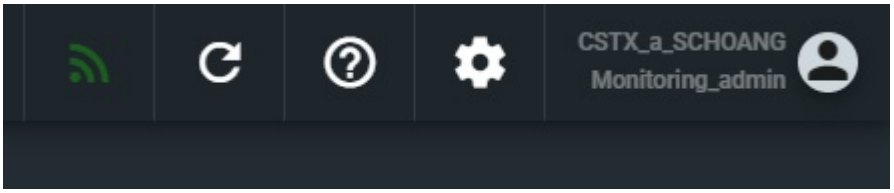


Timeline

The timeline shows the status of the environemt. The status is highlighted in color.
The timeline shows also the deployment slot. The deployment window is marked in blue color, means in that time a deployment of a service is running.

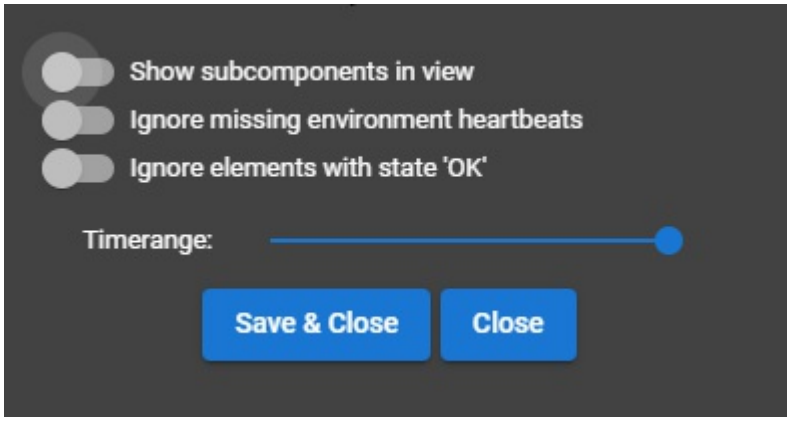
Toolbar

In the upper right corner you can see the Toolbar



The following icons/buttons are displayed:

- Receiving (real time data from monitoring backend)
- Refresh (manual refresh)
- Help (Linkout to the Helpfunction)
- Setting (The settings reference to the dashboard and the timeline)

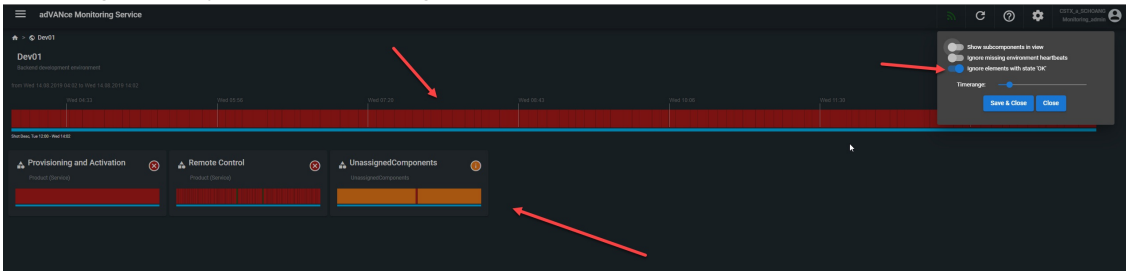


The following settings are available:

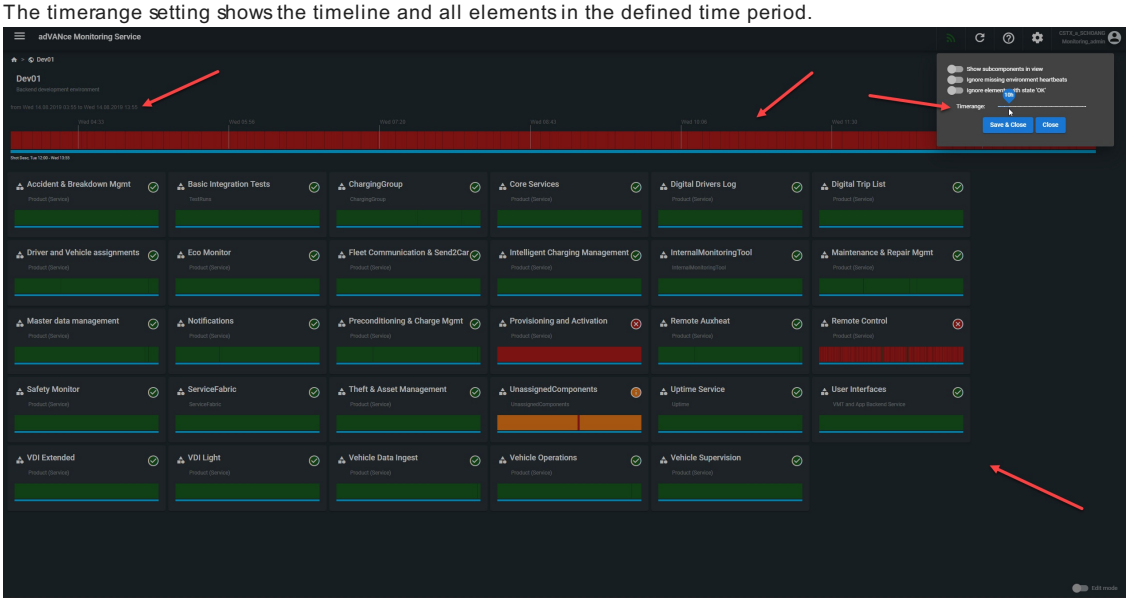
- Show subcomponents in view
  This setting contains the quantity and graphical display of all actions and components for the respective service.

- Ignore missing environment heartbeats
  This setting ignore missing heartbeats for all visible environments which means that no "No Data available" message will be shown for those environments.

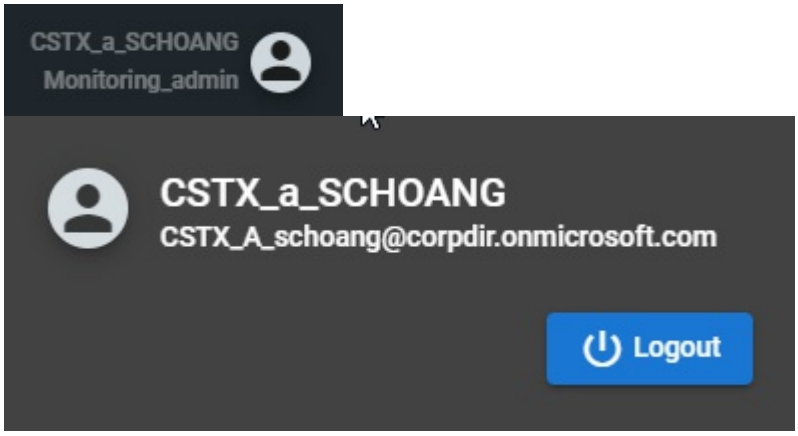
- Ignore elements with state "OK"
  This setting shows only elements with warnings or error. All elements with the status "OK" are not listed in this overview.



- Timerange setting
  The timerange setting shows the timeline and all elements in the defined time period.



- Login / Logout User (User Profile)
  Click on User Profile and "Logout" button if you want to logout.

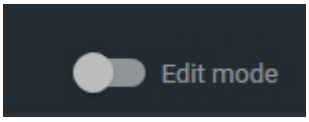


Search an Element

The search function can be used to search for elements in the environment to step directly to this element.

Administration

In the lower right corner you can see the toogle 'Edit mode', it is only visible for the administrator.
He can create, modify and delete environments, services, actions and components.



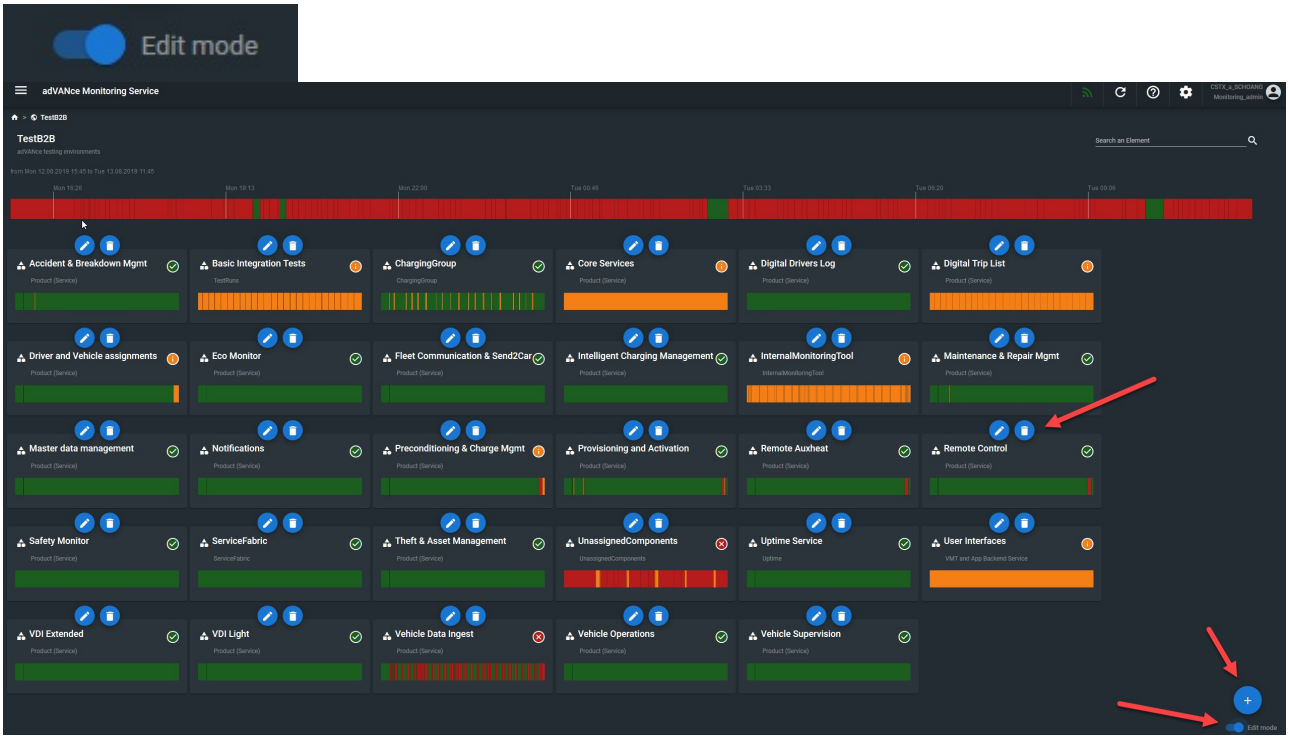
***

Administrative Operations

Administration

The Providence Service offers several ways to edit the dashboard. These are referred to as "administrative operations" and can be activated using the "Edit mode" switch. To see and activate the switch, a user must have the "Admin" role.
The admin can then create, modify or delete environments, services, actions and components directly on the dashboard.

When the "Edit mode" is active, the following icons are visible:



Click the "Edit" icon to update any element in the detail dialog.

Update Environment

Name *
Dev01

Description
UFT_Description1_Modify

Element Id

Subscription Id

Demo environment?

Save Cancel

Update action

Name *
UnassignedComponentsAction

Description
UnassignedComponentsAction

Element Id
UnassignedComponentsActiontestb2b

Save Cancel

Update component

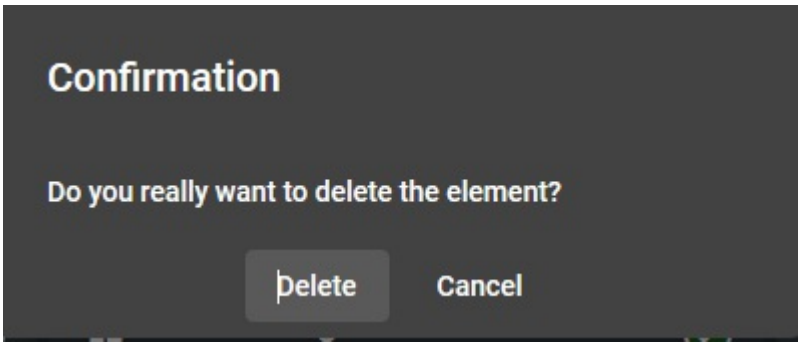
Name *
ChargingGroupMgmt/Service

Description
ChargingGroupMgmt/Service

Element Id
fabric:/Daimler.Van.SPP.ChargingGroupMgmt/Service

Component Type *
Component

Click on the "Delete" icon to delete the selected element.



Click on the "Add" icon to add a new element.

Create new environment

Name *

Description

Element Id *

Subscription Id *

Demo environment?

Save Cancel

Create new service

Name *

Description

Element Id *

Save Cancel

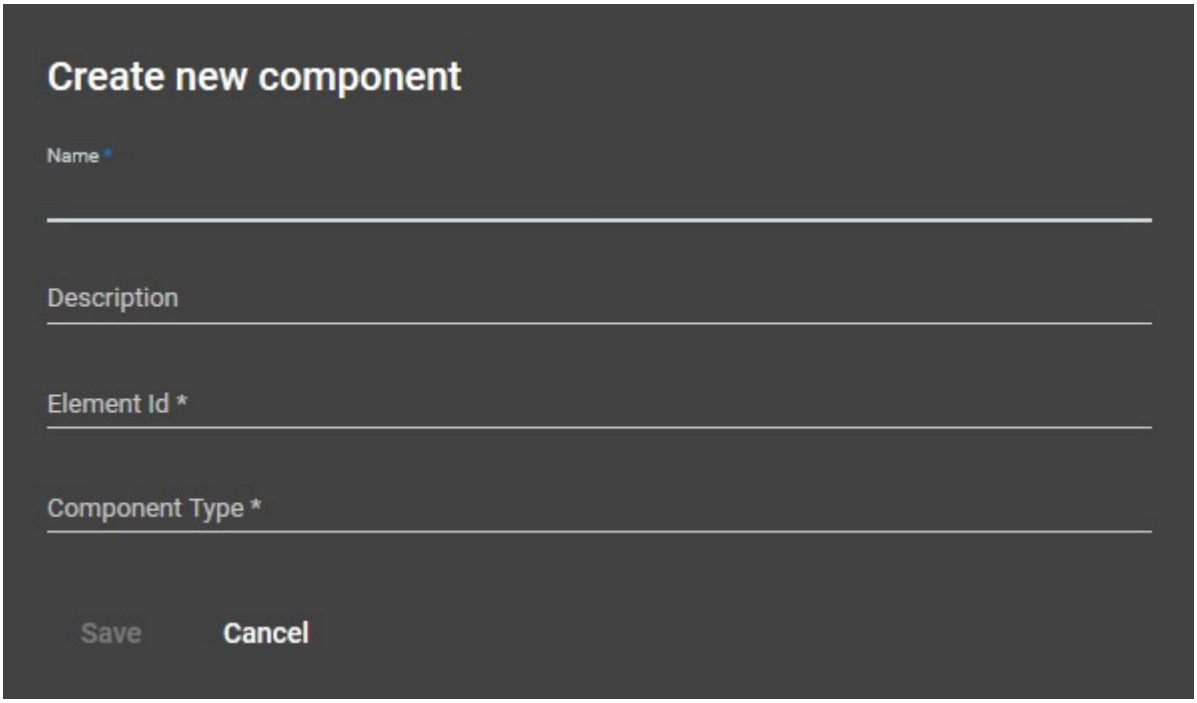
Create new action

Name *

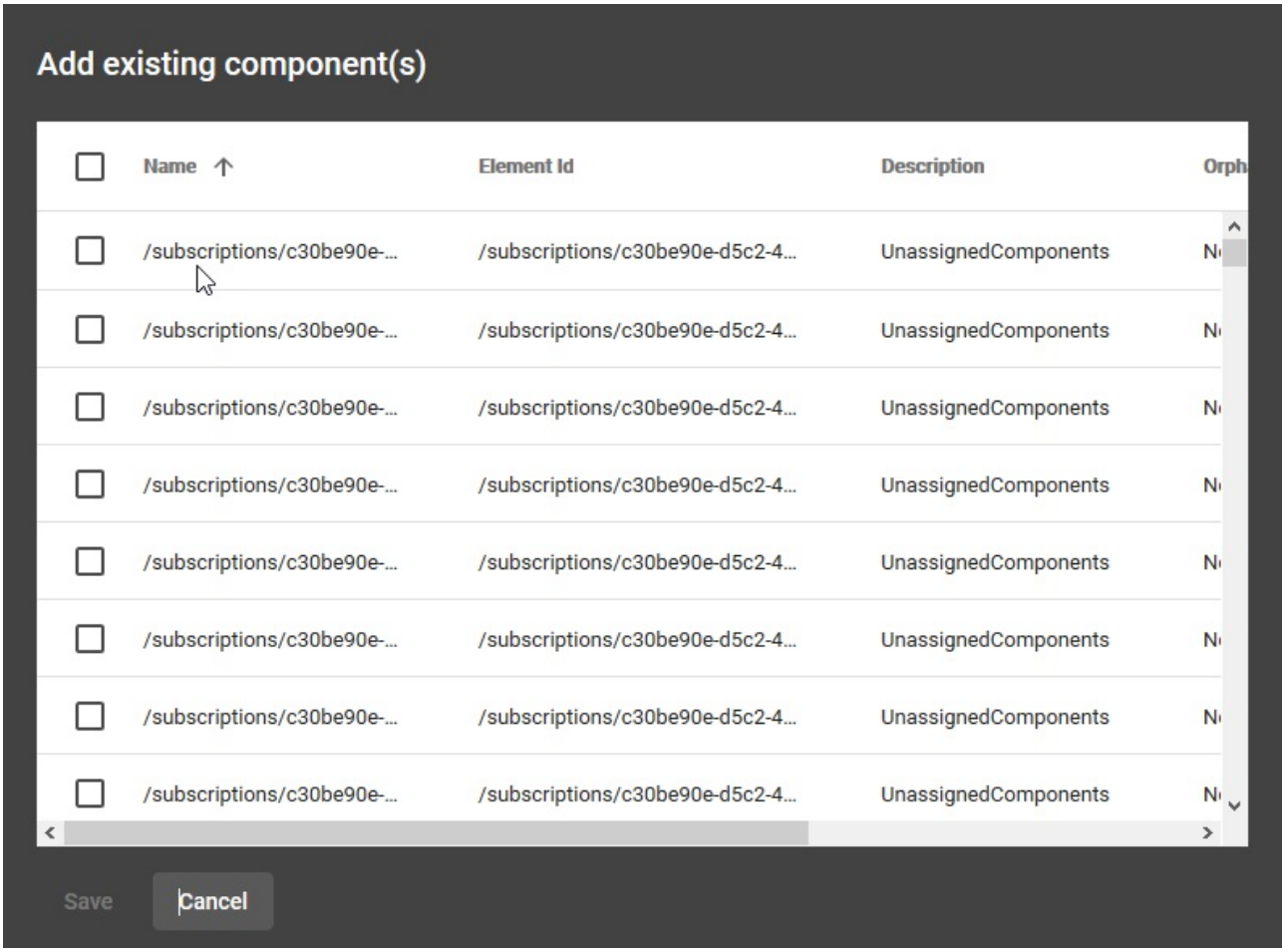
Description

Element Id *

Save Cancel



Click on "Add existing component(s)" icon to add an existing component(s).
This icon is only visible in the component level.



Operational Menu

Click on "menu" icon to make the column "operational menu" visible or to hide it again.

The operational menu includes the following pages:

- Dashboard Overview
- SLA Report
    - SLA Thresholds
- Deployment
- Alert ignores
- Checks
- Notification Rules
- State Increase Rules
- Changelog History

Dashboard Overview The dashboard contains information about the current status of the environment.

- Timeline of the environment (The timerange is adjusted in the toolbar)
- List of all services to be monitored:
    - Service name
    - Status icon (OK, Warning, Error)
    - Service timeline in color representation of the status
    - Quantity of actions with graphic display
    - Quantity of components with graphic display

SLA Thresholds

The system monitoring works by "observing" the individual instances of an environment and "tracking" the various metrics within that instance. Each metric has two defined thresholds, one for minor (warnings) and one for major or critical problems (errors).

If a metric is above the error threshold it will be seen as error state.
If a metric is above the warning threshold but under the error threshold it will be seen as warning state

The thresholds are administered by the Administrator and can be visualized in the SLA report.

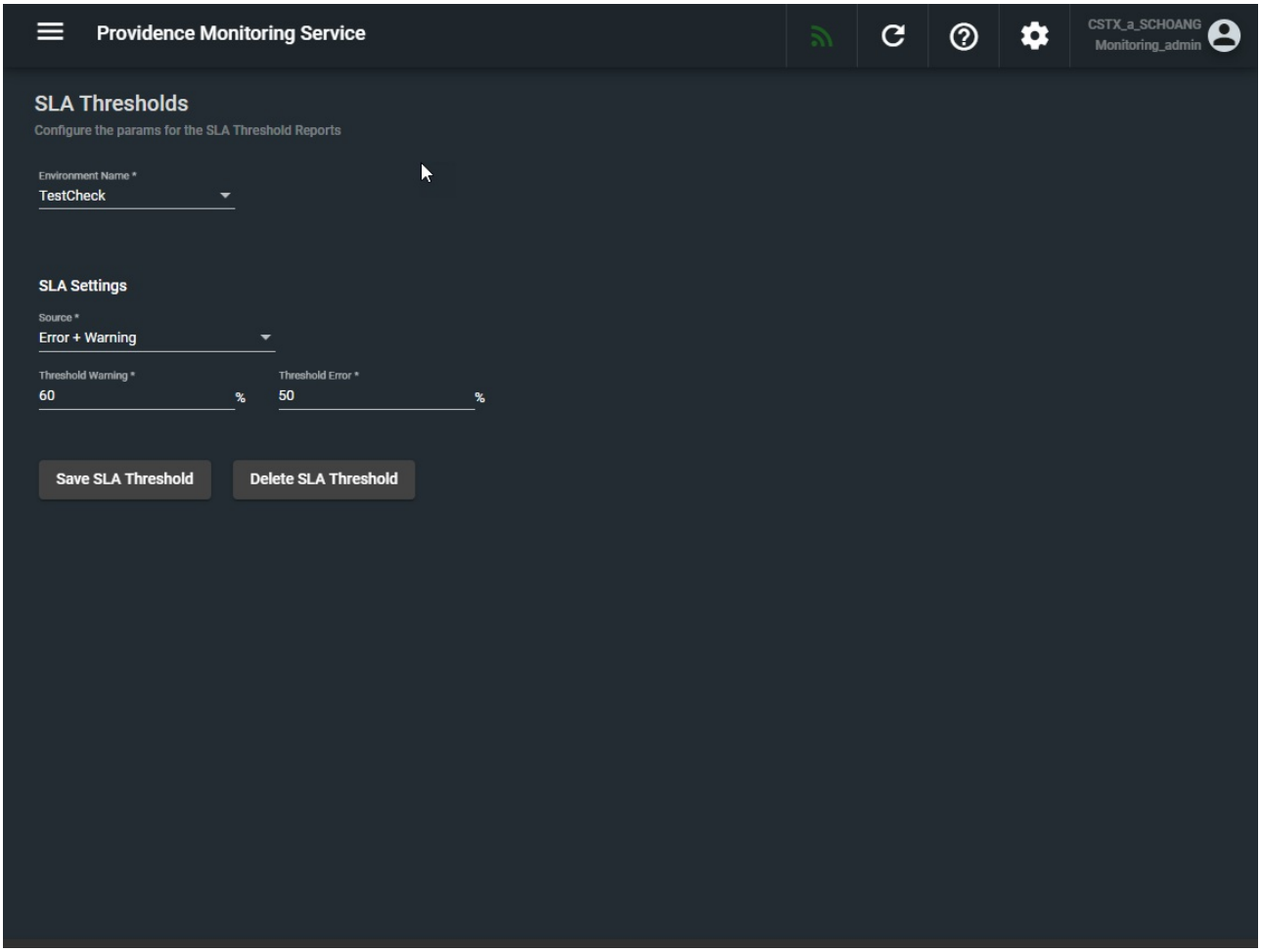
Hint: Define thresholds which reflect the expectations of your customers.

Warning: The experience of the enduser can be influenced badly without running in critical issues. With a warning you can indicate to fix things before your system is running in crictical circumstances.

Error: If you are in the critical state then you need to fix issues immediately to come back to normal business again.

With administrating the thresholds you define following options:

- **Environment Name:**
  Choose the environment to administrate.
- **Source:**
  Choose between Option "Error" and "Error + Warning". This specifies whether alerts with the status "Warning" are to be taken into account when calculating the service downtimes.
- **Threshold Warning:**
  Define the threshold for the warning in %, this values needs to be higher then the error threshold.
- **Threshold Error:**
  Define the threshold for the state error in %



SLA Report Using the SLA Reports you can monitor the compliance of the Service Level Agreement, and you can recognize incidents early.

Common example of a availability and outages:
A good overview gives the following table showing the cumulative availability of a service per year in percent.
The calculation is made on following basis: 365 days * 24 hours/day = 8.760 hours / year

Availability Percentage	Availability (Year)	Downtime (Year)
100 %	≙ 8.760 h	≙ 0 h
99 %	≙ 8.672 h	≙ 88 h
90 %	≙ 7.884 h	≙ 876 h
80 %	≙ 7.008 h	≙ 1.752 h
50 %	≙ 4.380 h	≙ 4.380 h
10 %	≙ 876 h	≙ 7.884 h

Calculation basis
Our calculations for the SLA-report are not calculated on the complete data, it will be calculated along a defined time range.
Also at first you need to define the time duration in which you want to have the SLA report. This time period is named "SLA-Period".

Downtime ≙ sum(Error-Duration) + sum(Warning-Duration) in SLA-period

Uptime ≙ 1- (Downtime/SLA-period) [%]

Sample:

SLA-period: 01.01.2020 - 02.01.2020 -> SLA-period: 24 [h]

Error: 01.01.2020 05:00 - 01.01.2020 6:00 -> Error-Duration: 1 [h]

Warning: 01.01.2020 07:00 - 01.01.2020 8:00 -> Warning-Duration: 1 [h]

Downtime = (1 + 1) = 2 [h]

Uptime = 1 - (2 /24) = 0,91 [%]

SLAReport Overview

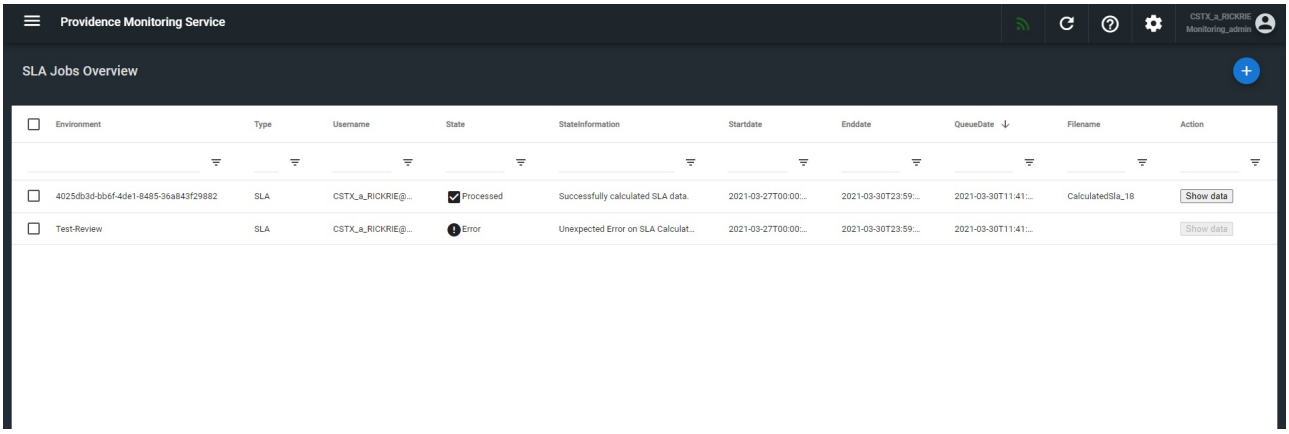
In order to retrieve the SLA calculation for a specifiy environment you have to specify:

- The environment name of your environment
- The start and end date of the calculation

Note: If the start and end date of the SLA period is not given we take as default the last 3 days for the calculation.

The calculations are based on 24 hours / day.

After starting a SLA calculation the queued job is shown in the SLA Jobs Overview

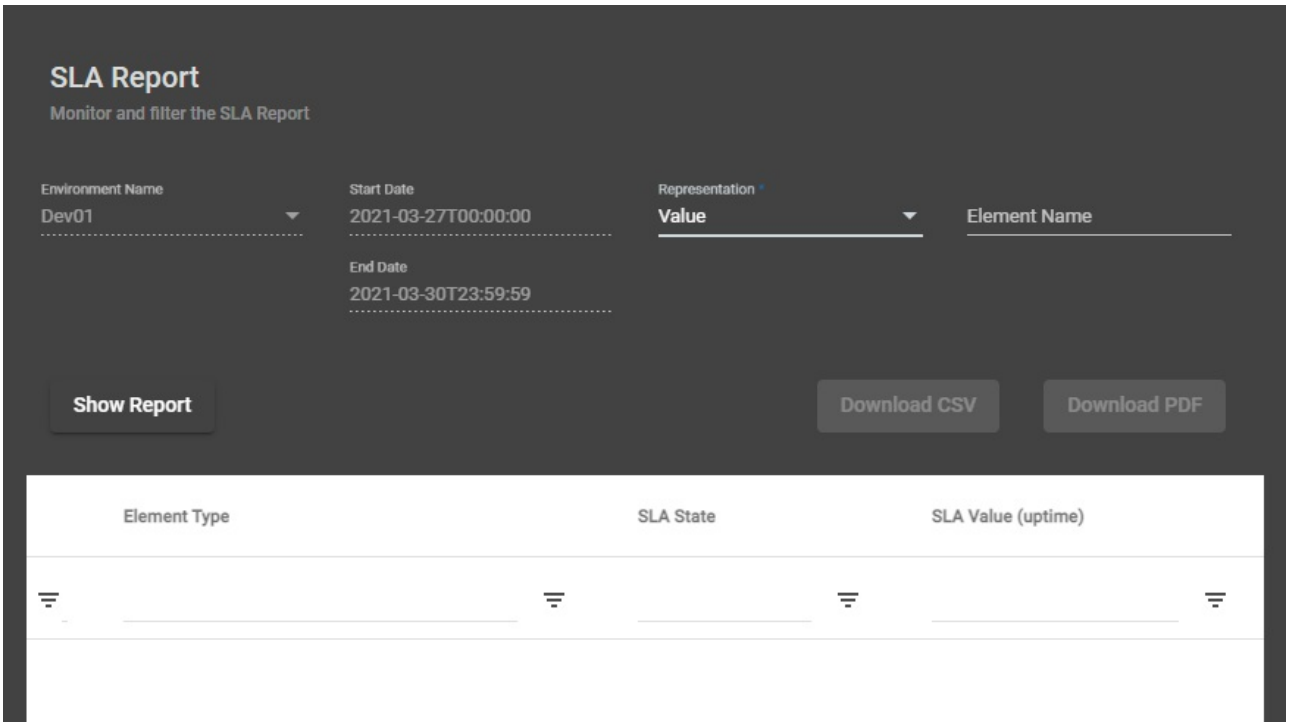


A queued job can have one of the following states:

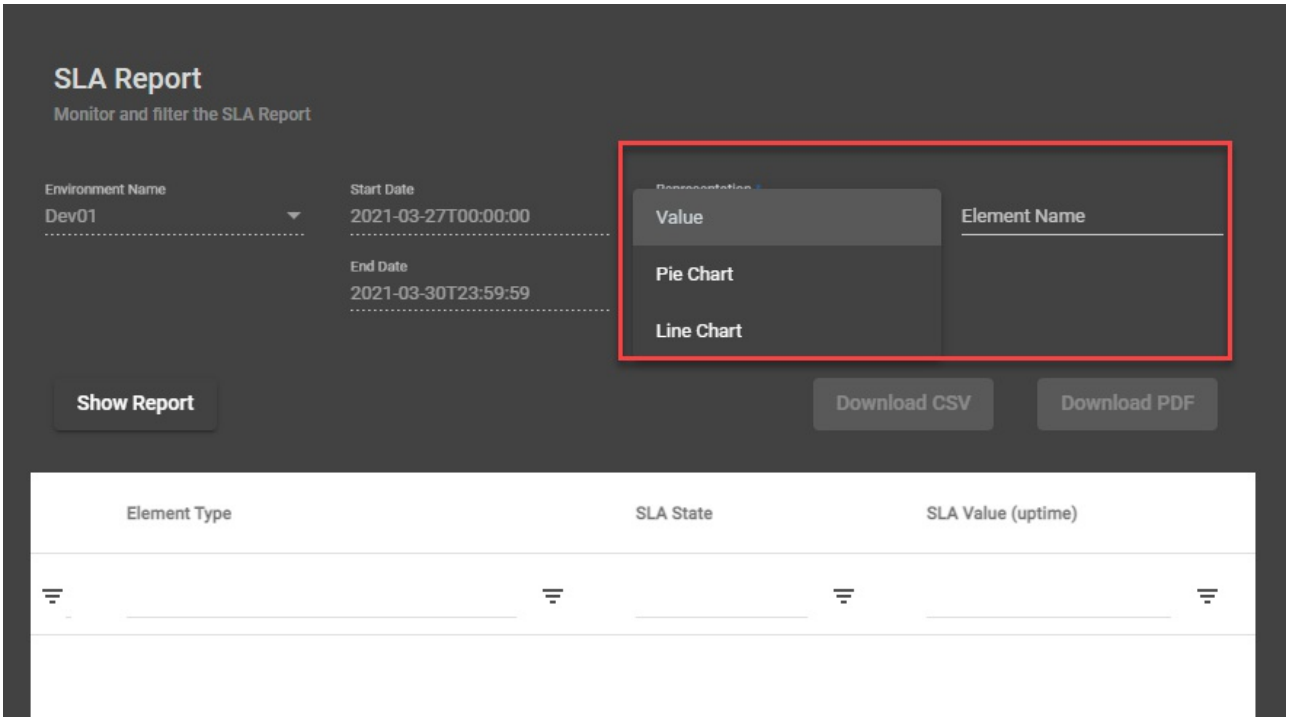
- Queued
- Running
- Processed
- Error

SLA Report Evaluation

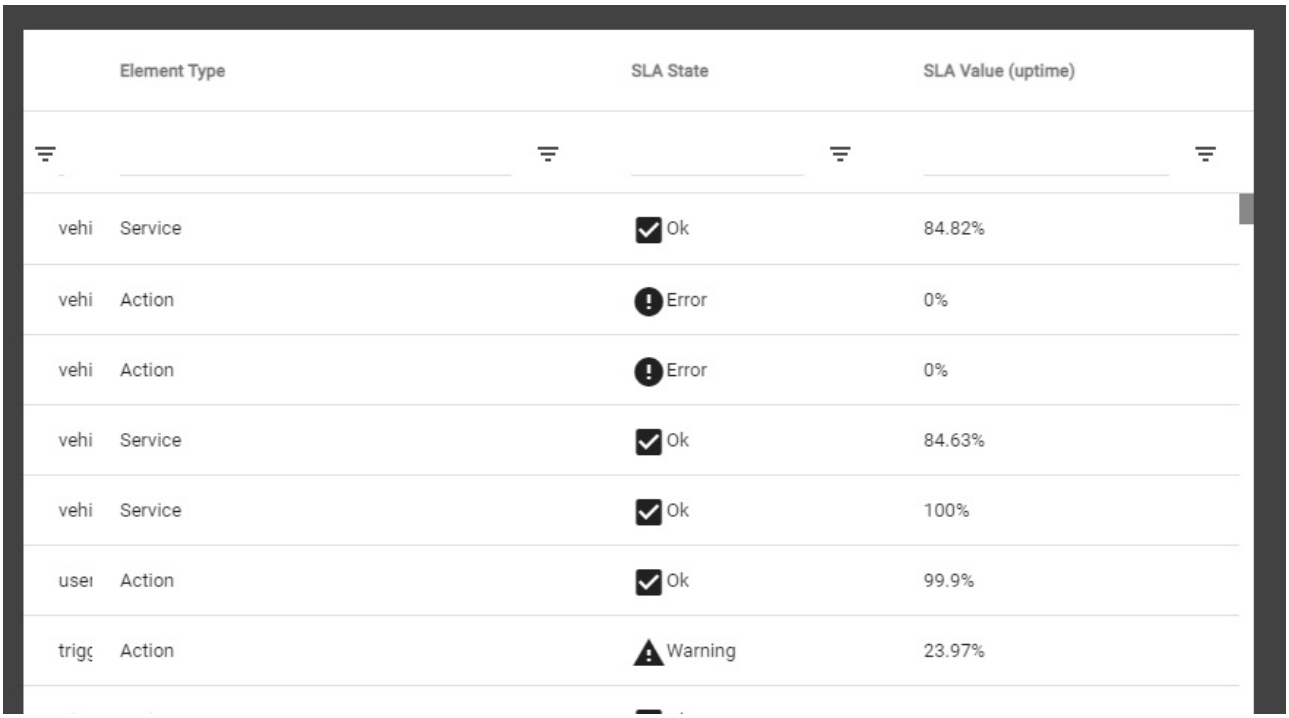
After the SLA is finished and the state was updated to "Processed" you can click on the "Show data" button to open the SLA Overview.



Here you can select the representation type and optionally a single element you want to retriev the SLA calculation for.



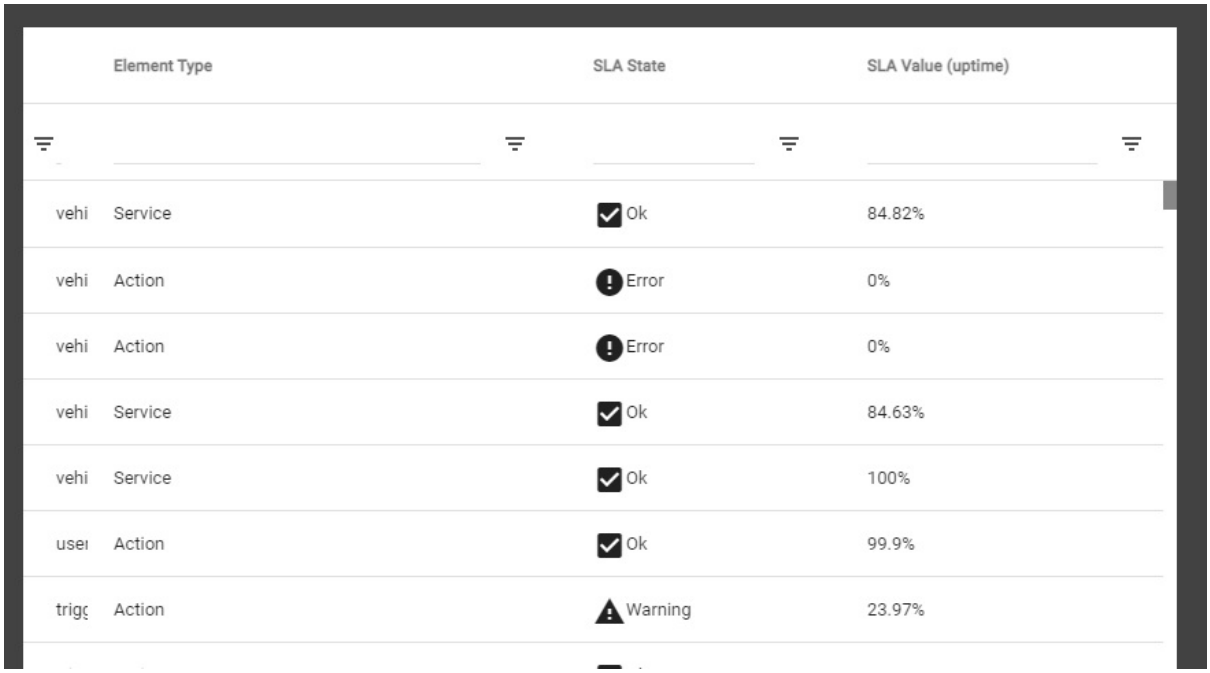
After selecting those properties you can click on "Show Report" to show the calculated SLA values



The "SLA State" delivers the state of the corresponding element and can contain "Error", "Warning" and "OK".
The corresponding value is shown as percent or as a chart (depends on representation format).
If no SLA data is available this is shown in the field "ComponentType" as "No SLA Data found".

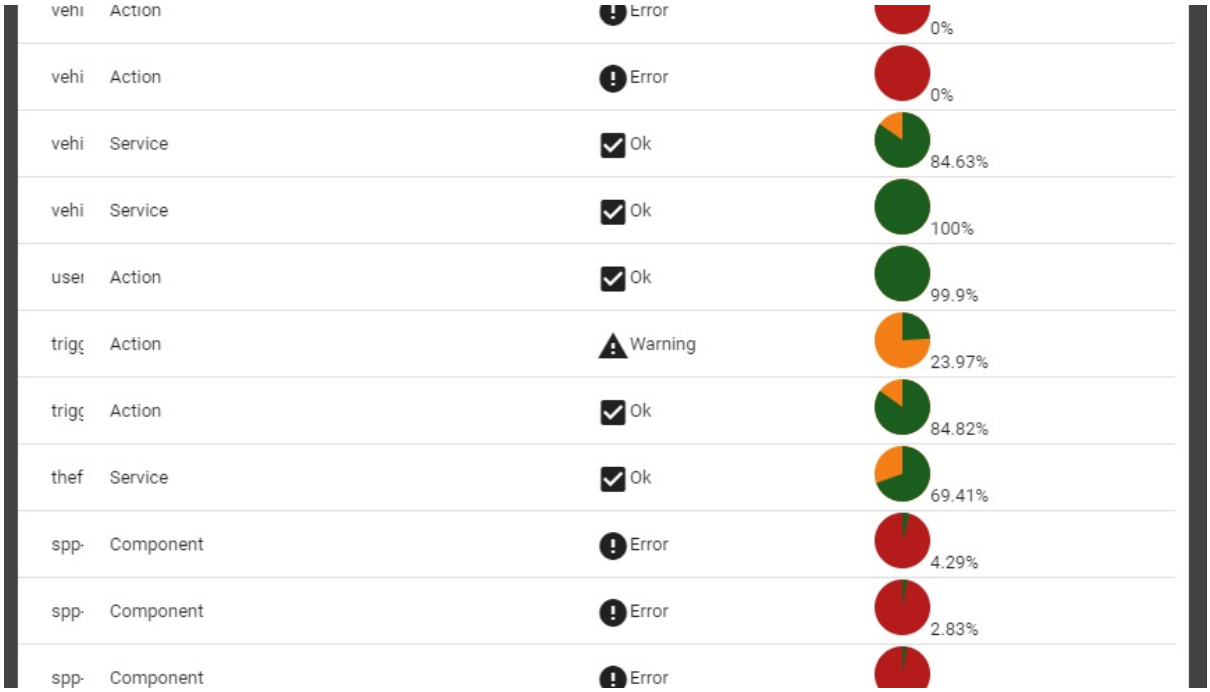
SLA Representation formats

- **"Value" representation format**
  The SLA value is always given in percent and correlates to the SLA uptime.



	Element Type	SLA State	SLA Value (uptime)
vehi	Service	Ok	84.82%
vehi	Action	Error	0%
vehi	Action	Error	0%
vehi	Service	Ok	84.63%
vehi	Service	Ok	100%
user	Action	Ok	99.9%
trigg	Action	Warning	23.97%

- **"Pie Chart" representation format**
  The SLA Pie correlates also always to the SLA Uptime.

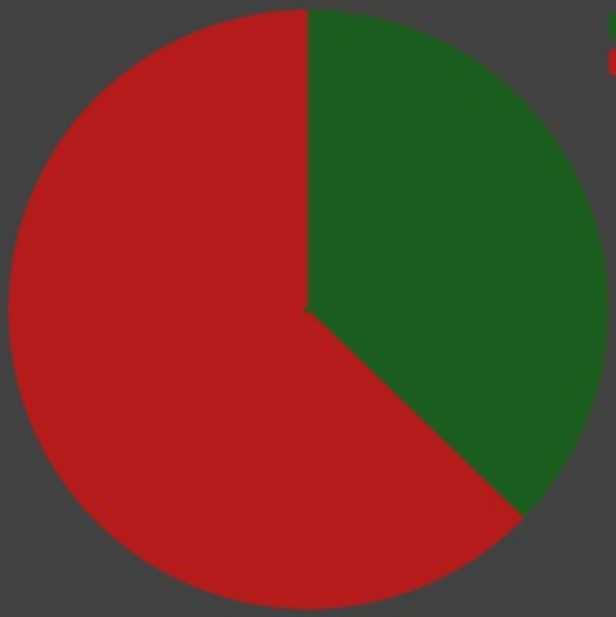


  - **Green part** = Element was **..% available**
  - **Yellow part** = Element was with a **..% Warning**
  - **Red part** = Element was with a **..% Error**

Mouse double click - on a corresponding pie shows a detail information of the data.

SLA report

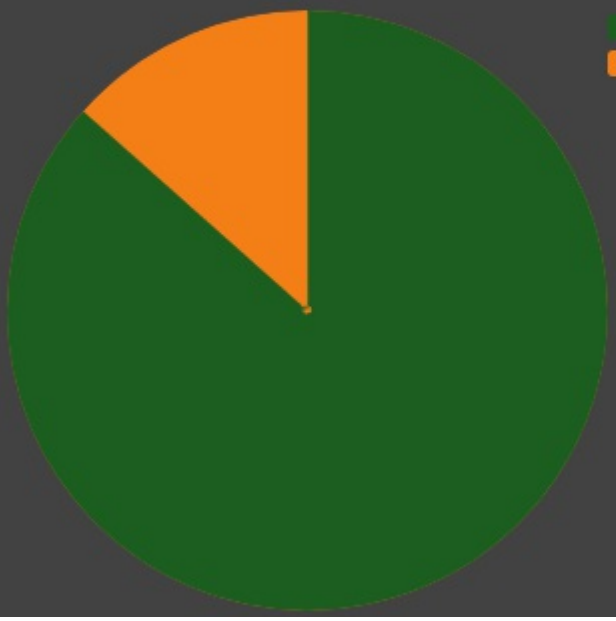
notificationsServiceDev01

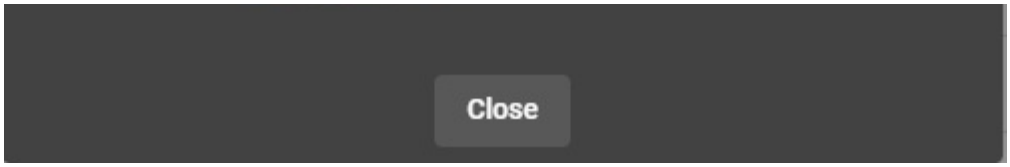


Close

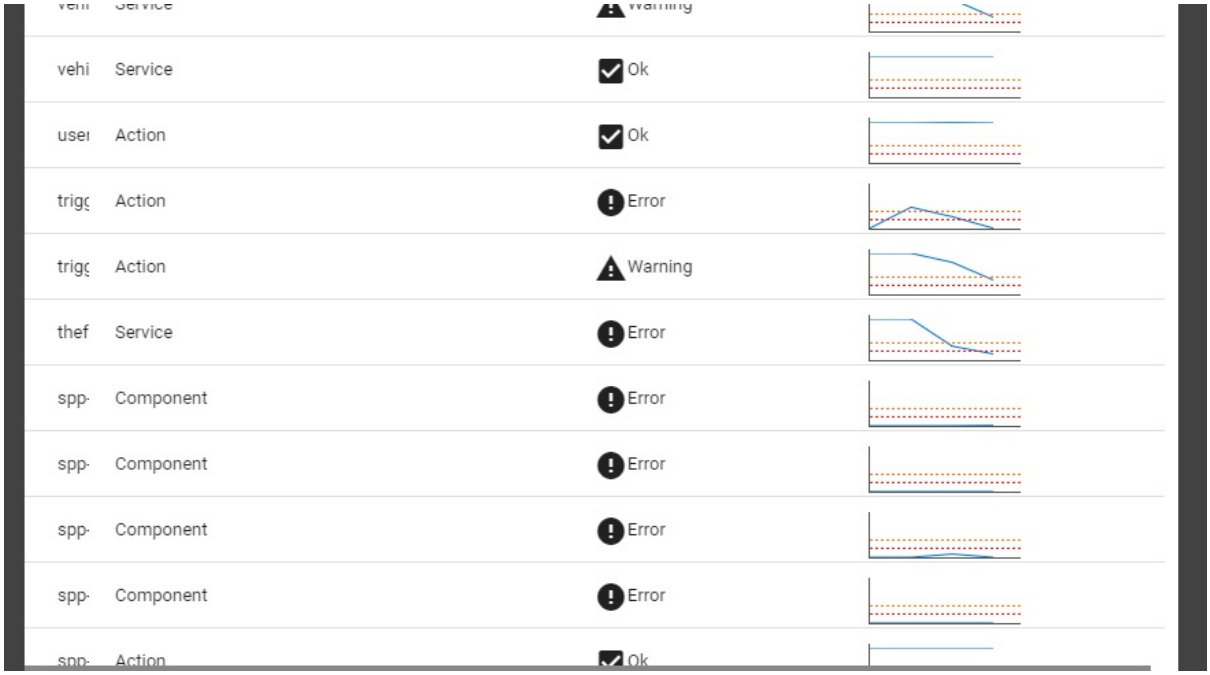
SLA report

/subscriptions/4025db3d-bb6f-4de1-8485-36a843f29882/resourceGroups/spp-monitoring-healthstate-Dev01/providers/Microsoft.Logic/workflows/sppsendtoeventhubhourlyDev01





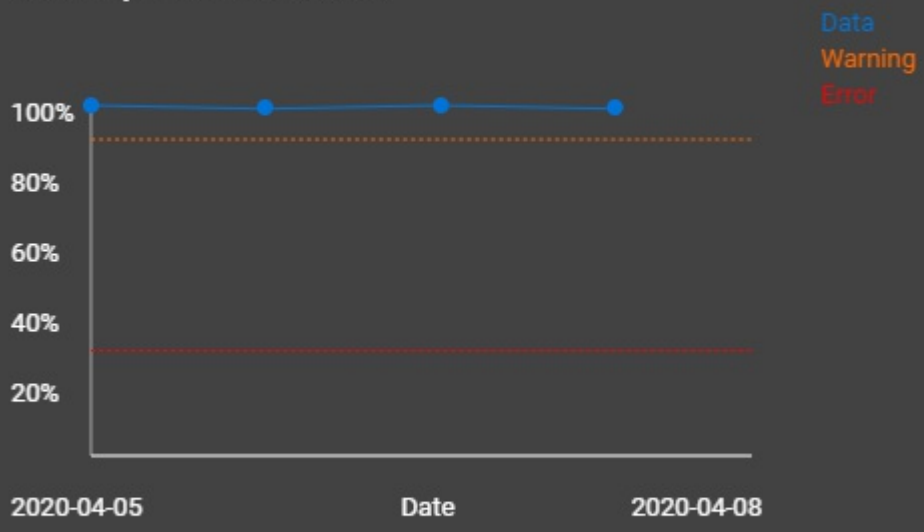
- **"Line Chart" representation format**
  The SLA line chart correlates also on the SLA uptime.



Mouse double click - on a corresponding line chart shows a detail information of the data.

SLA report

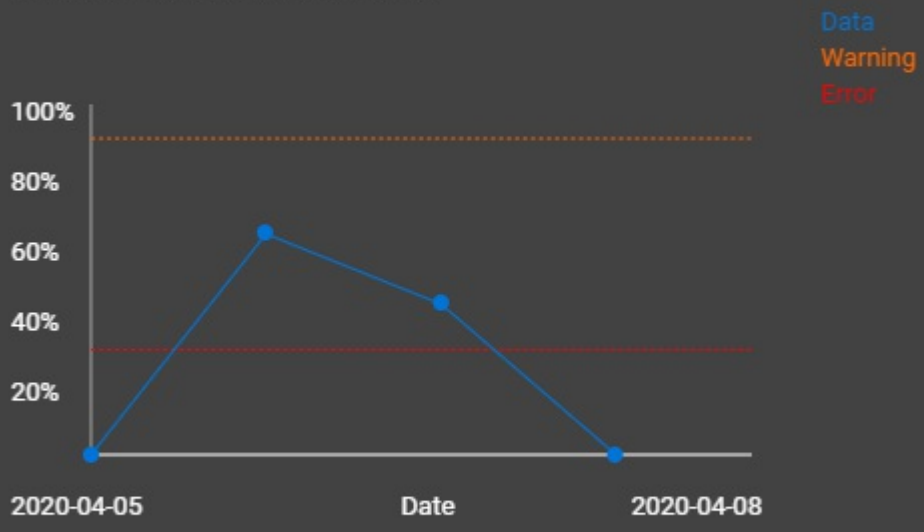
vehicleSupervisionServiceDev01



Close

SLA report

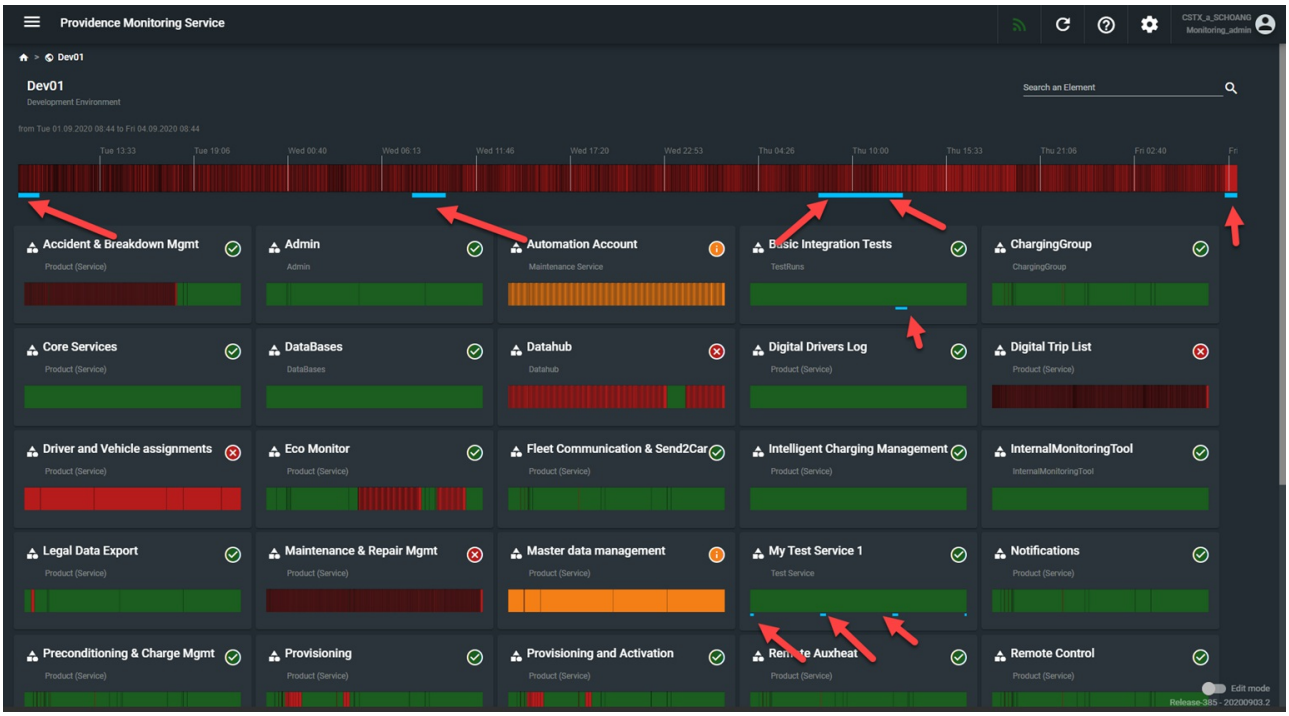
sendEmailNotificationsActionDev01



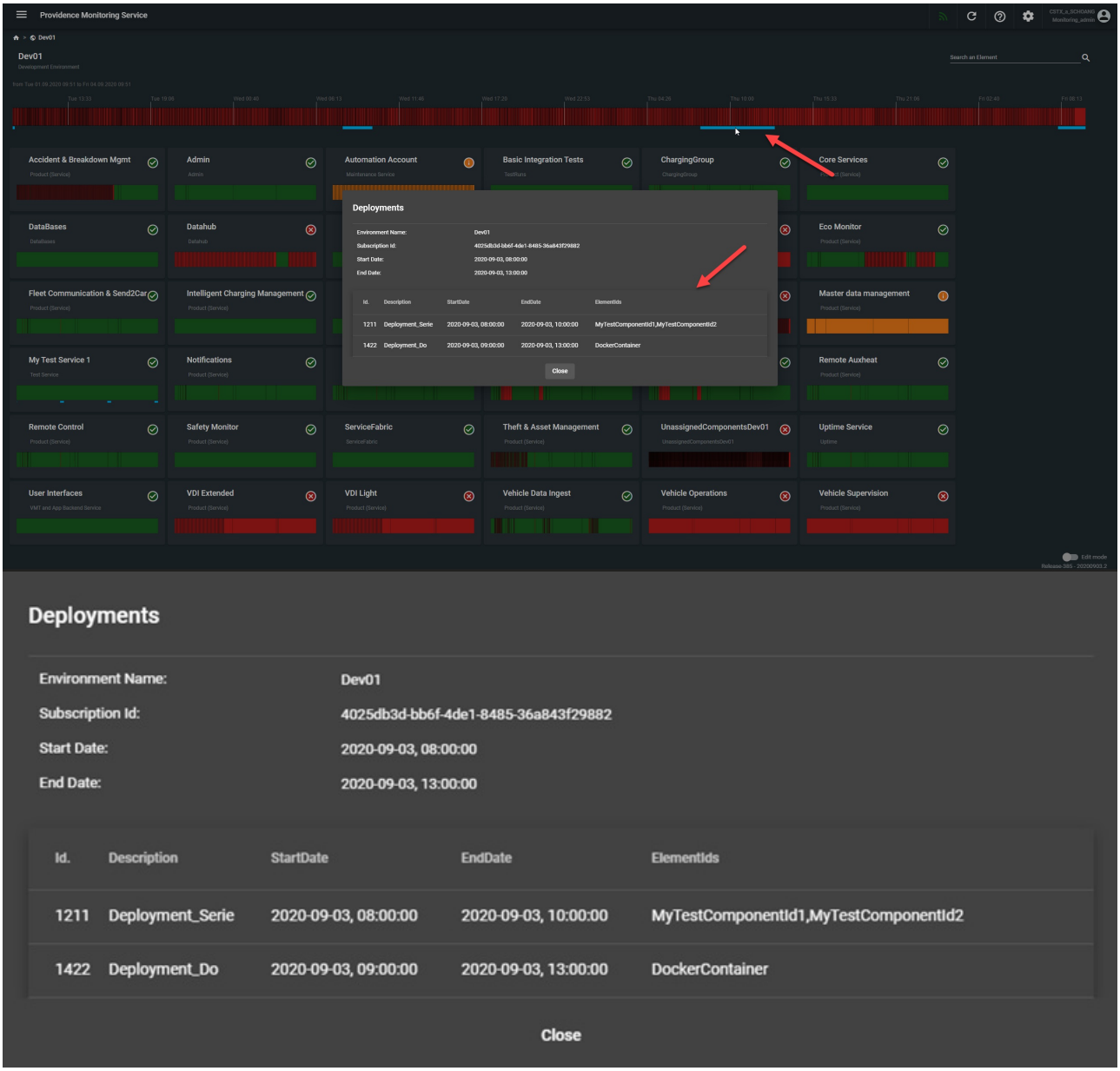
Close

Deployments

The Deployment notifies the UI when in the UI timeline a deployment window should be shown.
The sence for this behaviour is that the Operations-team can consider the deployments if a state of a element is in state "ERROR".
Screenshot of the timeline showing the deployment slot.
The deployment window is marked in blue color, means in that time a deployment of a service is running.

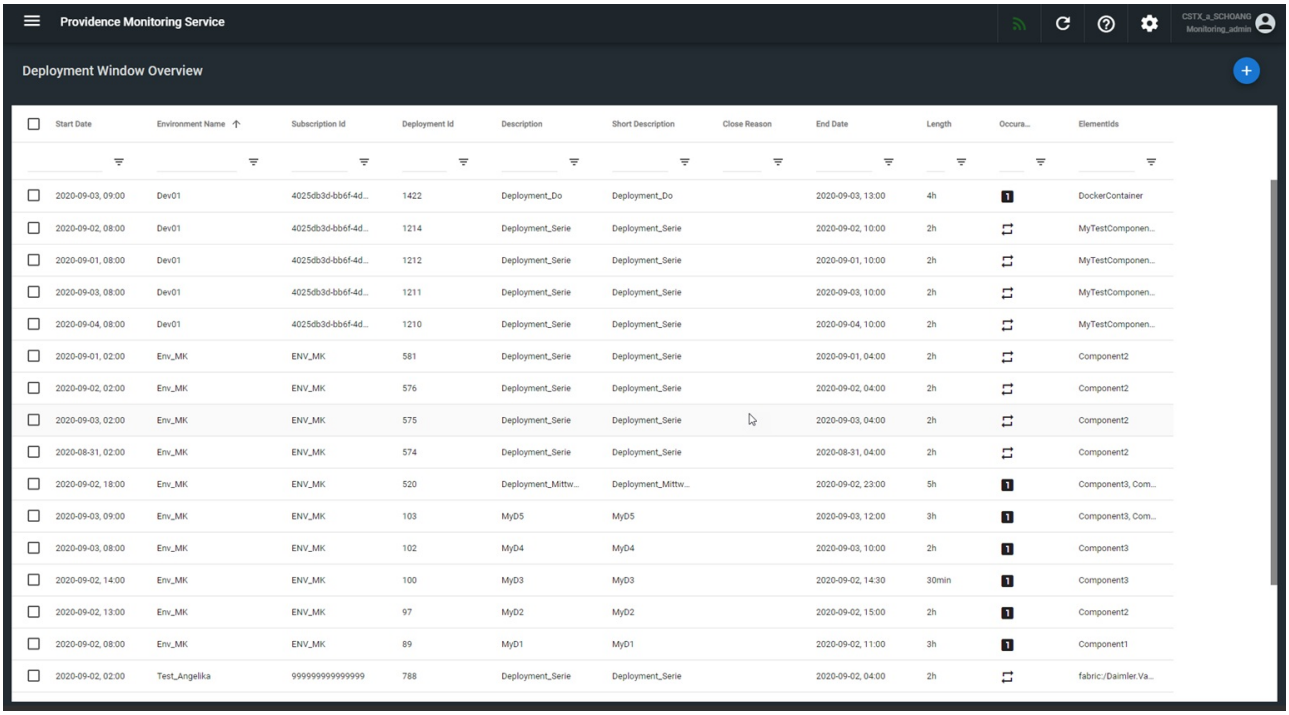


Deployment Details:
With double click on the deployment line, following deployments details are displayed



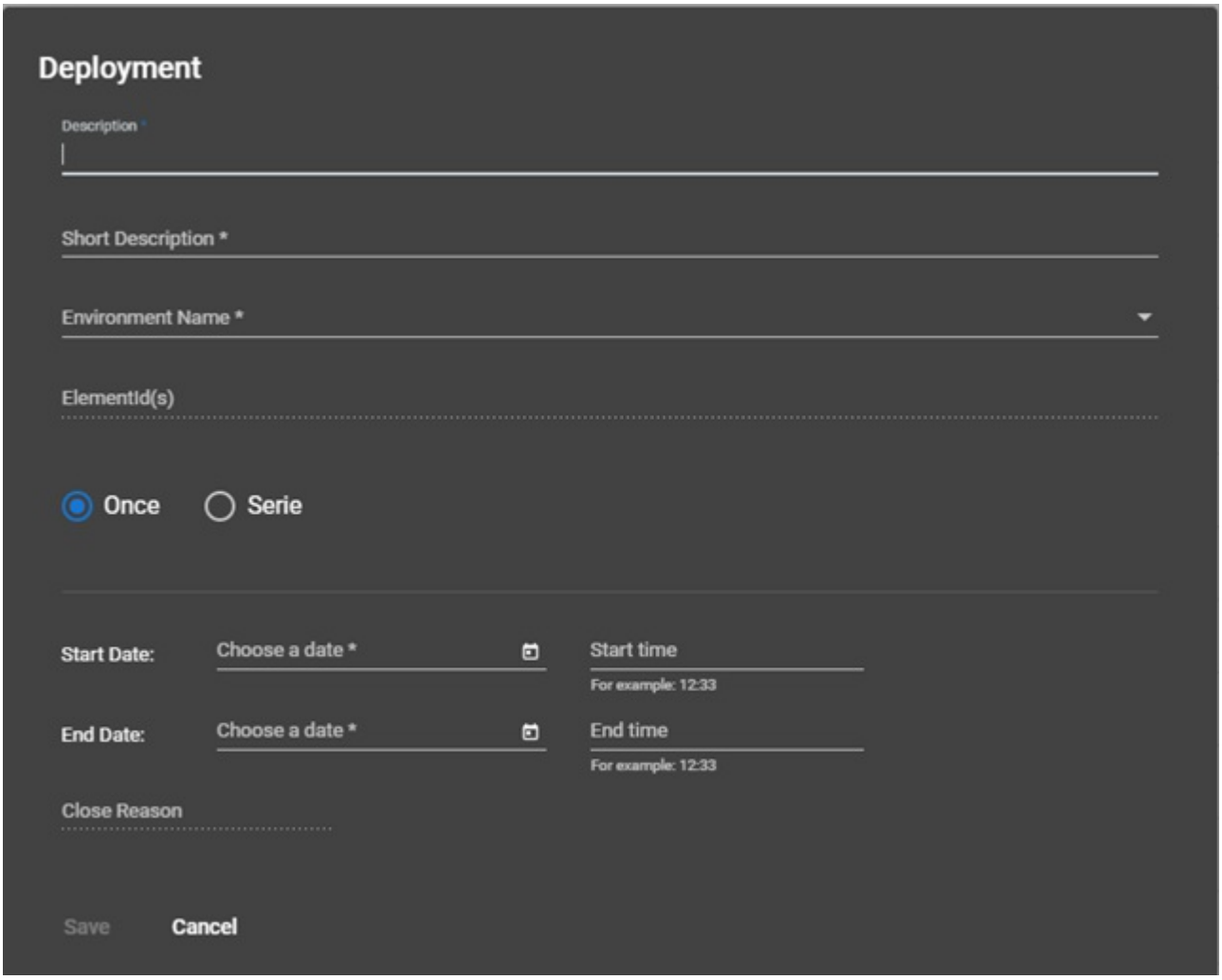
- Environment Name
- Subscription ID
- Start Date & Time
- End Date & Time
- Detail of individual Deployment:
    - Deployment ID
    - Description
    - Start Date & Time
    - End Date & Time
    - ElementID´s

Deployment Overview:



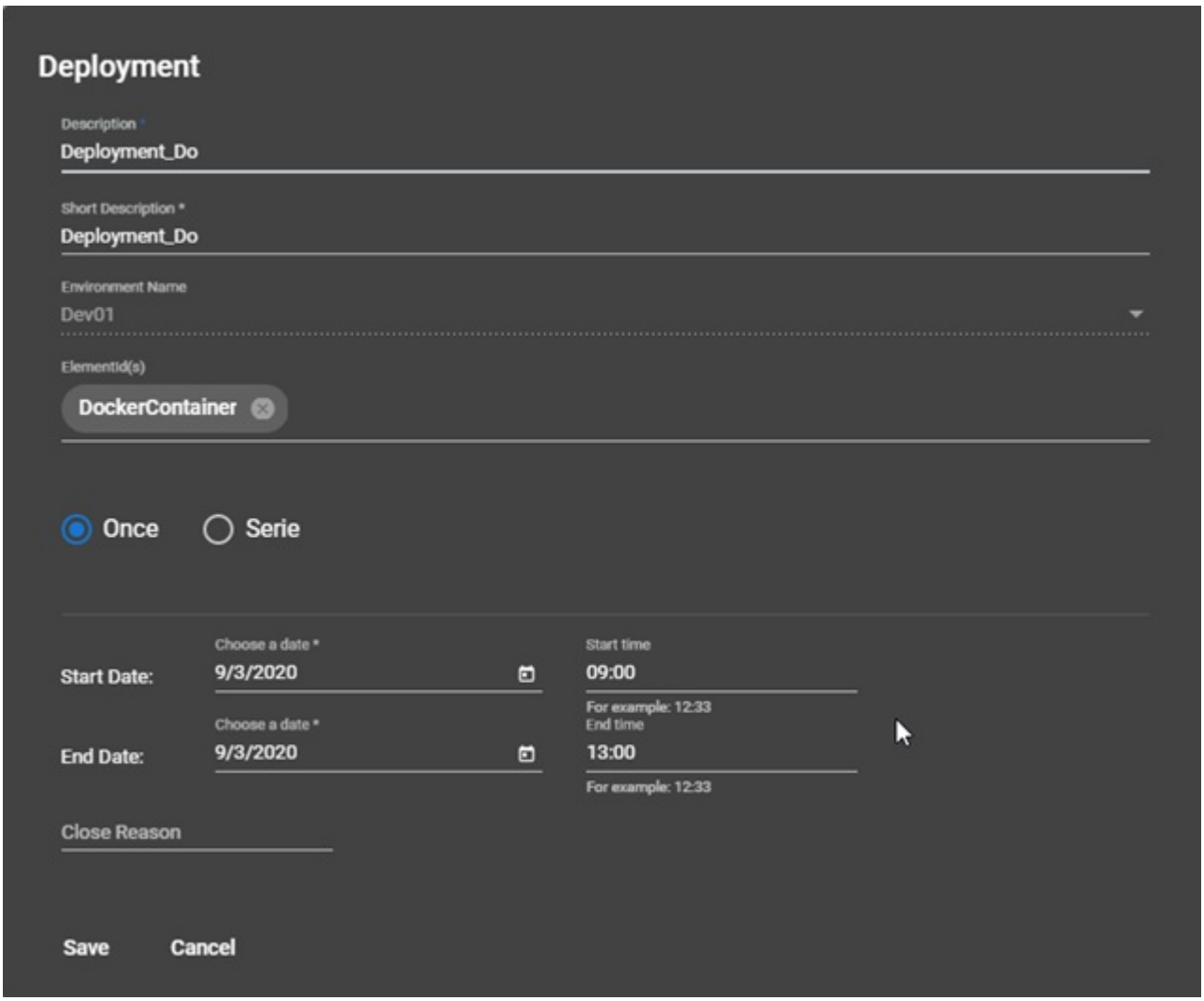
Add new Deployment:

Click on "Add" icon to add an new deployment.



It is possible to either create one single deployment or to create a set of reccuring deployments. If "Once" is chosen the following fields must be filled:

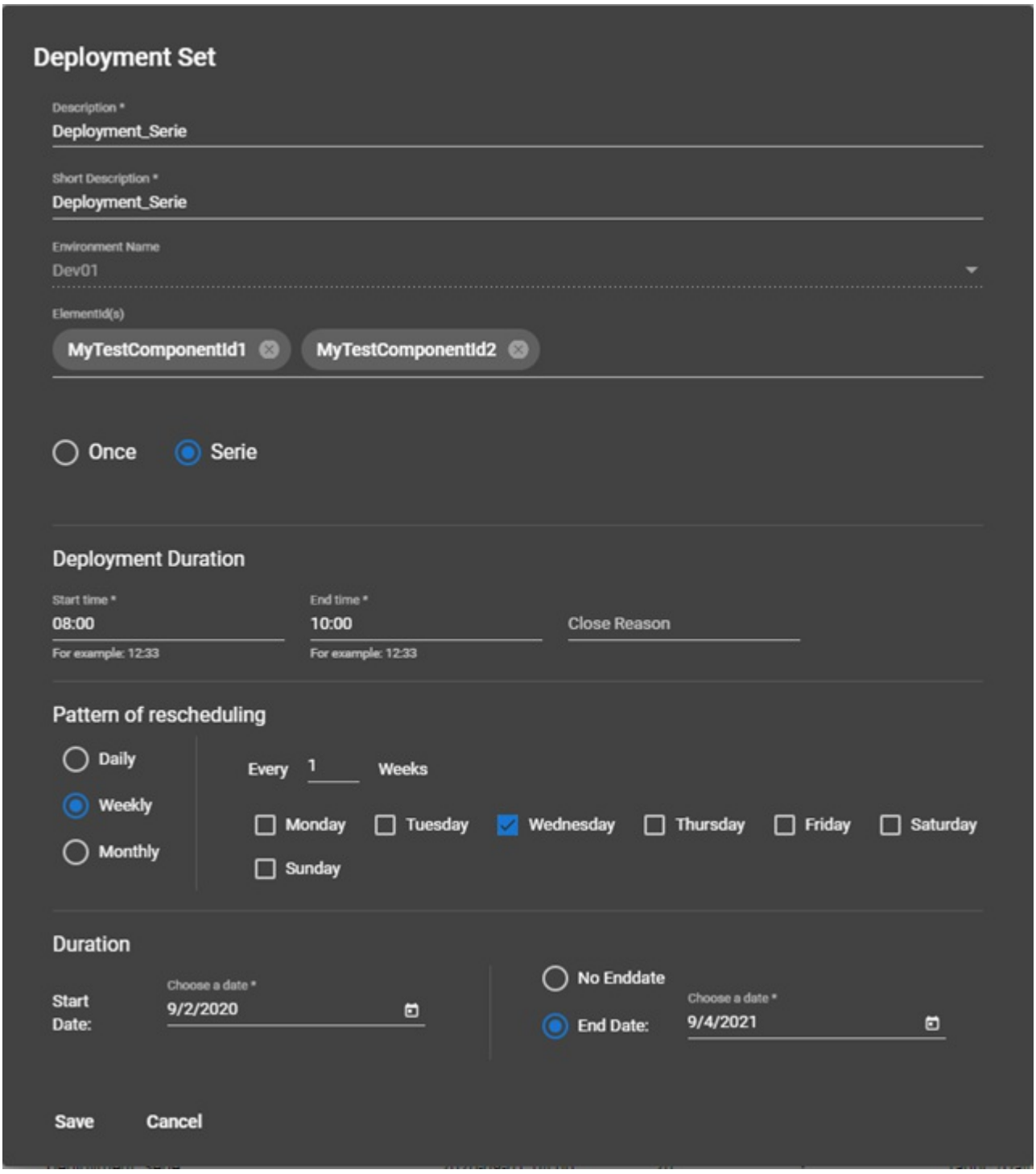
- Description
- Short Description
- Select the environment that is affected
- Select or Set the affected ElementId(s)
- Set the option "Once" deployment
- Set the start date or choose a date
- Set the start time
- Set the end date or choose a date
- Set the end time
- Set Reason for close



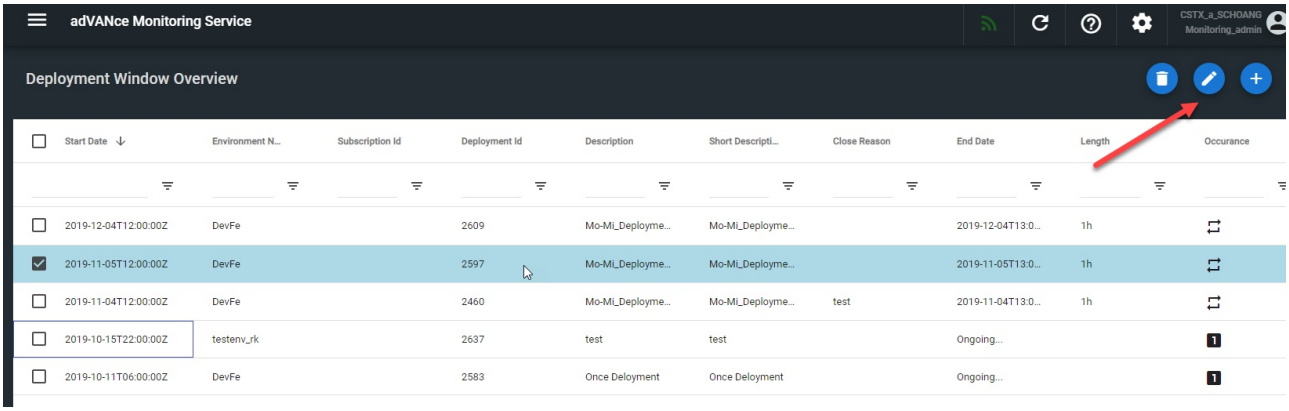
Add Deployment (recurring):
If "Serie" is chosen, to create a whole deployment set, tho following fields must be filled:

- Description
- Short Description
- Select the environment that is affected
- Select or Set the affected ElementId(s)
- Set the option "Serie" deployment
- Set the start time
- Set the end time
- Set Reason for close
- Set the start date
- Choose the pattern of rescheduling (Daily, Weekly, Monthly)
  (e.g. "Daily" and "every 2 days" for a deployment that will reoccur every two days)
    - Daily --> Choose day interval
    - Weekly --> Choose weeks interval (Mo - Su)
    - Monthly --> Choose monthly interval on the ... day of month (Mo - Su)
- Choose the duration date
- Choose "No Enddate"(default + 1 year) or the "End date"

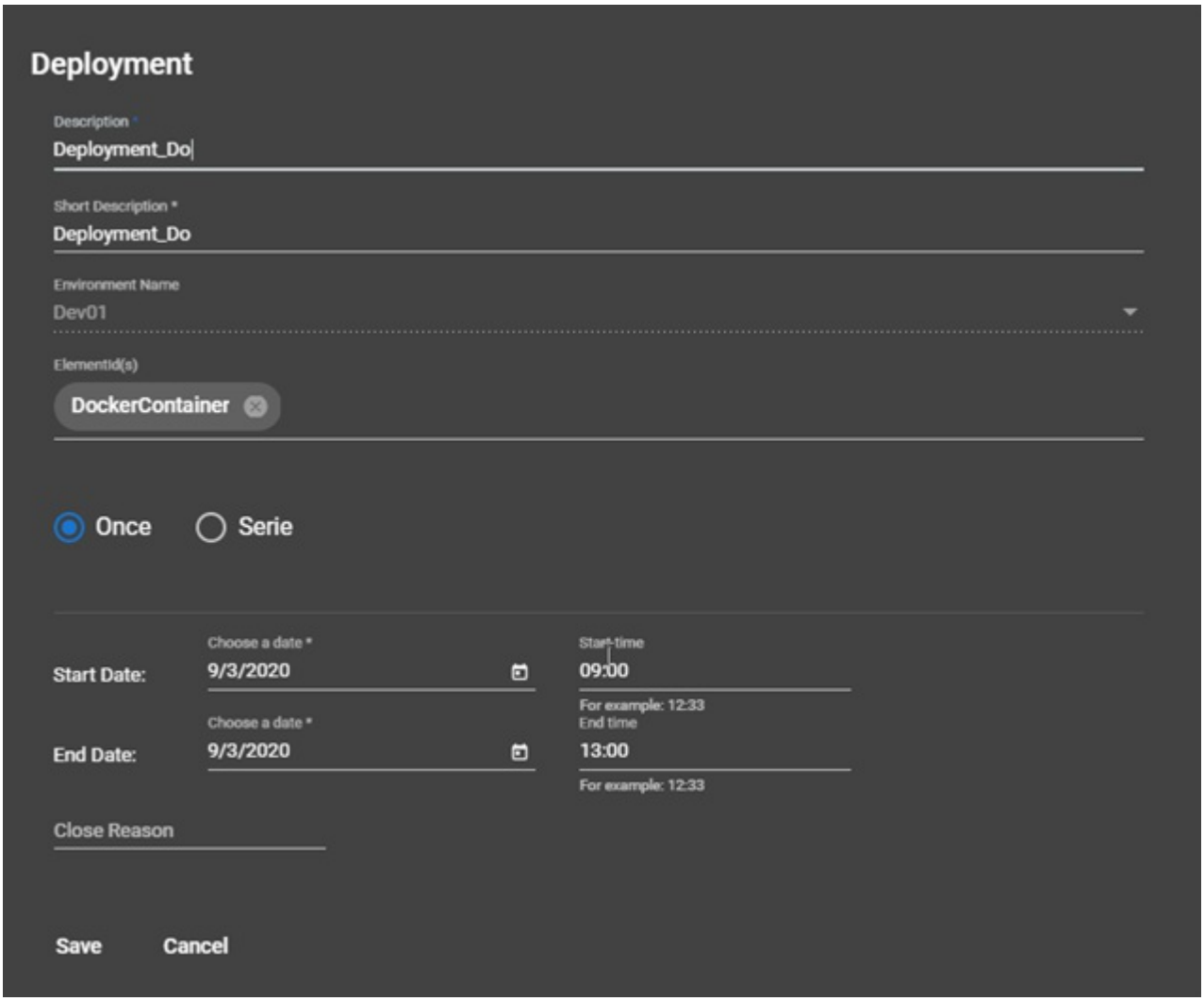
Optional: choose an end date (otherwise the deployment will reoccur for one year)



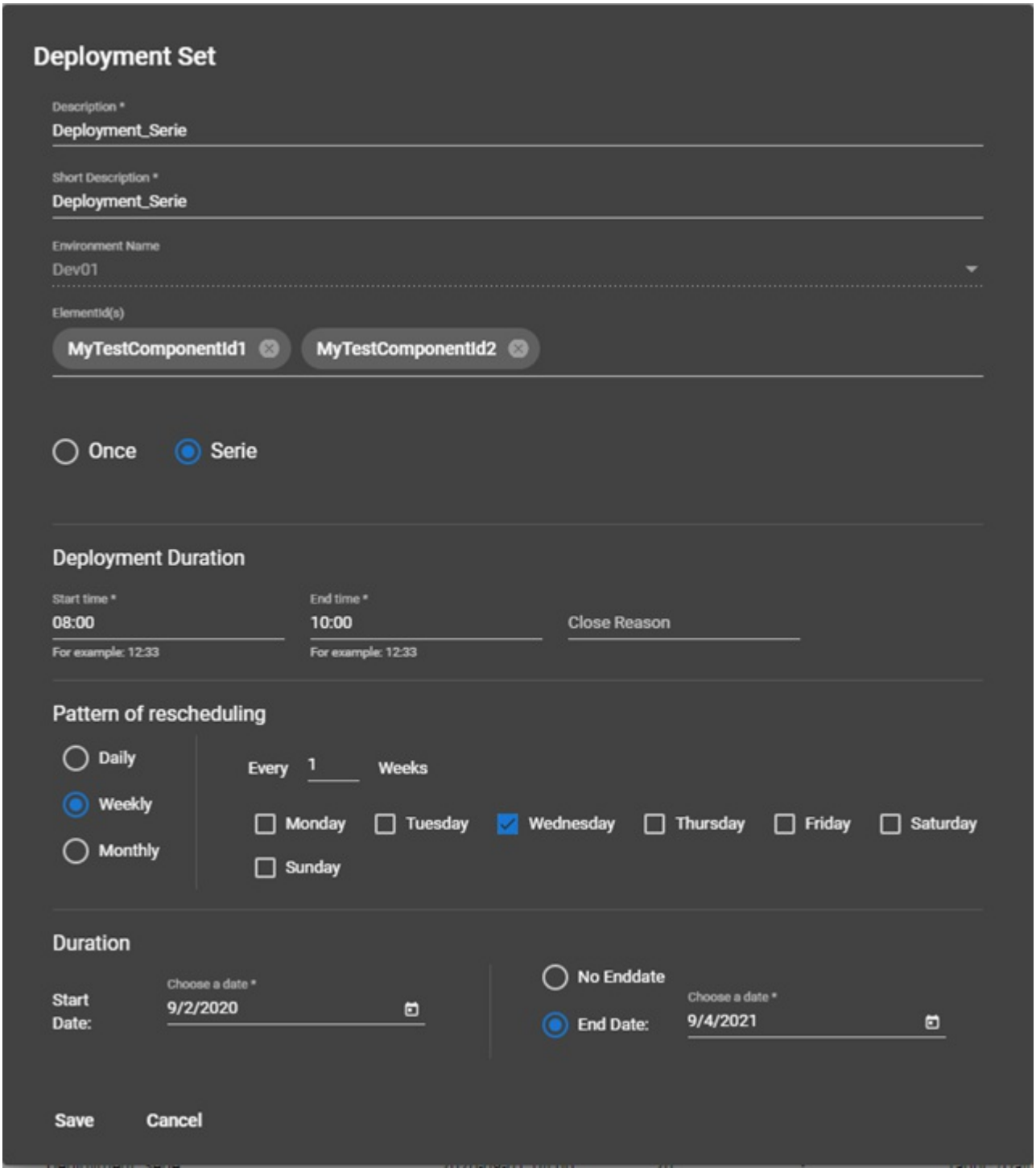
Modify Deployment:
Double-click on the deployment **or** select the deployment and click on the "Edit" icon



Deployment "Once":

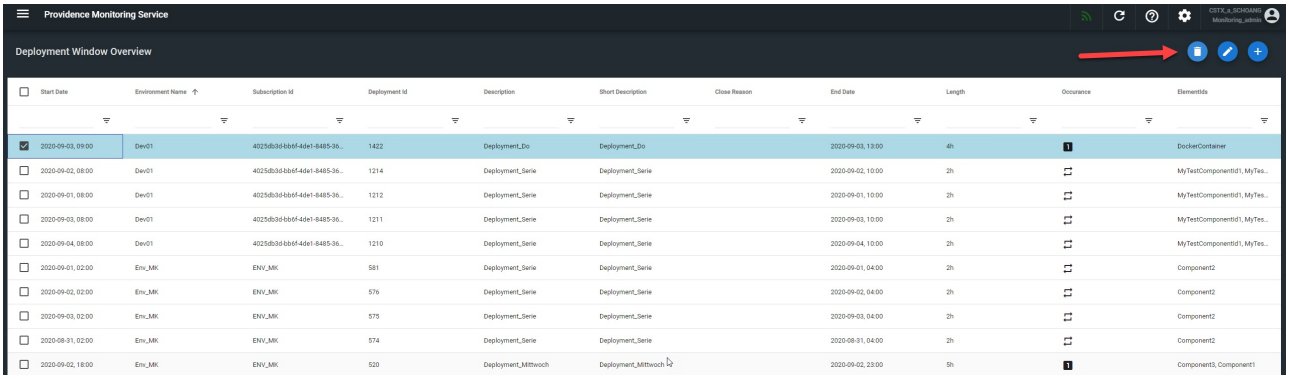


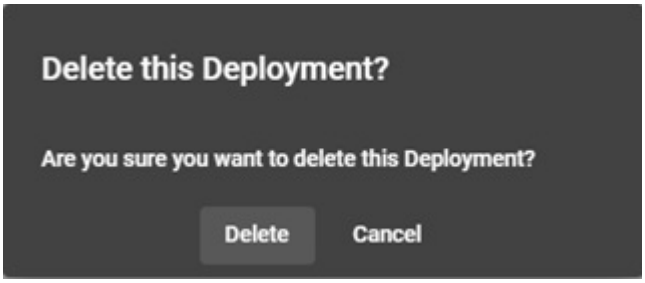
Deployment "Serie":



Delete Deployment (Once/Serie):
It is not possible to delete a serial deployment as a whole, because these are created as individual ones per day of the week. Deletion must be done manually and individually.

Select the deployment and click on the "Delete" Icon


Providence Monitoring Service
Deployment Window Overview
Start Date
Environment Name
Subscription Id
Deployment Id
Description
Short Description
Close Reason
End Date
Length
Occurance
ElementIds
2020-09-03, 09:00 Dev01 4025db3d-bb6f-4de1-8485-36... 1422 Deployment_Do Deployment_Do 2020-09-03, 13:00 4h 1 DockerContainer
2020-09-02, 08:00 Dev01 4025db3d-bb6f-4de1-8485-36... 1214 Deployment_Serie Deployment_Serie 2020-09-02, 10:00 2h MyTestComponentId1, MyTes...
2020-09-01, 08:00 Dev01 4025db3d-bb6f-4de1-8485-36... 1212 Deployment_Serie Deployment_Serie 2020-09-01, 10:00 2h MyTestComponentId1, MyTes...
2020-09-03, 08:00 Dev01 4025db3d-bb6f-4de1-8485-36... 1211 Deployment_Serie Deployment_Serie 2020-09-03, 10:00 2h MyTestComponentId1, MyTes...
2020-09-04, 08:00 Dev01 4025db3d-bb6f-4de1-8485-36... 1210 Deployment_Serie Deployment_Serie 2020-09-04, 10:00 2h MyTestComponentId1, MyTes...
2020-09-01, 02:00 Env_MK ENV_MK 581 Deployment_Serie Deployment_Serie 2020-09-01, 04:00 2h Component2
2020-09-02, 02:00 Env_MK ENV_MK 576 Deployment_Serie Deployment_Serie 2020-09-02, 04:00 2h Component2
2020-09-03, 02:00 Env_MK ENV_MK 575 Deployment_Serie Deployment_Serie 2020-09-03, 04:00 2h Component2
2020-08-31, 02:00 Env_MK ENV_MK 574 Deployment_Serie Deployment_Serie 2020-08-31, 04:00 2h Component2
2020-09-02, 18:00 Env_MK ENV_MK 520 Deployment_Mittwoch Deployment_Mittwoch 2020-09-02, 23:00 5h 1 Component3, Component1


Delete this Deployment?
Are you sure you want to delete this Deployment?
Delete
Cancel

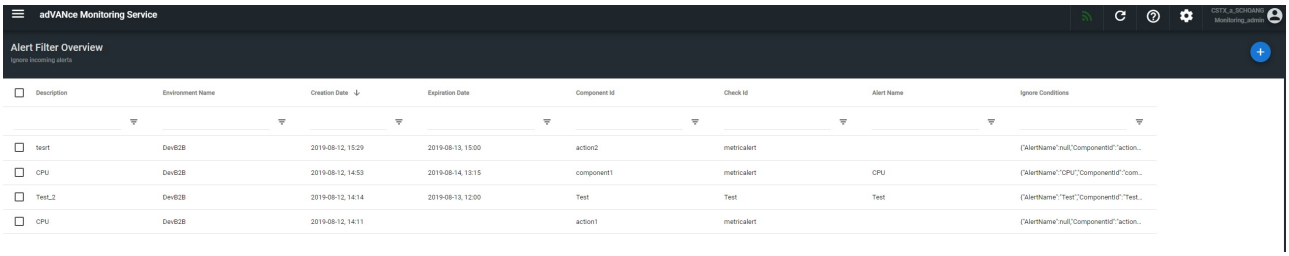
***

Alert Ignore Rules

The Alert Ignore Overview of the Providence Service supports operations for creating, updating and deleting alert ignore rules.

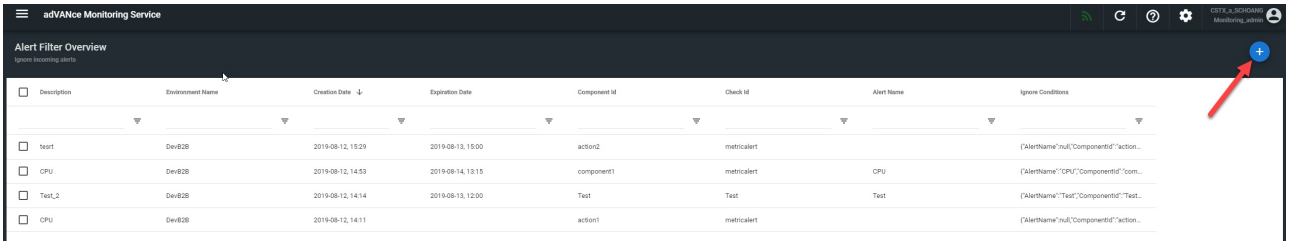
Alert ignore rules can be used to mute specifc alerts. For example all alerts with a specific CheckId can be muted to avoid StateTransitions resulting from their occurrence in the Providence Dashboard.

Alert Ignore Rule Overview:



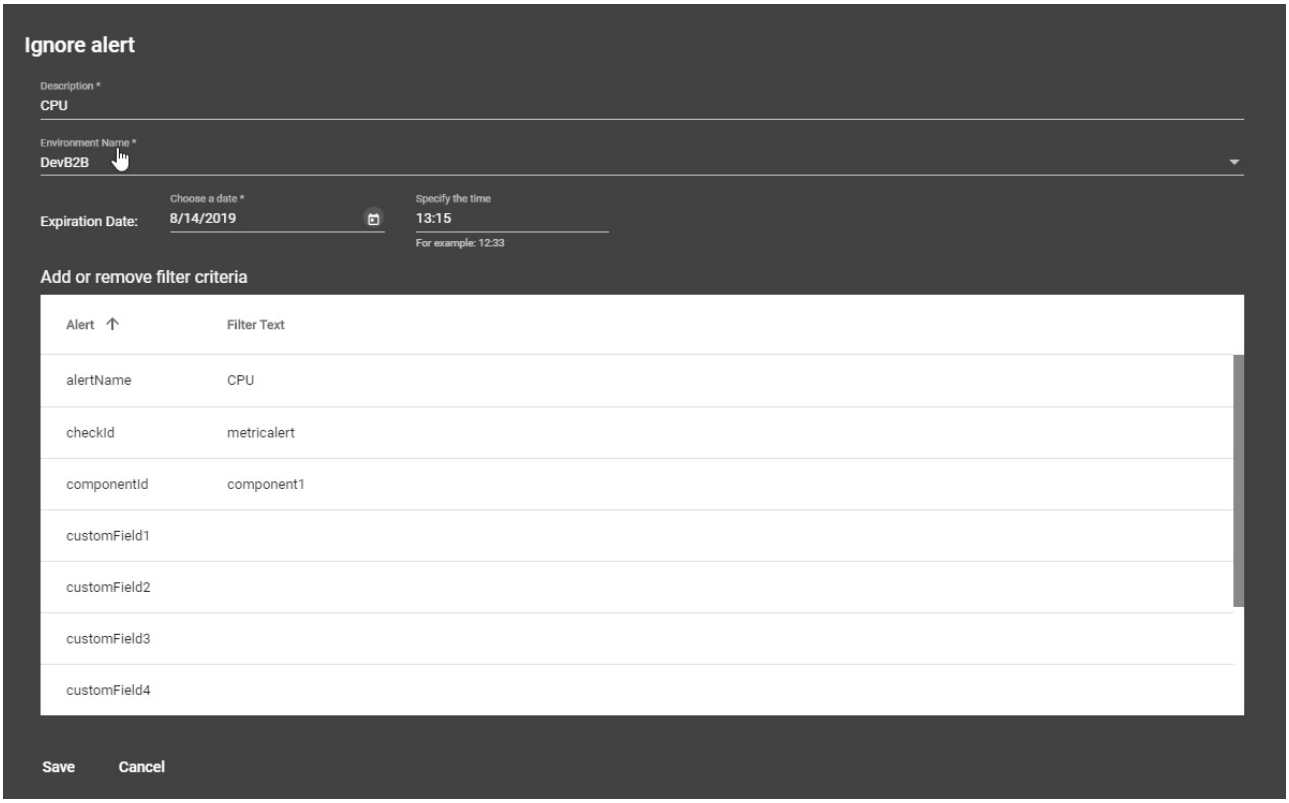
Add new Alert Ignore Rule:

Click on "Add" icon to add new alert ignore rule.



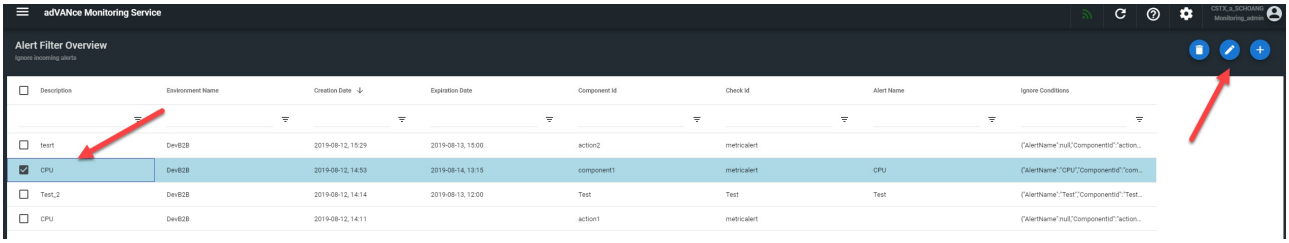
The following fields must be filled in:

- Description
- Select the environment that is affected
- Set the expiration date
- Select the alert property which shall be ignored and set a filter text (combination of multiple properties possible)

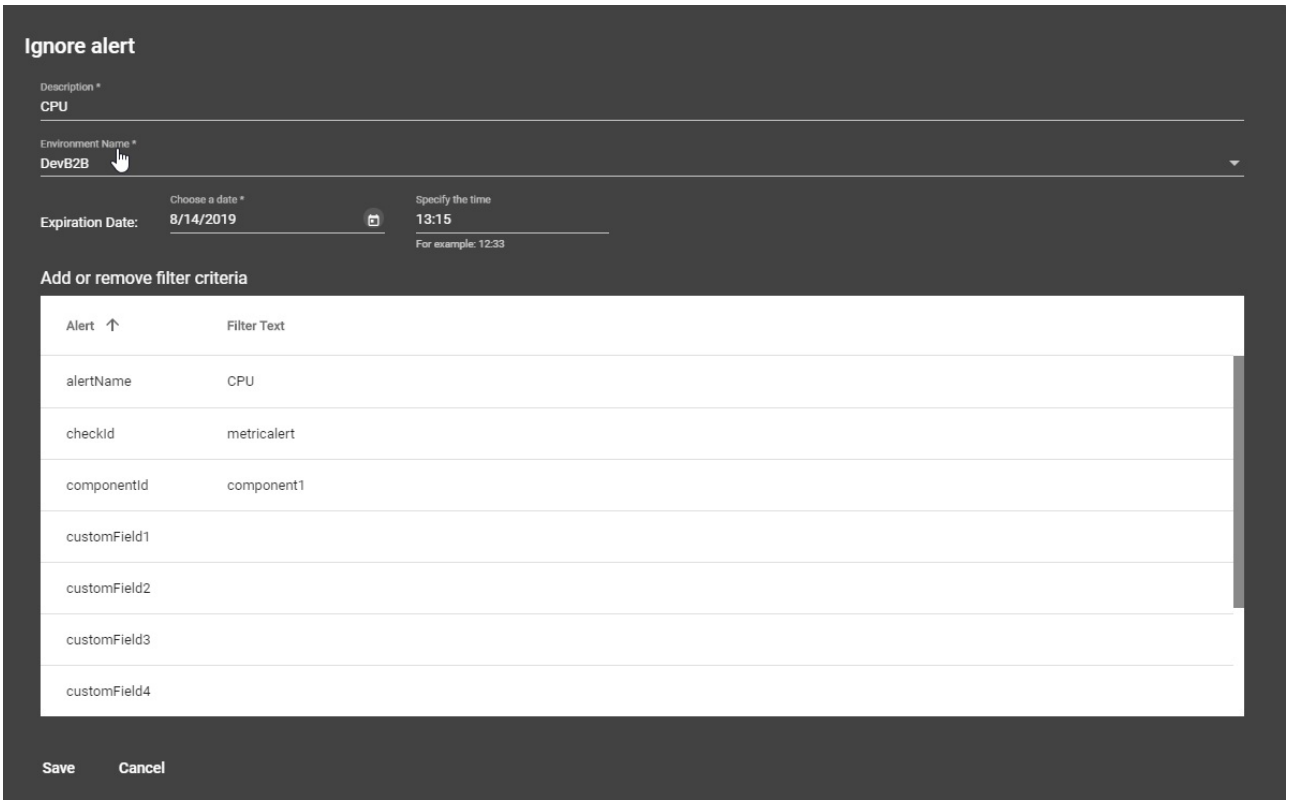


Modify Alert Ignore Rule:

Double-click on the rule you want to update **or**
Select the alert filter and click on the "Edit" icon

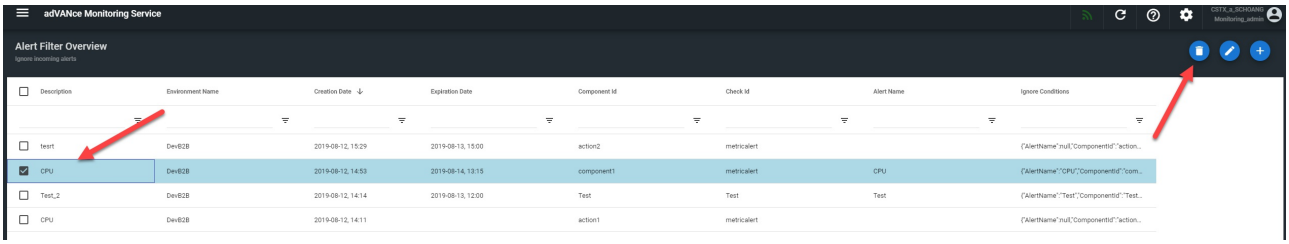


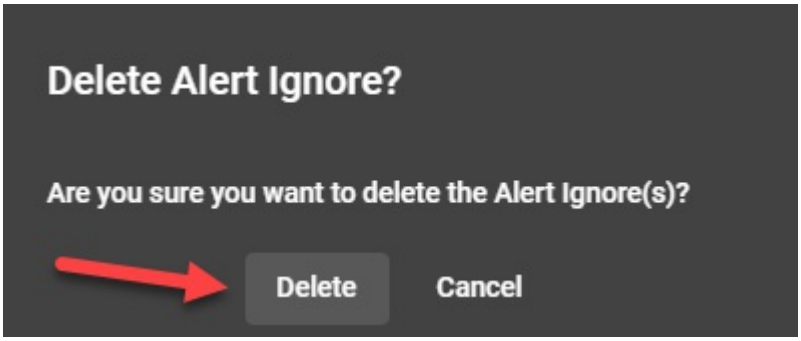
Make changes and save to complete.



Delete Alert Ignore Rule:

Select the alert ignore rule and click on the "Delete" icon

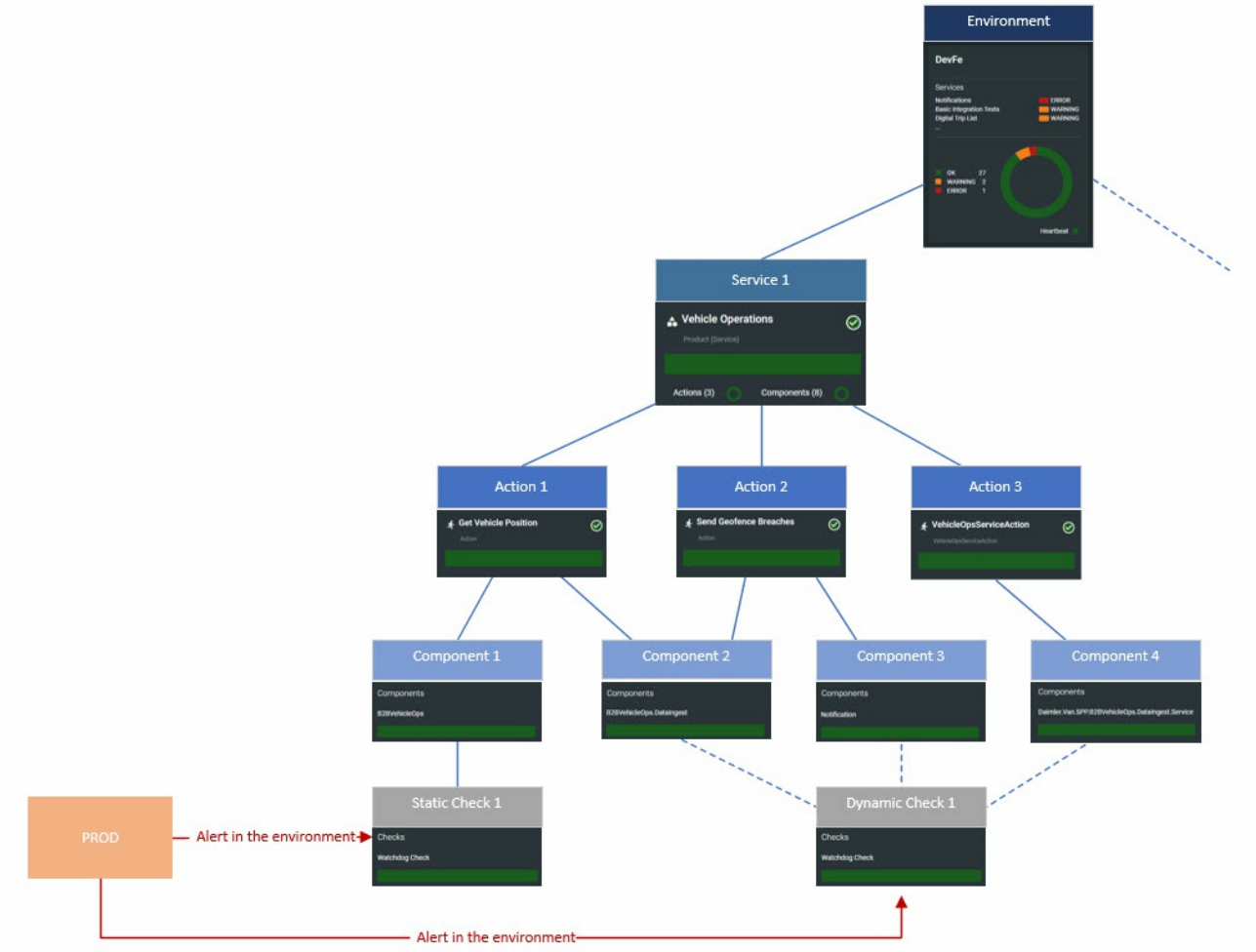




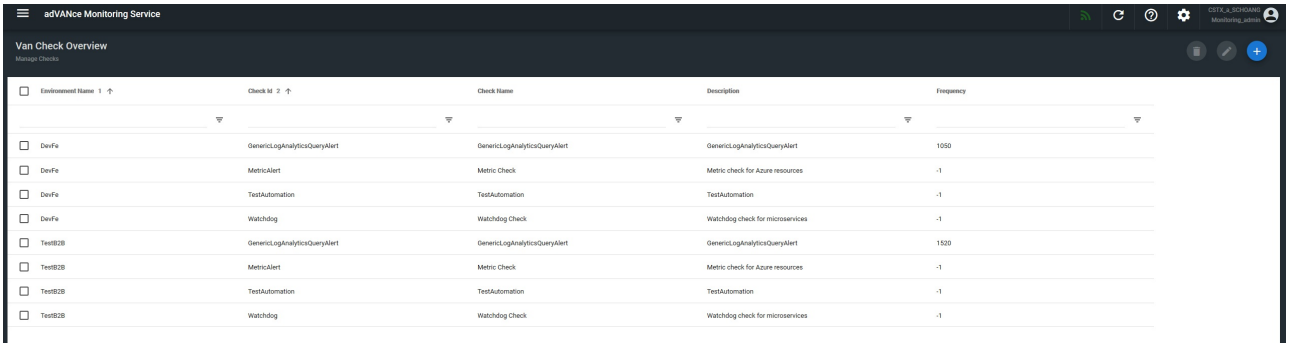
Checks

Checks are used as a mechanism to avoid unknown data to be processed by the Providence service backend. Only alerts with a valid and known "CheckId" property are processed by the service. If a valid "CheckId" was found the alert changes the state of all elements which are linked to the Check within the environment.Depending on the state defined in the alert the state then changes to "ERROR", "WARNING" or "OK".

Besides the functionality as a "Validation Gateway" for alerts checks are also used to configure other different features like for example the "Reset to Green" feature. You can read more about this feature within the "Internal Jobs" section.

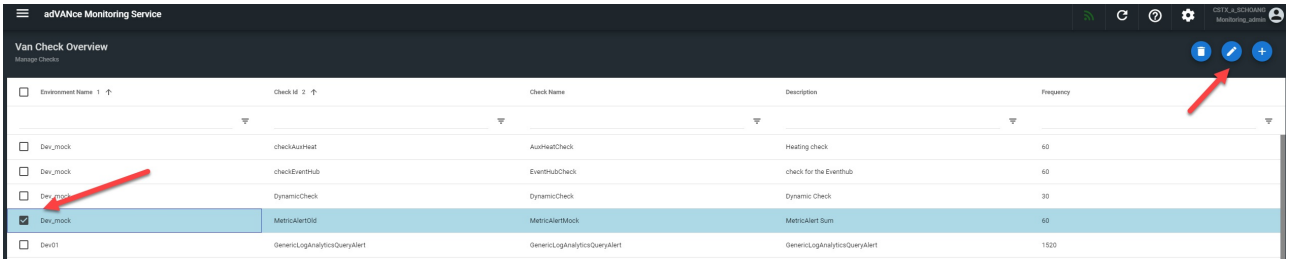


Check Overview:



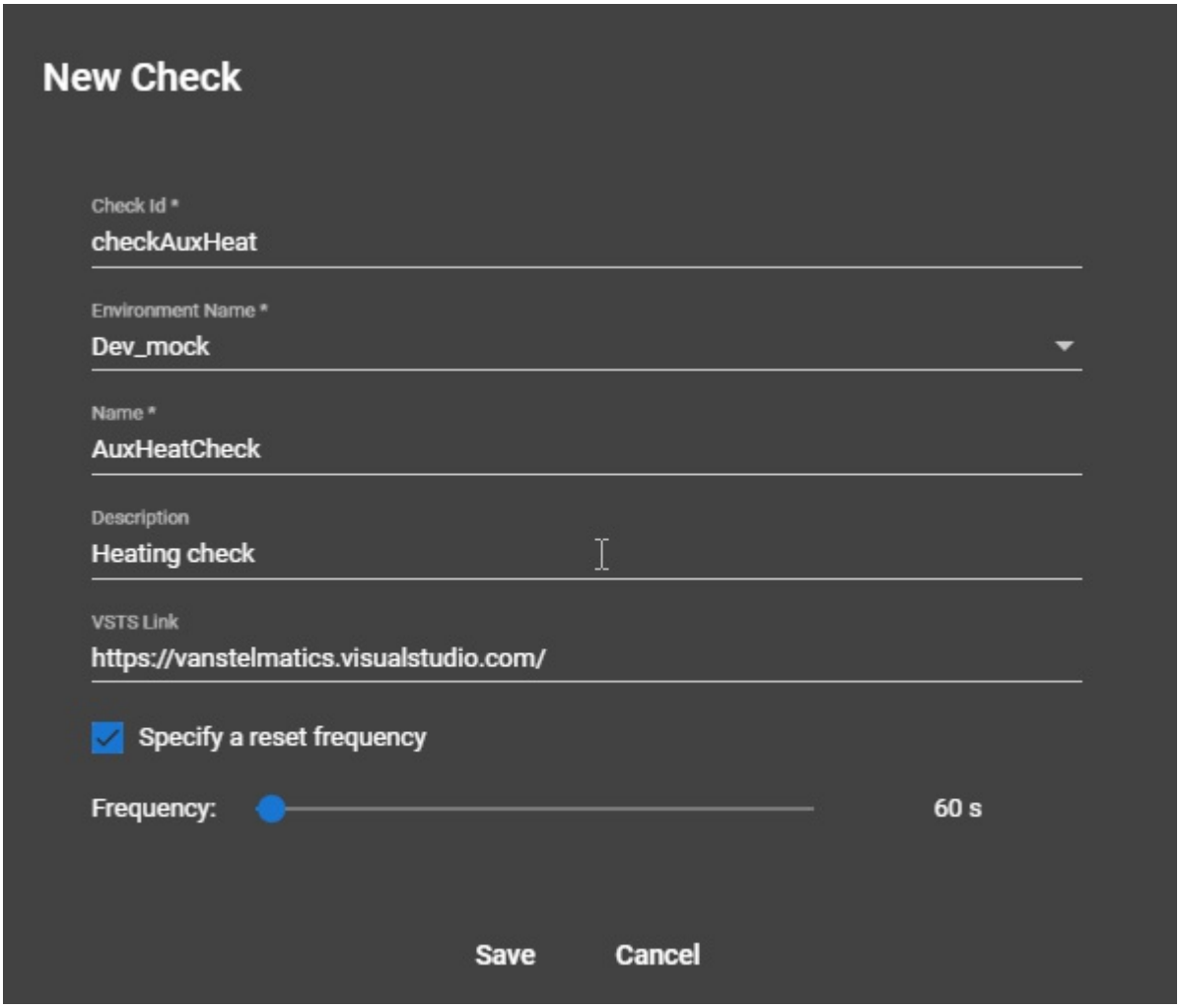
Add Check:

Click on "Add" icon to add a service check.



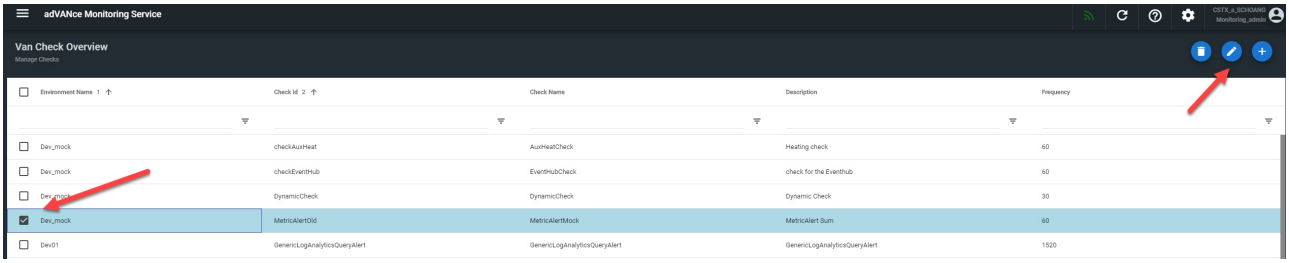
The following fields must be filled in:

- Check Id
- Select the environment that is affected
- Name
- Description
- VSTS Link
- Select "Specify a reset frequency"
- Set the frequency in second (10 - 1800 seconds)
- Save the settings

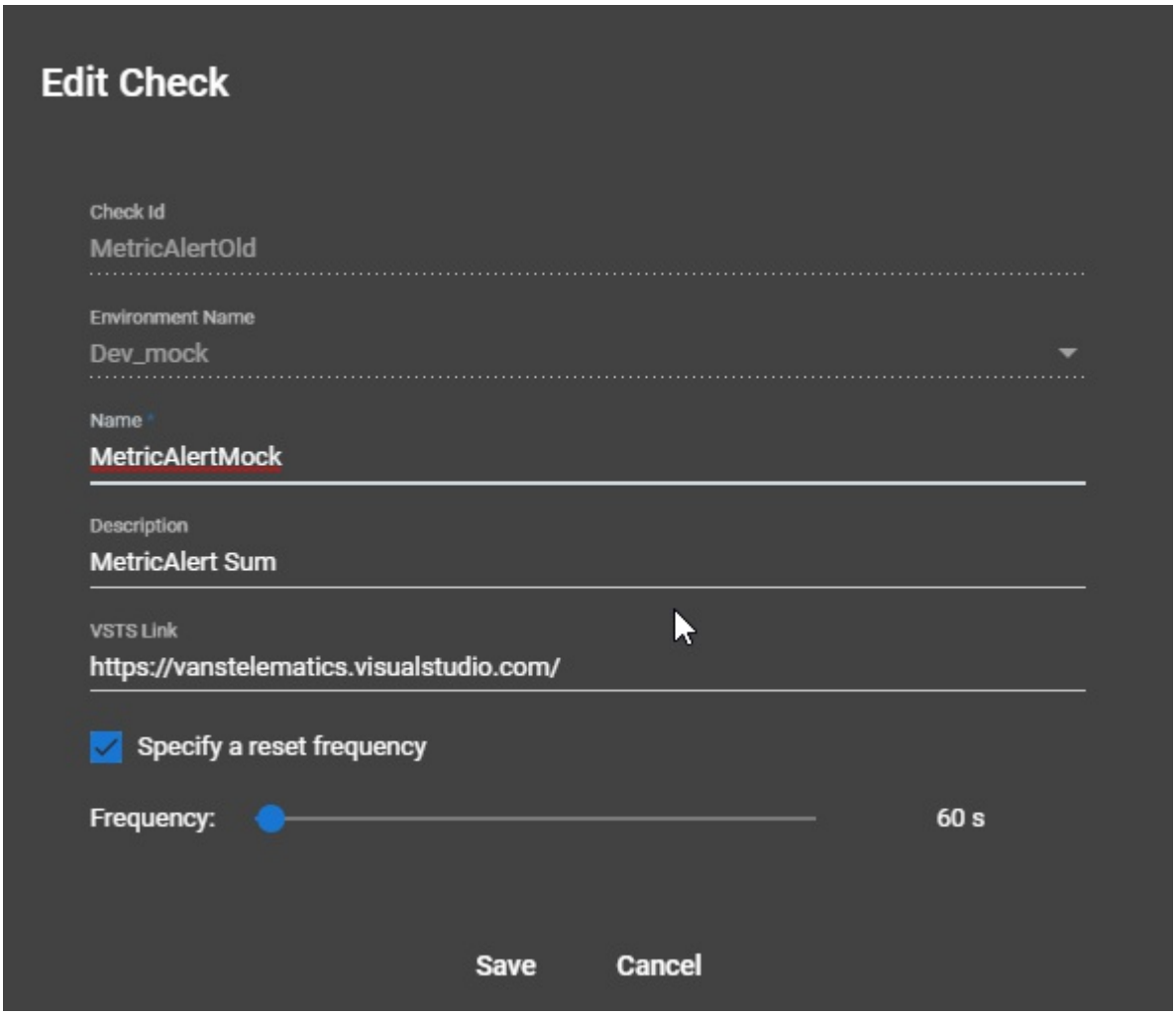


Modify Check:

Double-click on the service check **or** Select the service check and click on the "Edit" icon

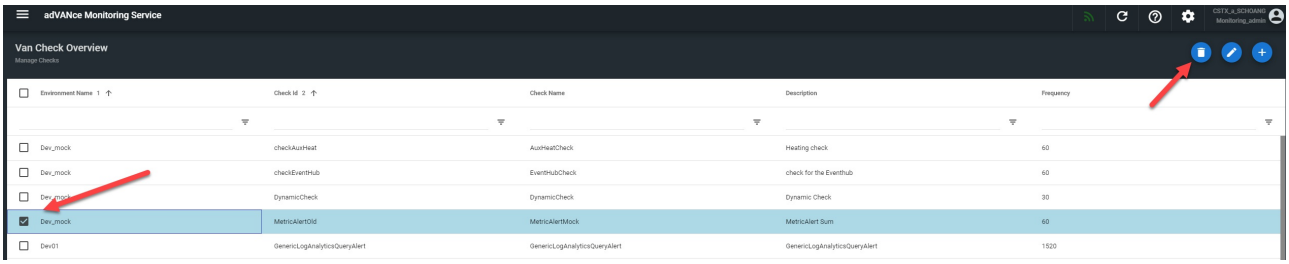


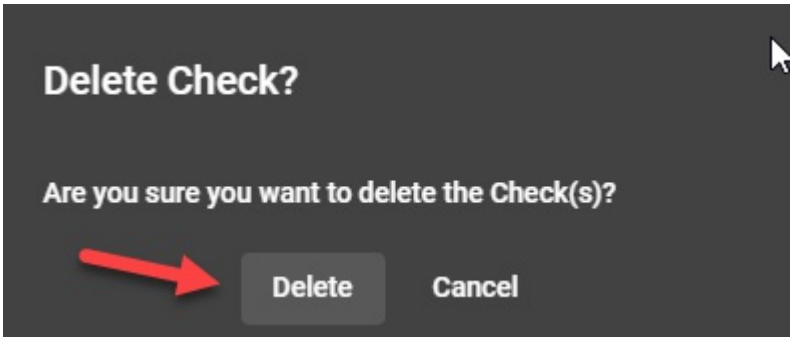
- Make changes and save to complete.



Delete Check:

Select the service check and click on the "Delete" icon

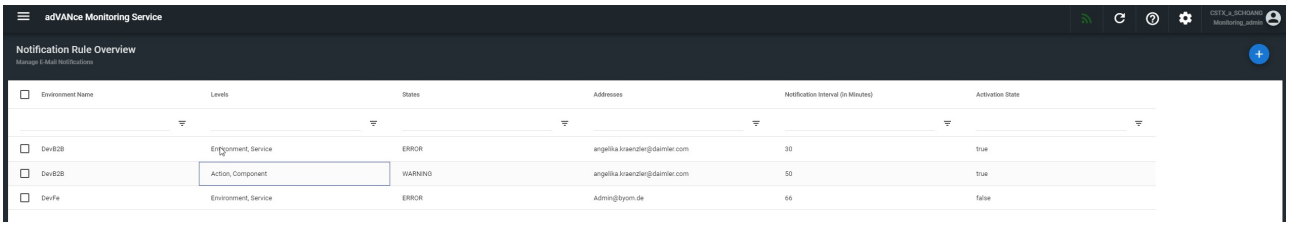



Delete Check?
Are you sure you want to delete the Check(s)?
Delete
Cancel

Notificaton Rules

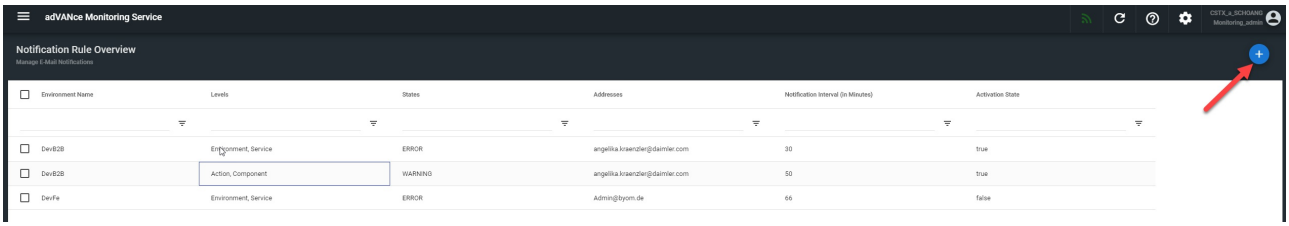
In order to receive emails from the Providence Service notification rules have to be configured.
These rules contain information about the user which wants to be notified, the elements the user want to be notified about and also a notification delay interval after what time he want to be notified about specific state changes.

Notification Overview:



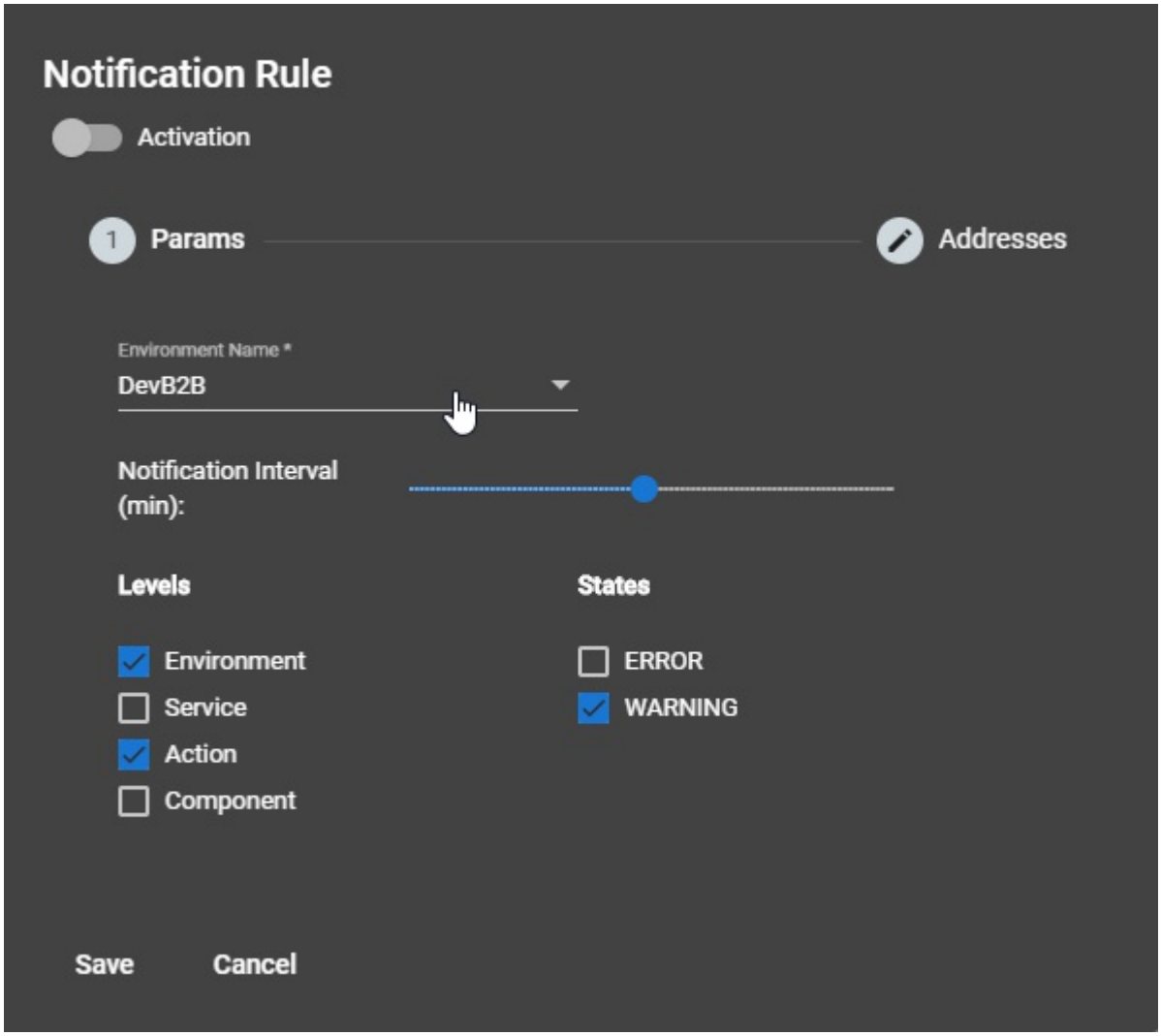
Add notification rule:

Click on "Add" icon to add a notification rule.

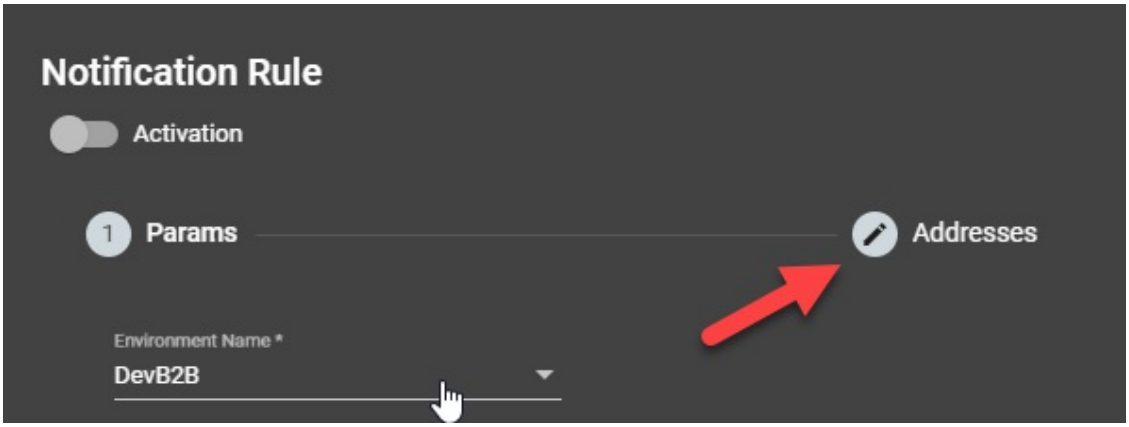


The following fields must be filled:

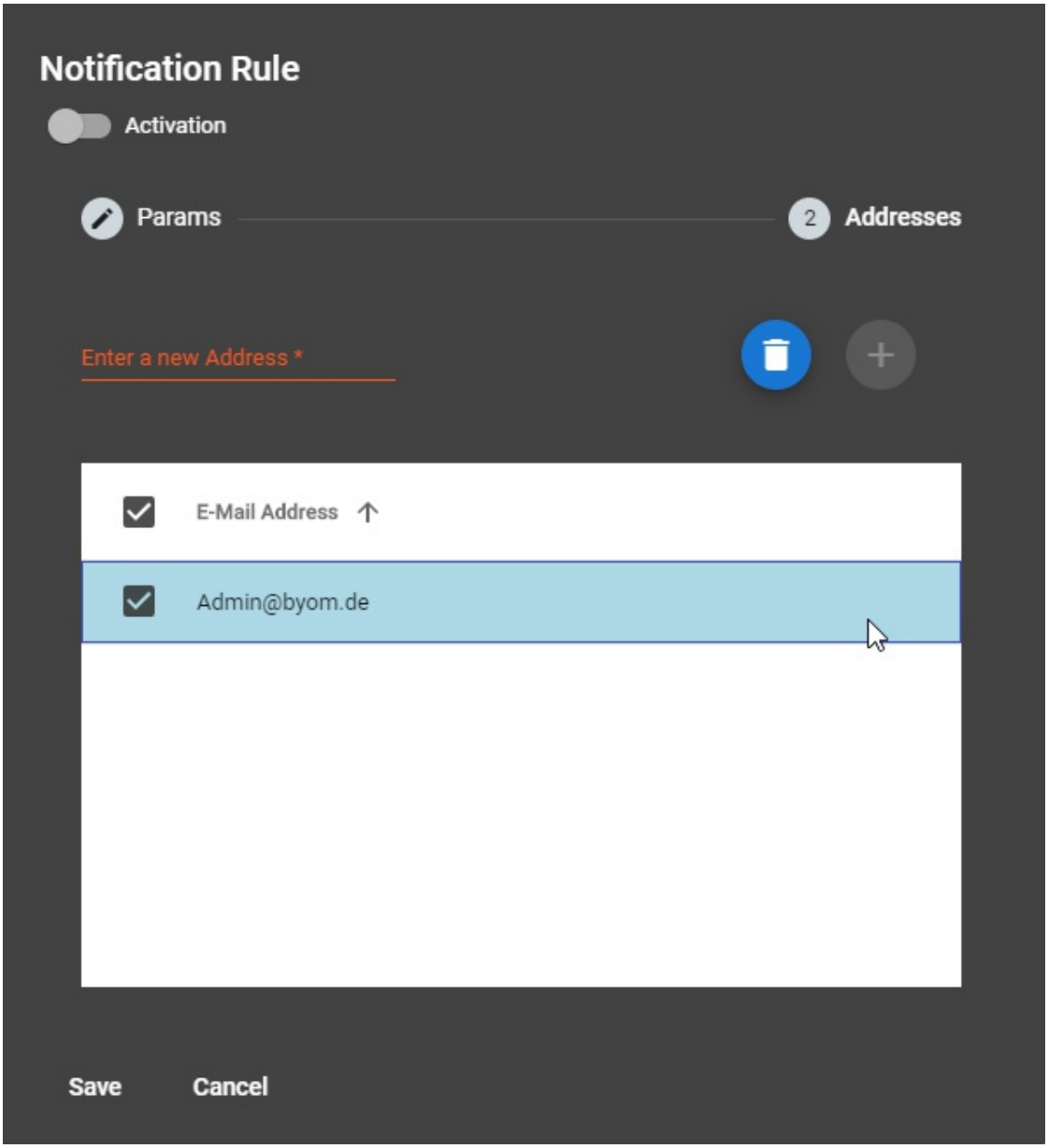
- Toggle "Activation"
- Parameter settings:
    - Select environment
    - Set notification delay interval (from 1 - 90 minutes)
    - Select levels (Environment, Service, Action, Component)
    - Select states (Error, Warning)



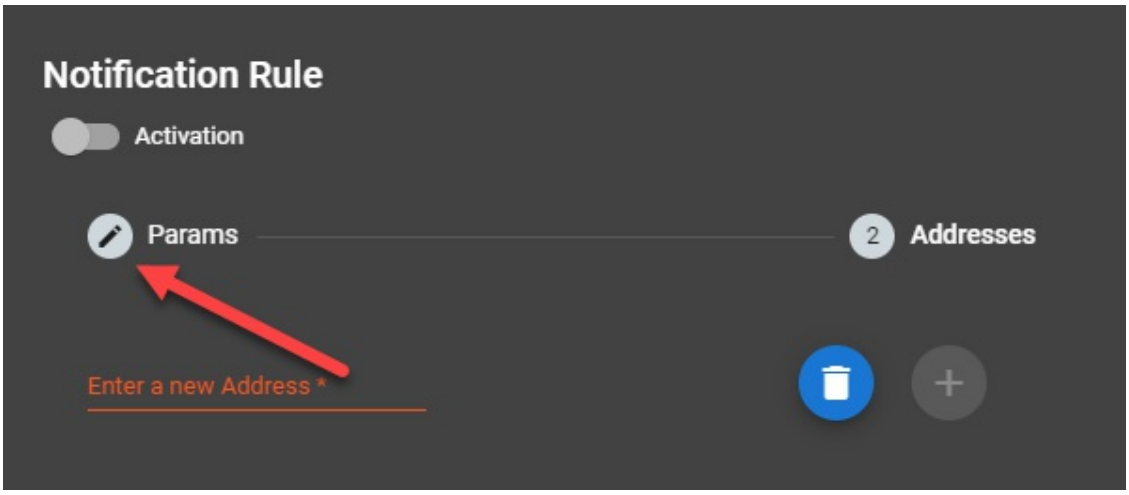
- Address settings:
  - Click on the "Edit" icon to set the Adresse:



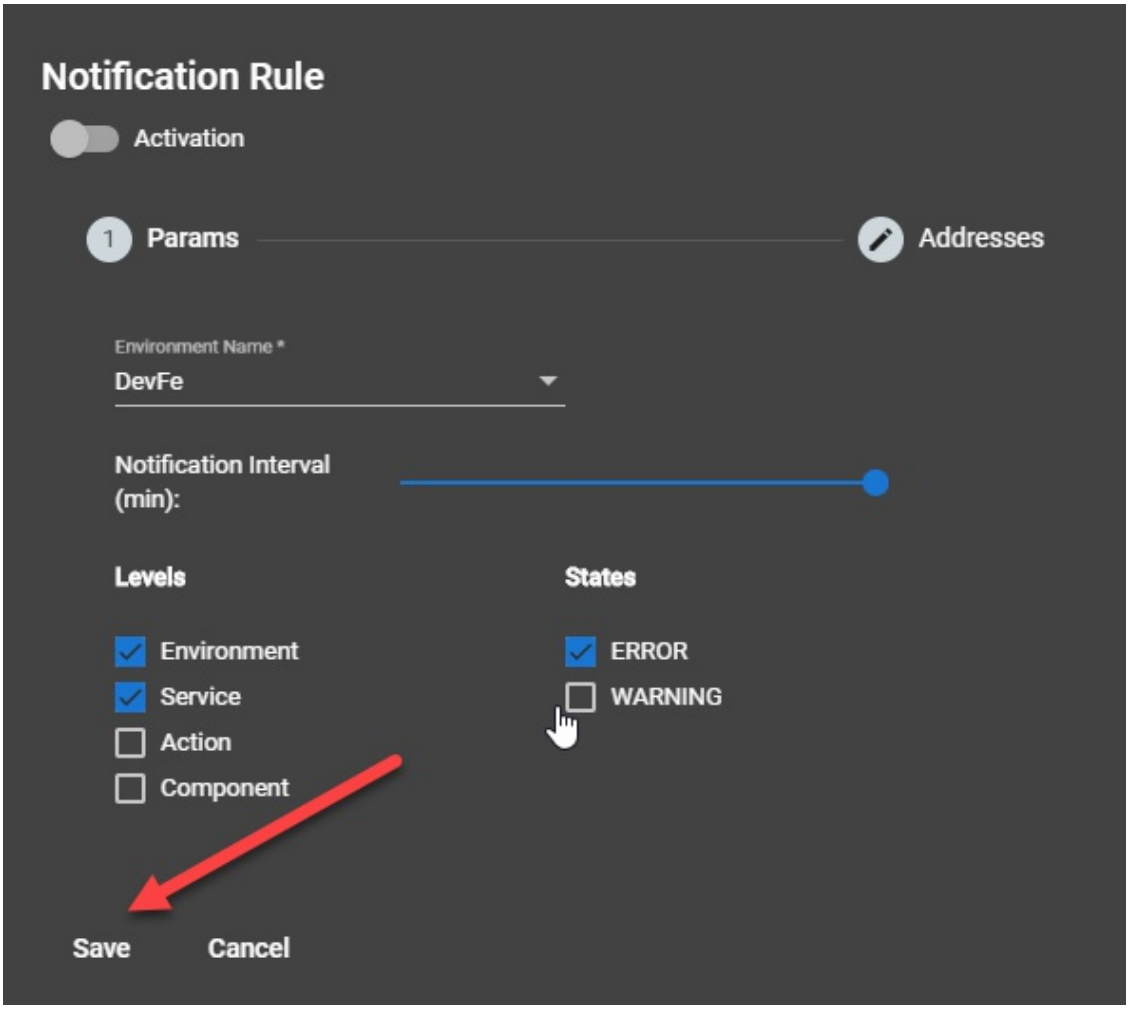
  - Enter a new e-mail adress and click on the "Add" icon
  - Select the new E-Mail address



- Click on the "Edit" icon to set the Parameter:

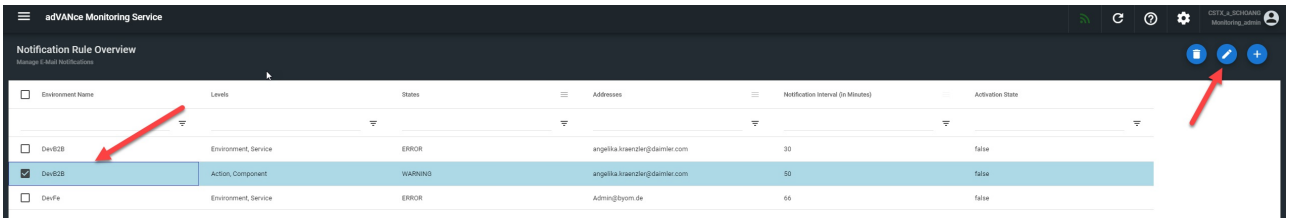


- Save the notification settings

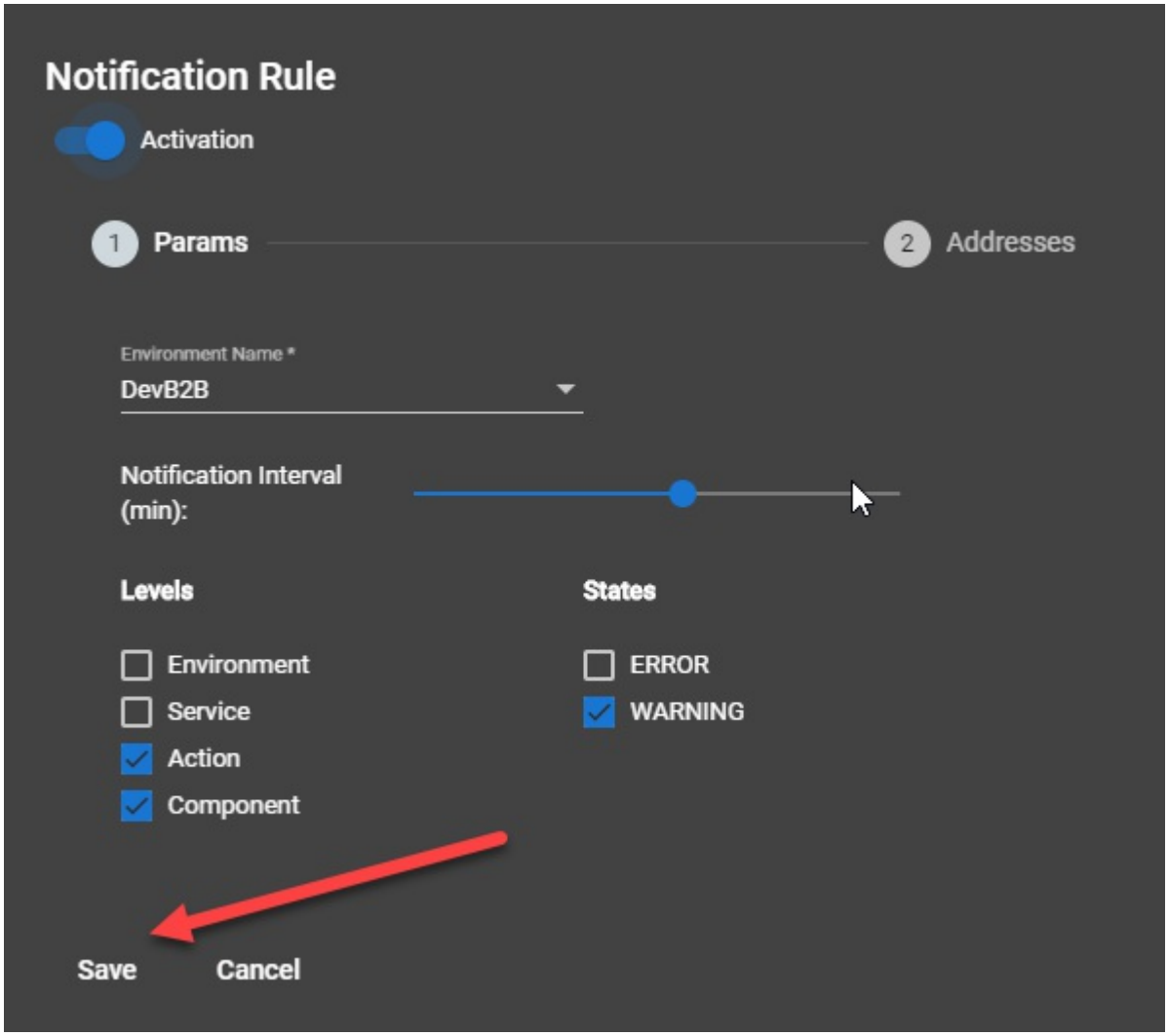


Modify notification rule:

Double-click on the notification rule **or**
Select the service notification and click on the "Edit" icon

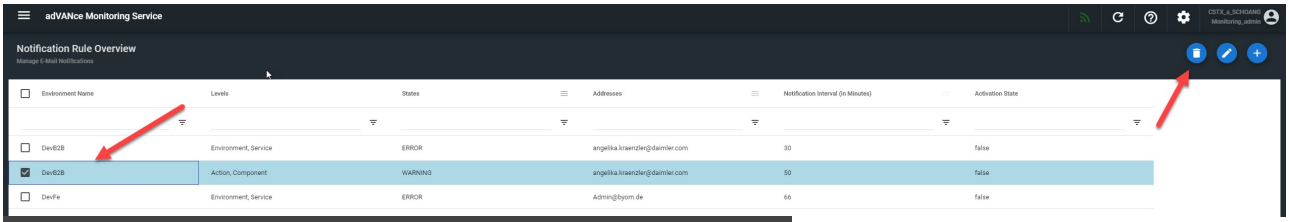


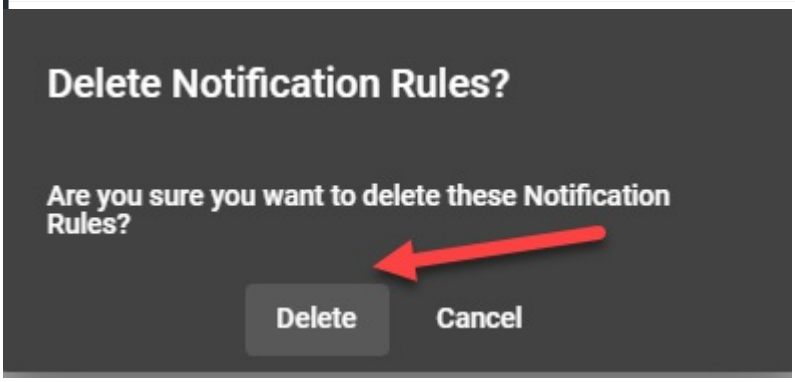
- Make changes and save to complete.



Delete service notification rule:

Select the notification rule and click on the "Delete" icon





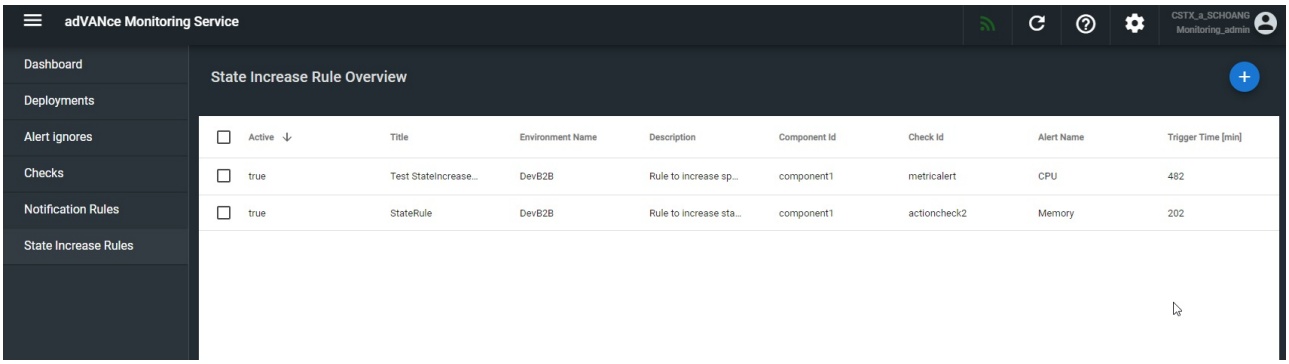
State Increase Rules

The State Increase Rule overview of the Providence Service supports operations for creating, updating and deleting specific rules.

State Increase Rules can be used to increase the state of specific alerts after a specified period of time.

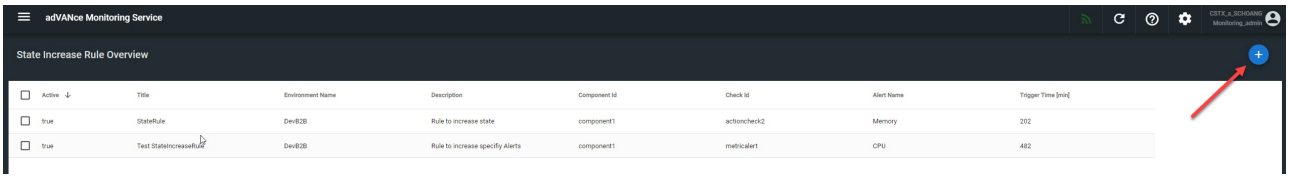
The state increase job checks in a specific period of time (configurable in app settings) if a check is in the state "WARNING" for a specific duration.
If so the job increases the state to "ERROR" to give this component more attention within the dashboard.

State Increase Rule overview:



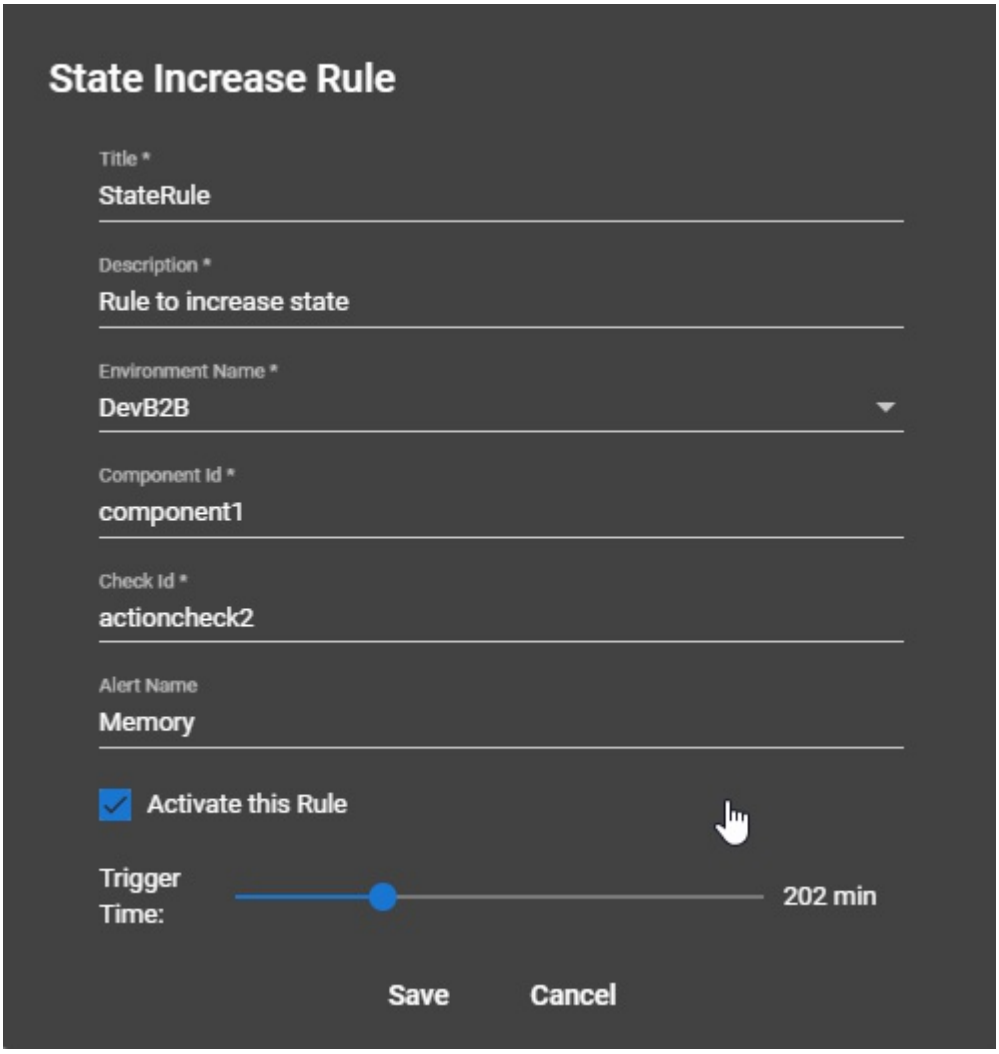
Add State Increase Rule:

Click on "Add" icon to add an new rule.



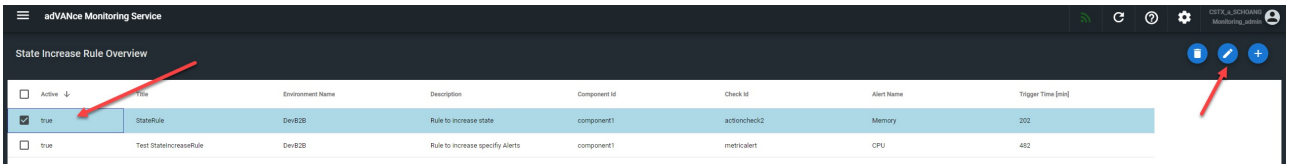
The following fields must be filled:

- Title
- Description
- Select the environment
- Select the Component Id
- Select the Check Id
- Set the Alert Name
- Set "Activate this Rule" (yes/no)
- Set Trigger time (from 1 - 720 minutes)

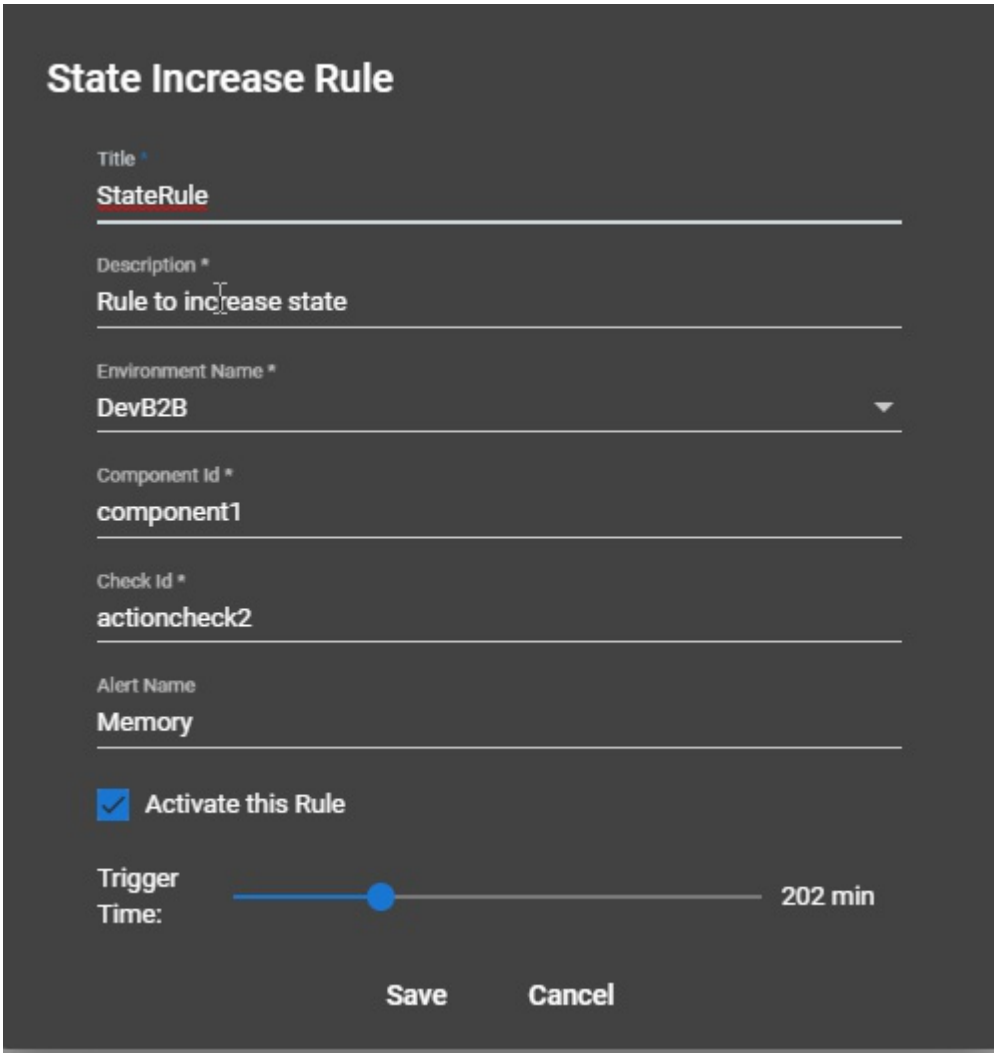


Modify State Increase Rule:

- Double-click on the rule **or** select the rule and click on the "Edit" icon

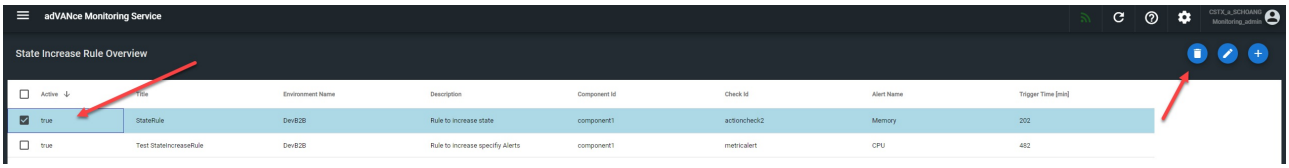


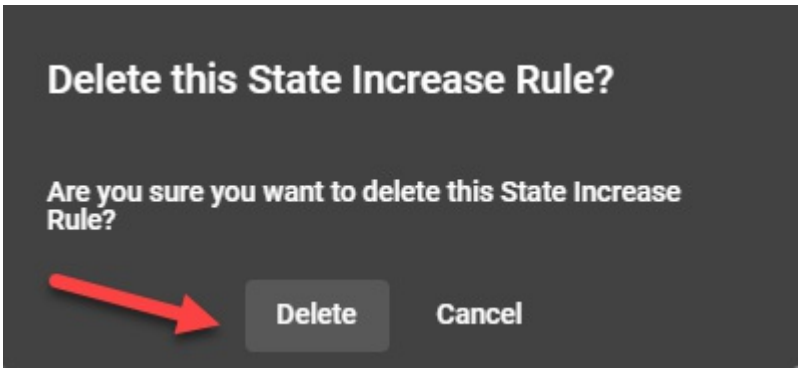
- Make changes and save to complete.



Delete State Increase Rule:

Select the rule and click on the "Delete" icon





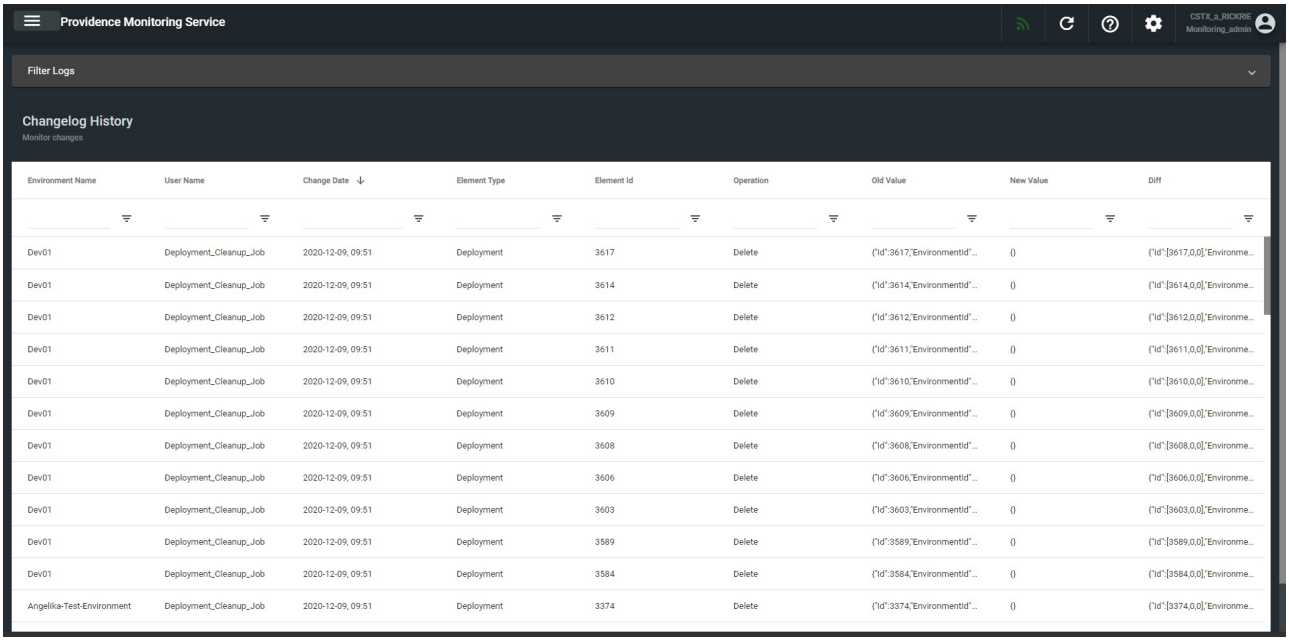
Changelog History The changelog history displays all created, updated or deleted new elements of the Providence service.

All Create/Update/Delete config should be visible in some kind of history backlog.

- If somebody changes an "State Increase Rule" we want to be able to track down who it was.
- If somebody removes a "Alert Ignore Rule" we need to know who did it.

All changes in the model of tree elements (Environment/Service/Action/Component) are exclude because these are changed frequently by the users.

Changelog History overview:



Filter Logs:
The filter can be used to limit the period:

- Set the start/end date or choose a date
- Set the start/end time

Internal Jobs

All internal jobs of the Providence Service are explained below.

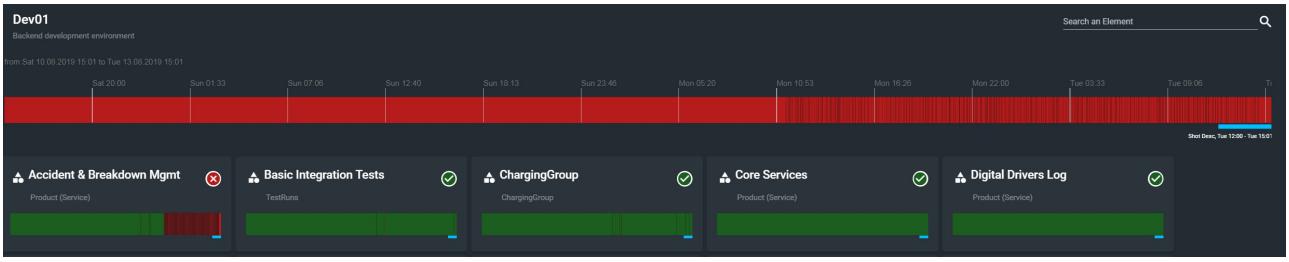
The Providence service includes the following internal jobs:

- Deployment Job
- State Increase Job
- Email notifications job
- Delete expired State transitions
- Auto refresh environments
- Reset to green

Deployment Job

This job notifies the UI when a new deployment which shall be shown in the timeline was created.
The UI then takes this new deployment and sorts it into the timeline so that it is visible as a blue bar under the timeline.

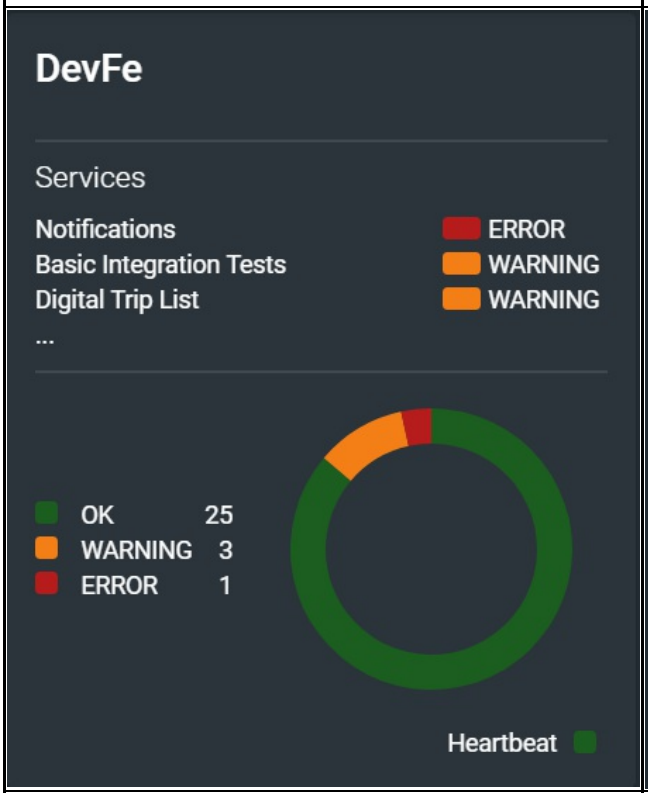
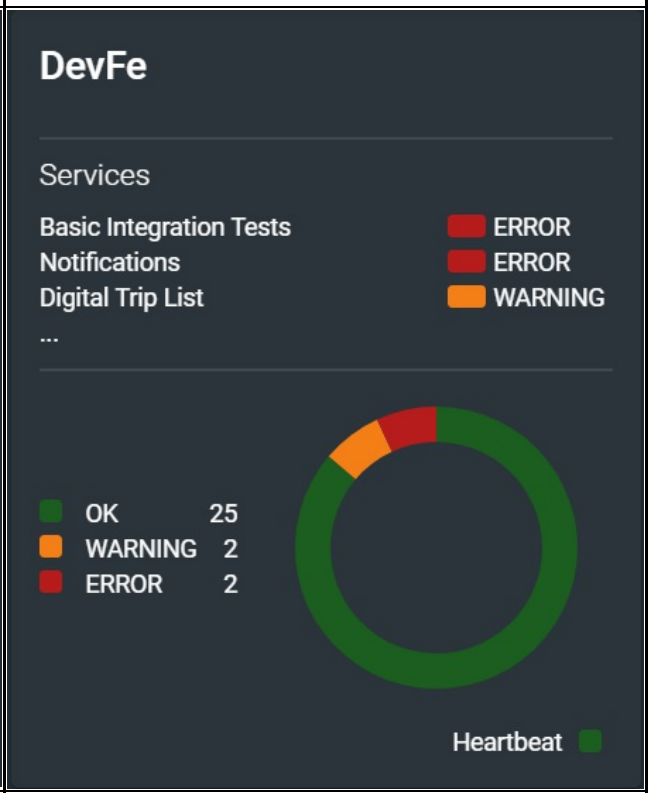
Screenshot of the timeline showing the deployment slot



State Increase Job

This job runs in a specified timeintervall and checks if there are elements with the state "Warning" which needs to be increased to the state "Error". The decission whether an element's state needs to be increased or not depends on the configured state increase rules of the system. If the criteria matches the job generates a new alert message to change the state of the element to "Error". In addition, all subsequent alerts that have the status "Warning" are automatically increased to alerts with the status "Error".

Screenshot with and without an active state increase rule.

Without active state increase rule	With active state increase rule
	

Email Notifications Job This job runs in a specified timeintervall and checks if the states of an element matches the configuration of any notification rules. If so the job notifies the users mentioned in the rule via an email that a specific event occured.

The emails contains the following information:

- Element (which element is affected)
    - Environment Name
    - Environment ID
    - Notification Level
    - Element ID
    - New State
    - URL to the Environment
- Alert (Which alert is responsible for the email)
    - State
    - Timestamp
    - Check ID
    - Affected component
    - Record ID
    - Alert Description

Delete expired Elements Job

This job is used to delete expired elements stored in the Providence database after a specific time period (minimum 1 week) in order to reduce the data sice within the state transition table and to increase the overall perfomance of the system.
The settings is environment specific and is configured within the app settings of the Providence service application.

Elements that are deleted by this job are:

- Deployments
- StateTransitions
- ChangeLog Entries
- Unassigned Components

Refresh environments Job

This job is used to clean up the state managers hold by the Providence backend. If necessary the job disposes and rebuilds the state managers within the backend. This is for example needed if changes were made directly on the database and need to be transfered into the state manager of a deployed instance.
The settings is environment specific and is configured within the app settings of the Providence service application.

Reset to Green Job

This job runs in a specified timeintervall and checks if there are elements with the state "Warning" or "Error" which needs to be reseted to the state "Ok". The decission whether an element needs to be resetet or not depends on the alerts which leed to the current state and on the "reset frequency" property of the check which belongs to the received alert. If the criteria matches the job generates a new alert message to change the state of the element to "Ok".

When creating a check the "reset frequency" can be added. The frequency needs to be in the timerange of 10 min to 1440 min.

API Documentation

This sub pages include documentation of following Providence APIs:

- Export and Import API
- Notification API
- Deployment API
- Alert Ignore API
- State Increase Rule API

Export and Import API

The Export/Import API of the Providence Service supports operations for exporting and importing whole environment trees.

API Documentation

`{{HOST}} = https://spp-monitoringservice-X.azurewebsites.net` where X stands for the environment the request is made to.

Export Environment

This endpoint is accessible via: **GET: {{HOST}}/api/masterdata/environmentUpdate?environmentSubscriptionId={environmentSubscriptionId}**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.OK** when the export was successful.
- **HttpStatusCode.BadRequest** when an invalid or no environmentSubscriptionId is provided.
- **HttpStatusCode.NotFound** when an unknown environmentSubscriptionId is provided.
- **HttpStatusCode.InternalServerError** when something went wrong during the export process.

Import Environment

This endpoint is accessible via: **POST: {{HOST}}/api/masterdata/environmentUpdate?instance_name={{environmentname}}&environmentSubscriptionId={{environmentsubscriptionid}}&replace={Replace}**

Parameters:

- **instance_name** is a placeholder variable. All occurences of {instance_name} will be replaced with the values of these variable.
- The **replace** parameter can have following values:
    - **True**: Hereby all given elements in the payload will be replaces in the according environment. (If they don't exist they will be created)
    - **False**: Hereby the elements in the environment will be supplemented with the elements in the payload.
    - **All**: Hereby the elements in the enviroment will be oberriden by the elements given in the payload.

The response of this endpoint call can be one of the following:

- **HttpStatusCode.OK** when the import was successful.
- **HttpStatusCode.BadRequest** when an invalid or no environmentSubscriptionId is provided.
- **HttpStatusCode.NotFound** when an unknown environmentSubscriptionId or environmentName is provided.
- **HttpStatusCode.InternalServerError** when something went wrong during the export process.

So for example it is possible to export the **dev** environment, replace in the response all appearances of "dev" with {environment_name}, and after that during the import again all occurences can be replaced by the right values with passing in the URL the value environment_name=test. With that mechanism all elements are imported with the correct envirnment name.

The payload of the import has following structure:

```
{
    "Services": [
        {
            "actions": [""],
            "elementId": "",
            "name": "",
            "description": ""
        }
    ],
    "Actions": [
        {
            "components": [""],
            "elementId": "",
            "name": "",
            "description": ""
        }
    ],
    "Components": [
        {
            "componentType": "",
            "elementId": "",
            "name": "",
            "description": ""
        },
        {
            "componentType": "",
            "elementId": "",
            "name": "",
            "description": ""
        }
    ],
    "Checks": [
        {
            "vstsLink": "",
            "frequency": -1,
            "elementId": "",
            "name": "",
            "description": ""
        }
    ]
}
```

Notification API

The Notification API of the Providence Service supports operations for creating, updating and deleting notification rules for specific users and element states.

API Documentation

`{{HOST}} = https://spp-monitoringservice-X.azurewebsites.net` where X stands for the environment the request is made to.

Create Notification Rule

This endpoint is accessible via: **POST: {{HOST}}/api/notificationRules**

Following payload is used to create a notification rule.

Notice:

- The **levels** list shows for which kind of elements you want to be notified (To avoid spam this always notifies you about the highest state of this list)
  So for example if a **Component** goes to state **ERROR** and therefore also an **Action** goes to **ERROR** you receive a notification for the higher element (the Action) only.
- The **states** list show for which kind of states you want to be notified
- the **notificationInterval** shows after which period of time (in seconds) you want to be notified about a state change

```
{
    "environmentSubscriptionId": "XYZ",
    "levels": ["Environment", "Service", "Action", "Component"],
    "emailAddresses": "NotifyMeUser@daimler.com",
    "states": ["Error","Warning"],
    "notificationInterval": 120,
    "isActive": true
}
```

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Created** when the notification rule was created successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the created notification rule or the environment with the specified "environmentSubscriptionId" was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the creation process.

Update Notification Rule

This endpoint is accesible via: **PUT: {{HOST}}/api/notificationRules/{id}**

Following payload is used to update an existing notification rules (it is the same as when creating a new notification rule)

```
{
    "environmentSubscriptionId": "XYZ",
    "levels": ["Environment", "Service", "Action", "Component"],
    "emailAddresses": "NotifyMeUser@daimler.com",
    "states": ["Error","Warning"],
    "notificationInterval": 120,
    "isActive": true
}
```

The response of this endpoint call can be one of the following:

- **HttpStatusCode.NoContent** when the notification rule was updated successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the created notification rule or the environment with the specified "environmentSubscriptionId" was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the update process.

Delete Notification Rule

This endpoint is accessible via: **DELETE: {{HOST}}/api/notificationRules/{id}**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Accepted** when the notification rule was deleted successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the notification rule to delete was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the deletion process.

Get All Notification Rules

This endpoint is accesible via: **GET: {{HOST}}/api/notificationRules**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Ok** when the searched notification rules were retrieved successfully.
- **HttpStatusCode.InternalServerError** when something went wrong during the get process.

Get Notification Rule by Id

This endpoint is accessible via: **GET: {{HOST}}/api/notificationRules/{id}**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Ok** when the searched notification rule were retrieved successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the searched notification rule was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the get process.

Notification Logic

After the API is clear this section should explain how the notification rules work.

There are 3 options how a user can configure a notification rule:

- User wants to be notified about **WARNING** and **ERROR** states [1]
- User wants only to be notified about **WARNING** states [2]
- User wants only to be notified about **ERROR** states [3]

Depending on the rule configuration following scenarios may occur:
NOTICE:

- **"Warning / Error occured"** notifications are only send if the state of the specified element lasts longer than the **notificationInterval** configured in the rule
- **"Warning / Error resolved"** notifications are only send if a previous Warning / Error notification was send (to avoid spam)
- **"Warning / Error resolved"** notifications are send immediately and do not depend on the **notificationInterval** of a rule

State Change: OK -> WARNING

- [1] User receives a **"Warning occured: ..."** notification
- [2] User receives a **"Warning occured: ..."** notification
- [3] User receives no notification

State Change: OK -> ERROR

- [1] User receives a **"ERROR occured: ..."** notification
- [2] User receives no notification
- [3] User receives a **"ERROR occured: ..."** notification

State Change: WARNING -> ERROR

- [1] User receives a **"ERROR occured: ..."** notification
- [2] User receives no notification
- [3] User receives a **"ERROR occured: ..."** notification

State Change: WARNING -> ERROR -> WARNING

- [1] User receives a **"ERROR resolved: ..."** notification
- [2] User receives no notification
- [3] User receives a **"ERROR resolved: ..."** notification

State Change: OK -> ERROR -> WARNING

- [1] User receives a **"ERROR resolved: ..."** notification

- [2] User receives a **"Warning occured: ..."** notification
- [3] User receives a **"ERROR resolved: ..."** notification

State Change: ERROR -> OK

- [1] User receives a **"ERROR resolved: ..."** notification
- [2] User receives no notification
- [3] User receives a **"ERROR resolved: ..."** notification

State Change: WARNING-> OK

- [1] User receives a **"WARNING resolved: ..."** notification
- [2] User receives a **"WARNING resolved: ..."** notification
- [3] User receives no notification

Deployment API

The Deployment API of the Providence Service supports operations for creating, updating and deleting deployment windows.
Using the "repeatInformation" property of a Deployment you can configure recurring deployments. On this way you for example can create a deployment widnow for each friday at 10PM.

API Documentation

`{{HOST}} = https://spp-monitoringservice-X.azurewebsites.net` where X stands for the environment the request is made to.

Create Deployment

This endpoint is accessible via: **POST: {{HOST}}/api/deployments**

Following payload is used to create a deployment.
Notice:

- A deployment for an environment can only be created if there isn't already an ongoing deployment.
- Also the property startDate is optional. If no value is provided or the provided value points to the future, the current date and time will be set as value.

```
{
    "environmentsubscriptionId": "",
    "description": "",
    "shortDescription": "",
    "startDate": "",   // StartDate of the Deployment Window
    "endDate" : "",    // EndDateDate of the Deployment Window
    "closeReason": "",
    "elementIds": ["",""],
    "repeatInformation":
    {
        "repeatType": 0,   // 0 = Daily, 1 = Weekly, 2 = Monthly
        "repeatInterval": 0, // Repeat every X days
        "repeatOnSameWeekDayCount": false,  // true if for example: Every 2nd Monday of a Month
        "repeatUntil": ""  // EndDateDate of the recurring Deployment Window
    }
}
```

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Created** when the deployment was created successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the created deployment was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the creation process.

Update Deployment

This endpoint is accessible via: **PUT: {{HOST}}/api/deployments/{environmentsubscriptionid}/{id}**

Following payload is used to update an existing deployment.

```
{
    "environmentsubscriptionId": "",
    "description": "",
    "shortDescription": "",
    "startDate": "",   // StartDate of the Deployment Window
    "endDate" : "",    // EndDateDate of the Deployment Window
    "closeReason": "",
    "elementIds": ["",""],
    "repeatInformation":
    {
        "repeatType": 0,   // 0 = Daily, 1 = Weekly, 2 = Monthly
        "repeatInterval": 0, // Repeat every X days
        "repeatOnSameWeekDayCount": false,  // true if for example: Every 2nd Monday of a Month
        "repeatUntil": ""  // EndDateDate of the recurring Deployment Window
    }
}
```

The response of this endpoint call can be one of the following:

- **HttpStatusCode.NoContent** when the deployment was updated successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the deployment to update was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the update process.

Delete Deployment

This endpoint is accessible via: **DELETE: {{HOST}}/api/deployments/{environmentsubscriptionid}/{id}**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Accepted** when the deployment was deleted successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the deployment to delete was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the deletion process.

Get All Deployments

This endpoint is accessible via: **GET: {{HOST}}/api/deployments**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Ok** when the searched deployment was retrieved successfully.
- **HttpStatusCode.InternalServerError** when something went wrong during the get process.

Get Deployment History for an Environment

This endpoint is accesible via: **GET: {{HOST}}/api/deployments/{environmentName}?startDate={startDate}&endDate={endDate}**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Ok** when the searched deployment were retrieved successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the searched deployment were not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the get process.

Alert Ignore API The Alert Ignore API of the Providence Service supports operations for creating, updating and deleting "Alert Ignore Rules".

Alert ignore rules can be used to mute specifc alerts. For example all alerts with a specific CheckId can be muted to avoid StateTransitions resulting from their occurrence in the Providence Service.

API Documentation

`{{HOST}} = https://spp-monitoringservice-X.azurewebsites.net` where X stands for the environment the request is made to.

Create Alert Ignore Rule

This endpoint is accessible via: **POST: {{HOST}}/api/alertIgnores**

Following payload is used to create an alert ignore rule.
Notice: The value of **environmentSubscriptionId** is mandatory and if **subscriptionId** is set in the **ignoreCondition** property those two values have to be identical.

```
{
    "name": "",
    "environmentSubscriptionId": "",
    "creationDate": "",
    "expirationDate": "",
    "ignoreCondition":
    {
       "checkId" : "",
       "componentId" : "",
       "subscriptionId" : "",
       "description" : "",
       "state": "",
       "customField1": "",
       "customField2": "",
       "customField3": "",
       "customField4": "",
       "customField5": "",
       "alertName": ""
    }
}
```

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Created** when the alert ignore was created successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the created alert ignore was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the creation process.

Update Alert Ignore Rule

This endpoint is accessible via: **PUT: {{HOST}}/api/alertIgnores/{id}**

Following payload is used to create an alert ignore rule.

```
{
    "name": "",
    "environmentSubscriptionId": "",
    "creationDate": "",
    "expirationDate": "",
    "ignoreCondition":
    {
       "checkId" : "",
       "componentId" : "",
       "description" : "",
       "state": "",
       "customField1": "",
       "customField2": "",
       "customField3": "",
       "customField4": "",
       "customField5": "",
       "alertName": ""
    }
}
```

The response of this endpoint call can be one of the following:

- **HttpStatusCode.NoContent** when the alert ignore was updated successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the alert ignore to update was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the update process.

Delete Alert Ignore Rule

This endpoint is accessible via: **DELETE: {{HOST}}/api/alertIgnores/{id}**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Accepted** when the alert ignore was deleted successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the alert ignore to delete was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the deletion process.

Get Alert Ignore Rule

This endpoint is accessible via: **GET: {{HOST}}/api/alertIgnores/{id}**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Ok** when the searched alert ignore was retrieved successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the searched alert ignore was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the get process.

Get Alert Ignore Rules

This endpoint is accessible via: **GET: {{HOST}}/api/alertIgnores**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Ok** when the searched alert ignore were retrieved successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the searched alert ignores were not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the get process.

State Increase Rule API The State Increase Rule API of the Providence Service supports operations for creating, updating and deleting specific rules.

State increase rules can be used to increase the state of specifc alerts after a specified period of time.

- For example a "warning" can be configuered to be processed as an "error" within the Providence Service if the state "warning" does not change after X minutes.

API Documentation

`{{HOST}} = https://spp-monitoringservice-X.azurewebsites.net where` X stands for the environment the request is made to.

Create State Increase Rule

This endpoint is accessible via: **POST: {{HOST}}/api/stateIncreaseRules**

Following payload is used to create a state increase rule.
Notice: Only the value of **alertName** is optional. All other values are **mandatory**

```
{
    "name": "",
    "description": "",
    "environmentSubscriptionId": "",
    "checkId": "",
    "alertName": "",
    "componentId": "",
    "triggerTime": int,
    "isActive": bool
}
```

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Created** when the State Increase Rule was created successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the specified environment was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the creation process.

Update State Increase Rule

This endpoint is accessible via: **PUT: {{HOST}}/api/stateIncreaseRules/{id}**

Following payload is used to update a state increase rule.

```
{
    "name": "",
    "description": "",
    "environmentSubscriptionId": "",
    "checkId": "",
    "alertName": "",
    "componentId": "",
    "triggerTime": int,
    "isActive": bool
}
```

The response of this endpoint call can be one of the following:

- **HttpStatusCode.NoContent** when the State Increase Rule was updated successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the State Increase Rule to update was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the update process.

Delete State Increase Rule

This endpoint is accessible via: **DELETE: {{HOST}}/api/stateIncreaseRules/{id}**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Accepted** when the State Increase Rule was deleted successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.

- **HttpStatusCode.NotFound** when the State Increase Rule to delete was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the deletion process.

Get State Increase Rule

This endpoint is accessible via: **GET: {{HOST}}/api/stateIncreaseRules/{id}**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Ok** when the searched State Increase Rule was retrieved successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the searched State Increase Rule was not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the get process.

Get State Increase Rules

This endpoint is accessible via: **GET: {{HOST}}/api/stateIncreaseRules**

The response of this endpoint call can be one of the following:

- **HttpStatusCode.Ok** when the searched State Increase Rule were retrieved successfully.
- **HttpStatusCode.BadRequest** when the payload/url contains invalid parameter.
- **HttpStatusCode.NotFound** when the searched State Increase Rules were not found in the database.
- **HttpStatusCode.InternalServerError** when something went wrong during the get process.

Postman Collection

A full Postman Collection for calling the Providence endpoint can be downloaded on this page.

Postman Collection:
Providence_Service.postman_collection.zip (./src/help/assets/help/assets/Providence_Service.postman_collection-04d5ca84-b9fe-4afe-92d7-a0c0bd31fa2b.zip)

Swagger UI

The Swagger documentation of the Providence Service can be found under following url:

- https://serviceurl/swagger