Realisation documents

Internship project

Jelle Van Langendonck and Amber Swevers

Table of Content

Reports

1. Description

In this document we will describe the technical implementation of the project. This will be done in 2 parts, firstly the front-end and then the back-end.

We've built an engine to generate and download reports as a PDF file. The engine is equipped with a drag and drop system. The client can drag components (e.g. graphs, charts, tables, text, etc.) and choose which information is shown. There are also pre-defined templates that can be generated with static content. The client also has the possibility to save the custom templates they've built and regenerate them later.

2. Front-end

Language/framework

The front-end framework used in this project is Angular. Angular is an application design framework and development platform to efficiently create single-page applications.

Components

This is a single page application built with many different components. Below is a tree that shows how the different components are nested and what they are used for.

- app.component
    - rps-template.component: loads the Resource Protection Status template
    - rbs-template.component: loads the Resource Backup Status template
    - rrs-template.component: loads the Resource Restore Status template
    - custom-template.component: contains the components to build a custom template and builds the PDF
        - custom-template-preview.component: shows a preview of the custom template the user has built
            - piechart.component: loads the different pie charts into the custom template
            - overview.component: loads the different tables into the custom template
            - linechart.component: loads the different line charts into the custom template
            - barchart.component: loads the different bar charts into the custom template

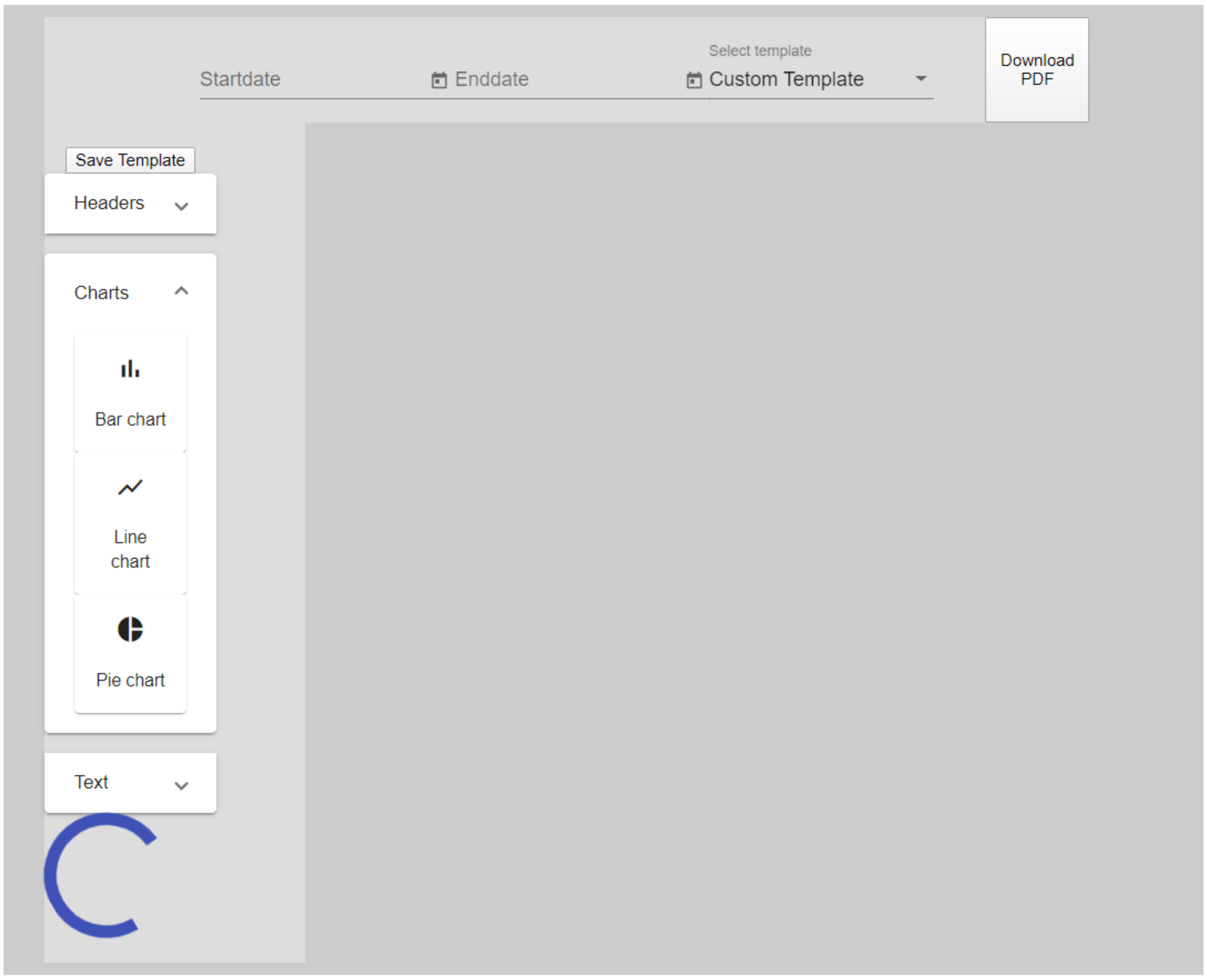
- header.component: loads the different headers into the custom template
- logo.component: loads the logo into the custom template

Getting started

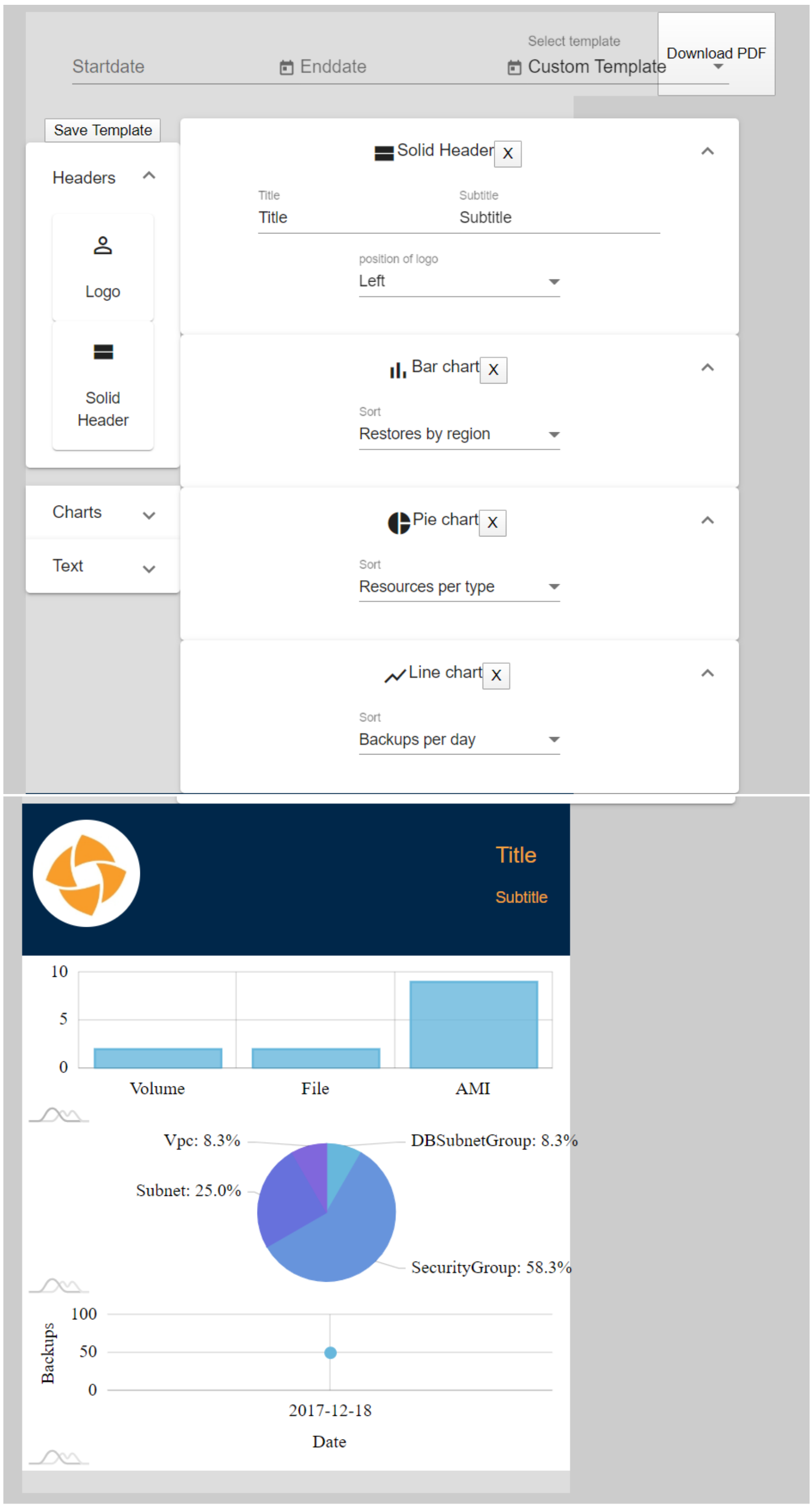
In order to use the application you must complete the following steps.

1. Open a terminal in the root of the project
2. Type ‘npm I’ to install all the dependencies
3. After the depencies have finished installing, type ‘ng serve’ to start the Angular app
4. Open a second terminal in the root of the project
5. Type ‘node server.js’ in the second terminal to run the backend of the application
6. Open your browser of choice and navigate to ‘lhttp://ocalhost:4200’. This will show you the page.
    a. If that port is currently in use, the Angular will run the application on another port. Check the terminal to determine which port is being used.

Screenshots



1. Screenshot of page with the Custom Template starting screen.



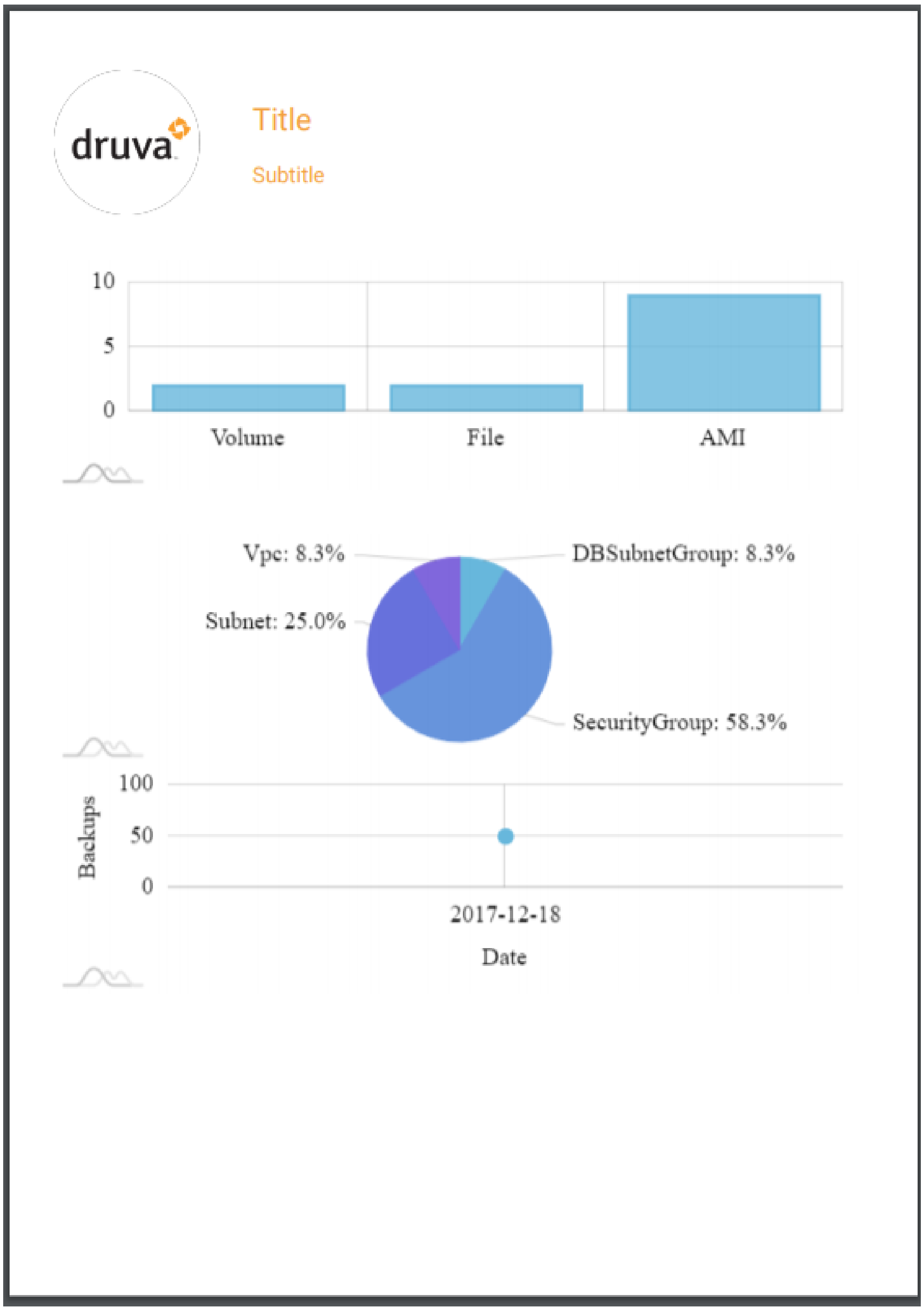
Title: Title
Subtitle: Subtitle
position of logo: Left

Sort: Restores by region

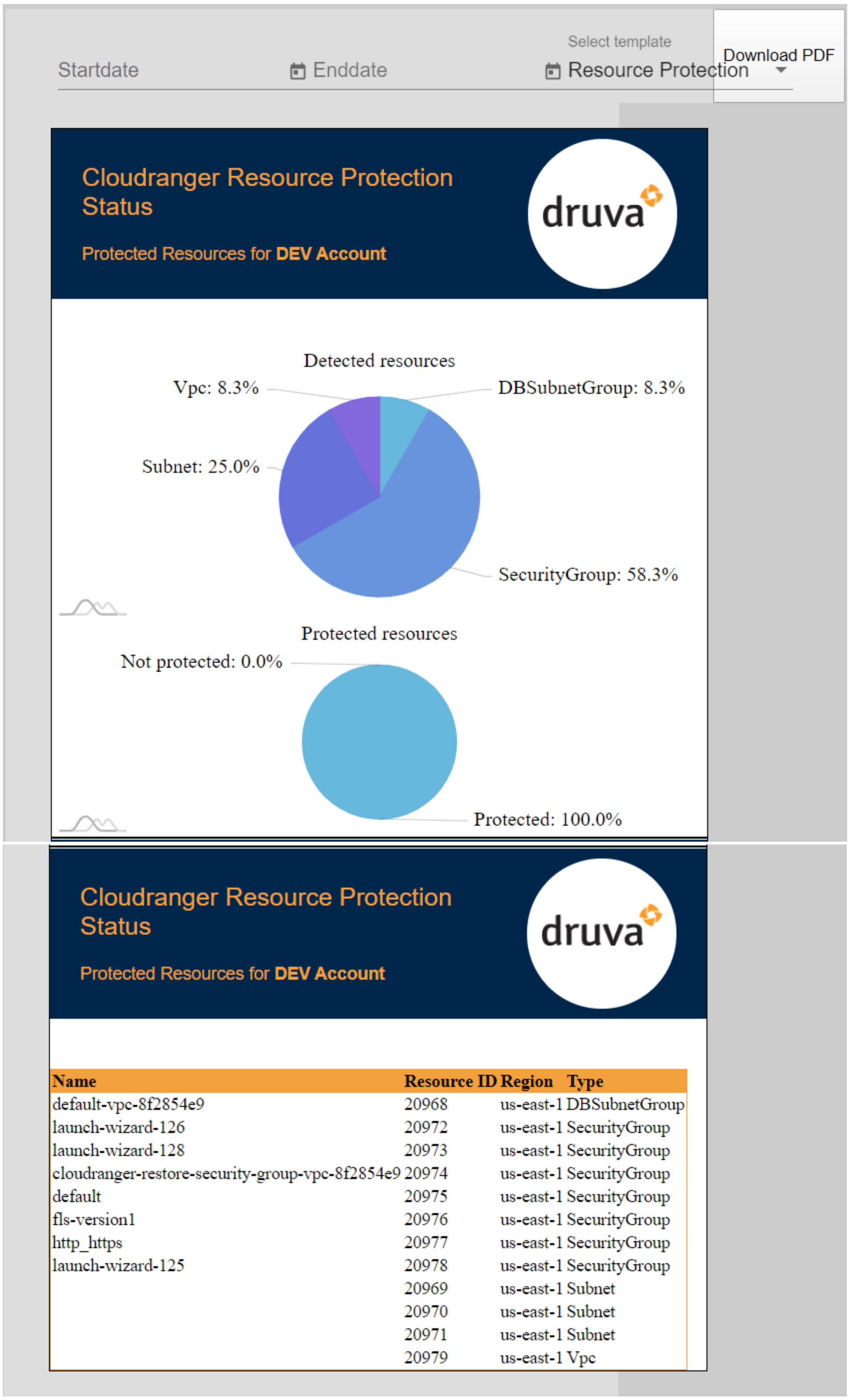
Sort: Resources per type

Sort: Backups per day

2. Starting page with components dragged in



3. The generated PDF



Cloudranger Resource Protection Status

Protected Resources for **DEV Account**

Cloudranger Resource Protection Status

Protected Resources for **DEV Account**

Name	Resource ID	Region	Type
default-vpc-8f2854e9	20968	us-east-1	DBSubnetGroup
launch-wizard-126	20972	us-east-1	SecurityGroup
launch-wizard-128	20973	us-east-1	SecurityGroup
cloudranger-restore-security-group-vpc-8f2854e9	20974	us-east-1	SecurityGroup
default	20975	us-east-1	SecurityGroup
fls-version1	20976	us-east-1	SecurityGroup
http_https	20977	us-east-1	SecurityGroup
launch-wizard-125	20978	us-east-1	SecurityGroup
	20969	us-east-1	Subnet
	20970	us-east-1	Subnet
	20971	us-east-1	Subnet
	20979	us-east-1	Vpc

4. Starting page with the pre-defined Resource Protection Status template

Dependencies

AMCharts

Used to create the charts and graphs.

AMCHARTS

pdfmake

Generates the PDF's.

pdfmake

Additional features

There are a few features that will be added in the future by Druva CloudRanger employees. These features were out of scope for our project.

Scheduling

The main feature that will be added after be conclude our internship is adding a scheduling option. With this feature, users will be able to auto-generate their reports daily, weekly, monthly, etc.

Datepickers

We have added datepickers to our project, but they are non-functional for the time being. This was not a priority for us since we are less familiar with the API than the CloudRanger engineers. The datepickers will be used to give users the possibilities to look at data from a specific period.

3. Back-end

Server

The backend will be hosted on AWS. AWS is a service that provides on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis.

Language/framework

The backend is written in Node.js. Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. The backend is solely used as a contact point between the API and the frontend and therefore can be made redundant by calling the API directly from the frontend.

Metabase dashboard

1. Description

In this document we will describe the technical implementation of the project. This will be done in 2 parts, firstly the components and then the dashboard.

2. Components

Language/framework

For this project we need to build a dashboard consisting of multiple components, these components are build directly in Metabase with simple SQL queries.

for this project we need to create these components:

-Account active check

-Organization subscribed check

-Amount of policies the organization has

-Amount of schedules the organization has

-Expiration date of subscription

-Amount of servers last 30 days

-Amount of accounts

-Amount of members

-Days since trial

-Kind of billing plan

-List of succesfull and failed jobs

-List of accounts

-Piechart account credential status (successfull/not configured)

-List of account credentials

-Piechart Jobtypes

-Piechart amount of jobs (succesfull/failed)

-Piechart Restoretypes

-List of Jobs

-List of failures

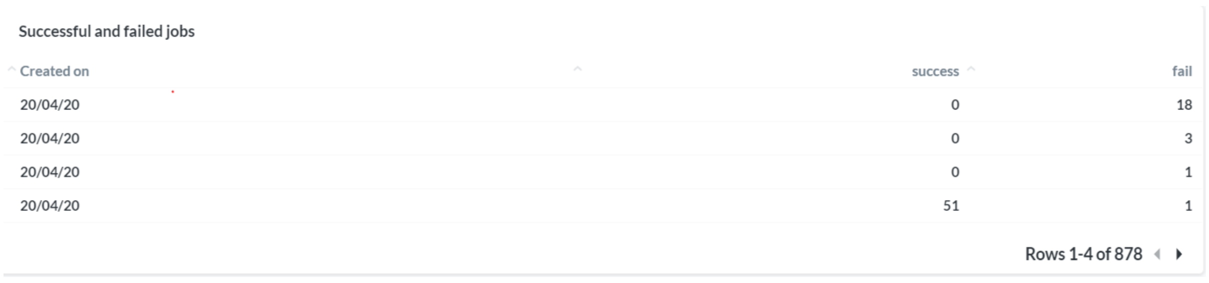
-Region map of amount of backups

-Piechart backups per region

-Linechart amount of servers during the last 30days

-Linechart amount of servers during the last 6 months

3. Examples

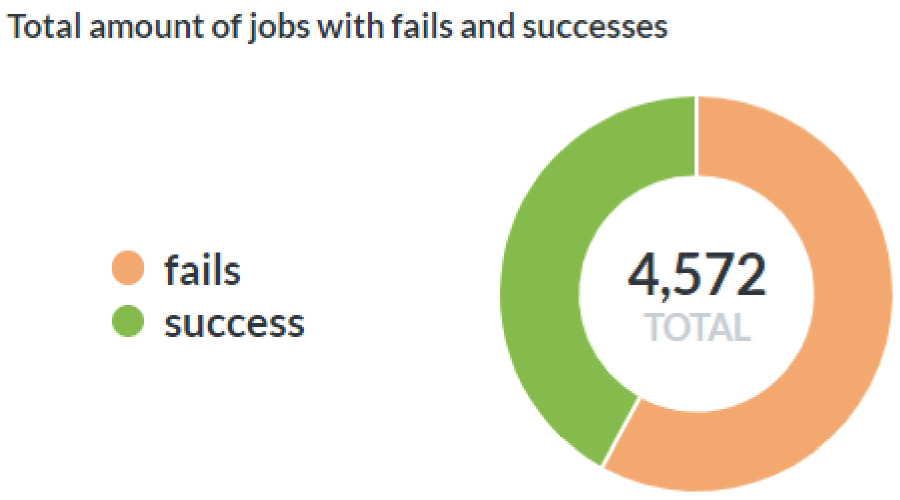


Successful and failed jobs

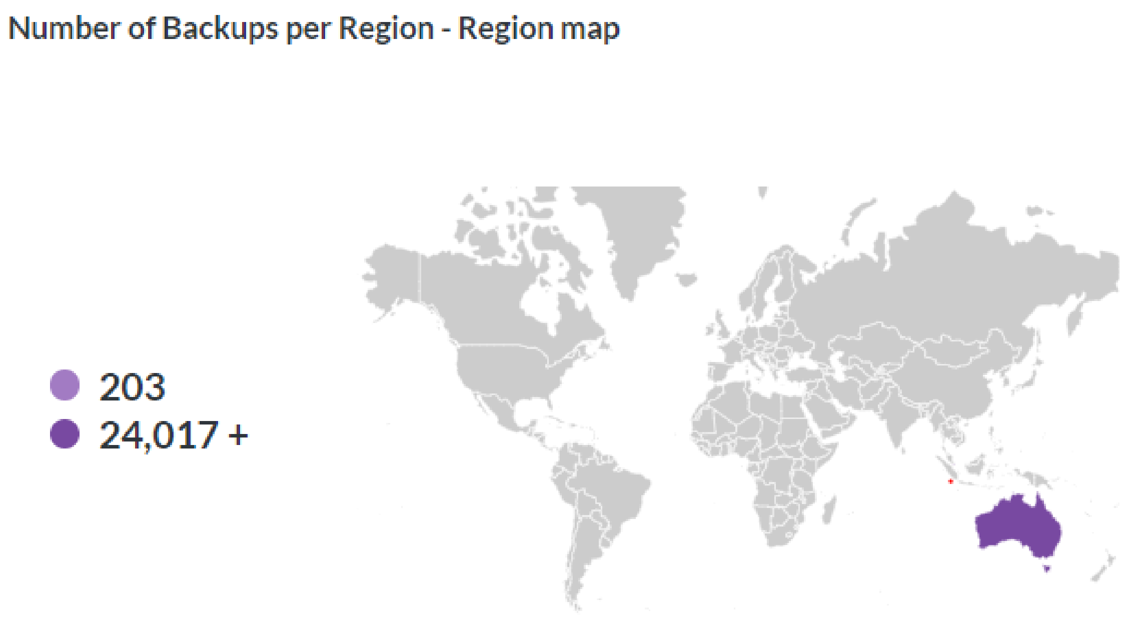
Created on	success	fail
20/04/20	0	18
20/04/20	0	3
20/04/20	0	1
20/04/20	51	1

Rows 1-4 of 878

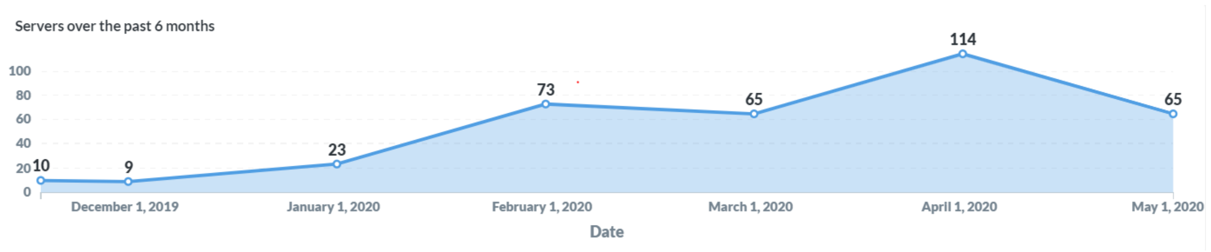
5 Component List Jobs (failed/successfull)



6 Piechart amount of jobs (failed/successfull)



7 Region map amount of backups



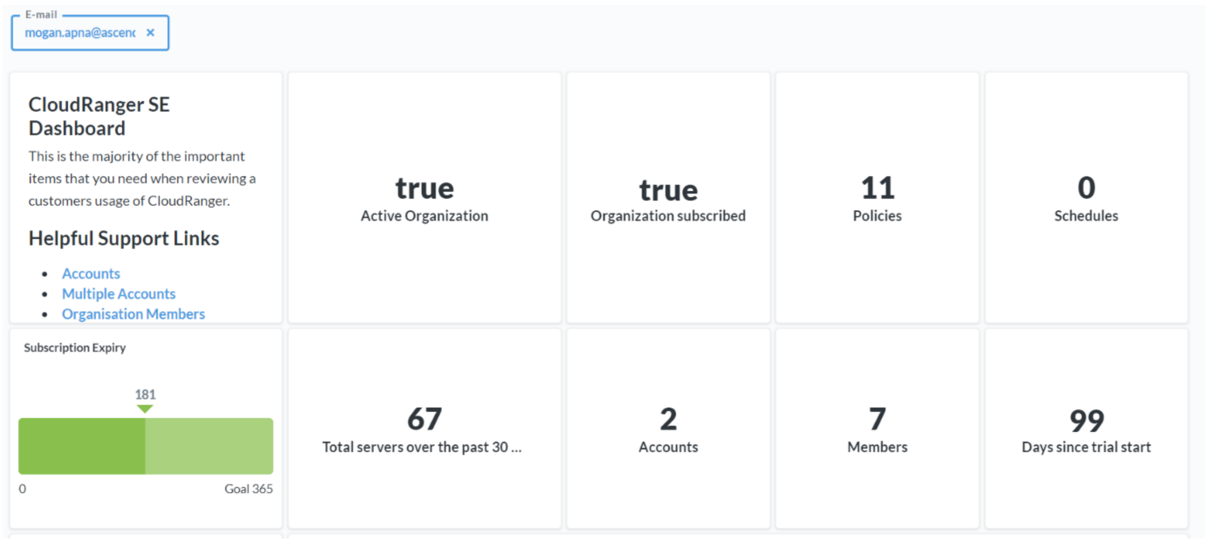
8 Linechart amount of servers during the last 6 months

4. Dashboard

Language/framework

The dashboard will also be build directly in Metabase with the drag and drop builder it provides. With this we can drag and drop the components we created and layout our dashboard. Also can we connect a mail input to the components.

Examples



E-mail
mogan.apna@ascenc ×

CloudRanger SE Dashboard

This is the majority of the important items that you need when reviewing a customers usage of CloudRanger.

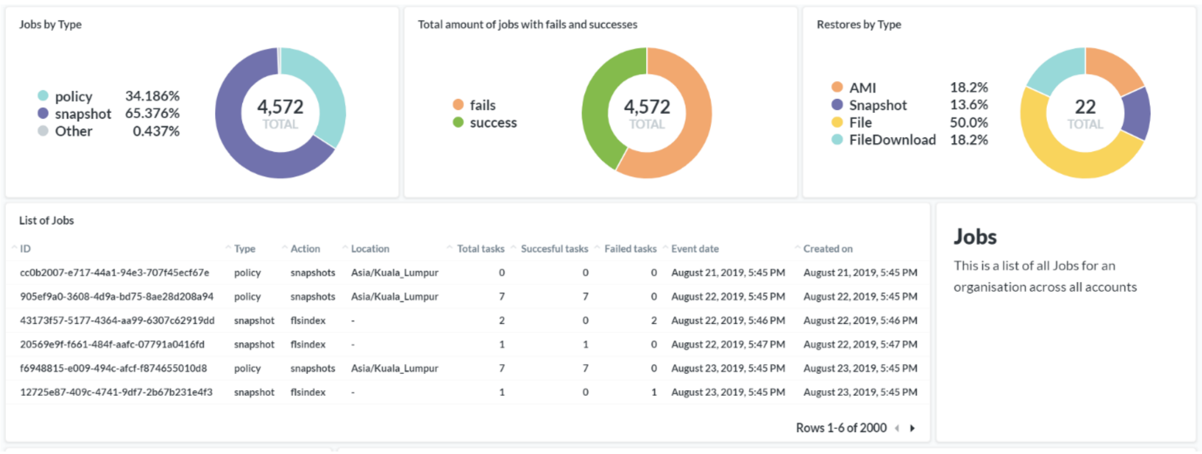
Helpful Support Links

- Accounts
- Multiple Accounts
- Organisation Members

true	Active Organization
true	Organization subscribed
11	Policies
0	Schedules
67	Total servers over the past 30 ...
2	Accounts
7	Members
99	Days since trial start

Subscription Expiry

9 Example of dashboard



Jobs by Type

policy	34.186%
snapshot	65.376%
Other	0.437%

Total amount of jobs with fails and successes

Restores by Type

AMI	18.2%
Snapshot	13.6%
File	50.0%
FileDownload	18.2%

List of Jobs

ID	Type	Action	Location	Total tasks	Succesful tasks	Failed tasks	Event date	Created on
cc0b2007-e717-44a1-94e3-707f45ecf67e	policy	snapshots	Asia/Kuala_Lumpur	0	0	0	August 21, 2019, 5:45 PM	August 21, 2019, 5:45 PM
905ef9a0-3608-4d9a-bd75-8ae28d208a94	policy	snapshots	Asia/Kuala_Lumpur	7	7	0	August 22, 2019, 5:45 PM	August 22, 2019, 5:45 PM
43173f57-5177-4364-aa99-6307c62919dd	snapshot	flsindex	-	2	0	2	August 22, 2019, 5:46 PM	August 22, 2019, 5:46 PM
20569e9f-f661-484f-aafc-07791a0416fd	snapshot	flsindex	-	1	1	0	August 22, 2019, 5:47 PM	August 22, 2019, 5:47 PM
f6948815-e009-494c-afcf-f874655010d8	policy	snapshots	Asia/Kuala_Lumpur	7	7	0	August 23, 2019, 5:45 PM	August 23, 2019, 5:45 PM
12725e87-409c-4741-9df7-2b67b231e4f3	snapshot	flsindex	-	1	0	1	August 23, 2019, 5:45 PM	August 23, 2019, 5:45 PM

Rows 1-6 of 2000

Jobs

This is a list of all Jobs for an organisation across all accounts

10 Example of dashboard

Snapshot TCO calculator

1. Description

In this document we will describe the technical implementation of the project. This will be done in 2 parts, firstly the front-end and then the back-end.

2. Front-end

Language/framework

The front-end used in this project is HTML5 with CSS3 and JavaScript, HTML5 is a mark-up language used for structuring and presenting content on the World Wide Web. While CSS3 is a style sheet language used for describing the presentation of a document written in a mark-up language like HTML5. And JavaScript is used to make the content generated by the mark-up language interactive.

Input

We need to be able to create a input for the back-end consisting of variables that it will use to calculate the different costs. This inputs needs to consist of at least:

-instanceCount: the amount of instances the client has running on Amazon.

-volumeSize": How large the instances are in Gigabyte.

-chosenRegion": On what region are these instances running.

-retentionDailySnapshots": How many daily snapshots the client wants to save.

-retentionWeeklySnapshots": How many weekly snapshots the client wants to save.

-retentionMonthlySnapshots": How many monthly snapshots the client wants to save.

-retentionYearlySnapshots": How many yearly snapshots the client wants to save.

-dailyChangeRate": How much the overall instance changes daily.

-weeklyChangeRate": How much the overall instance changes weekly.

-monthlyChangeRate": How much the overall instance changes monthly.

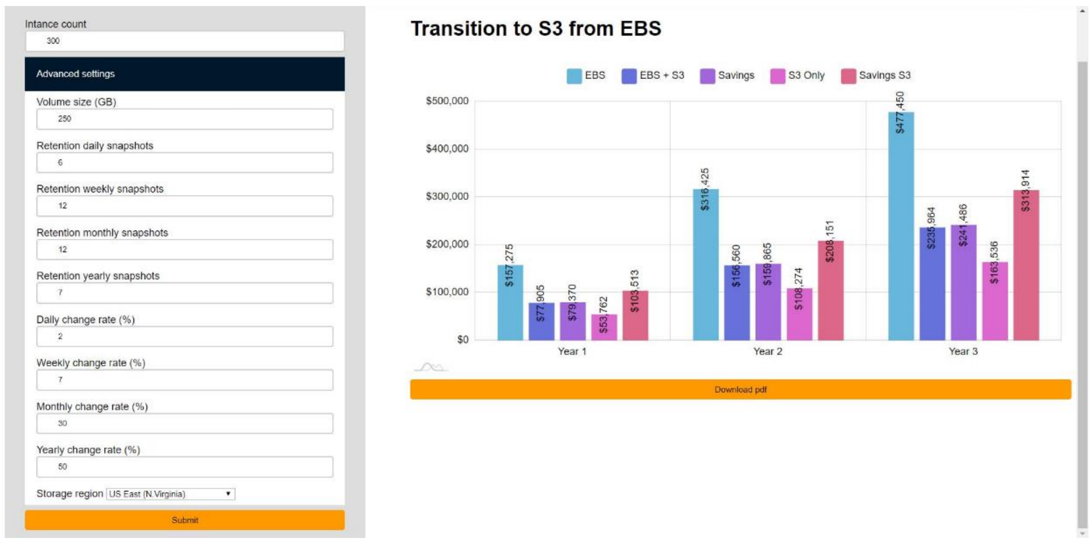
-yearlyChangeRate": How much the overall instance changes yearly.

All this data will be inserted by the user with a HTML form where after the user presses submit the frontend will compile all inserted data to JSON and proceed to send it to the backend hosted on AWS.

Output

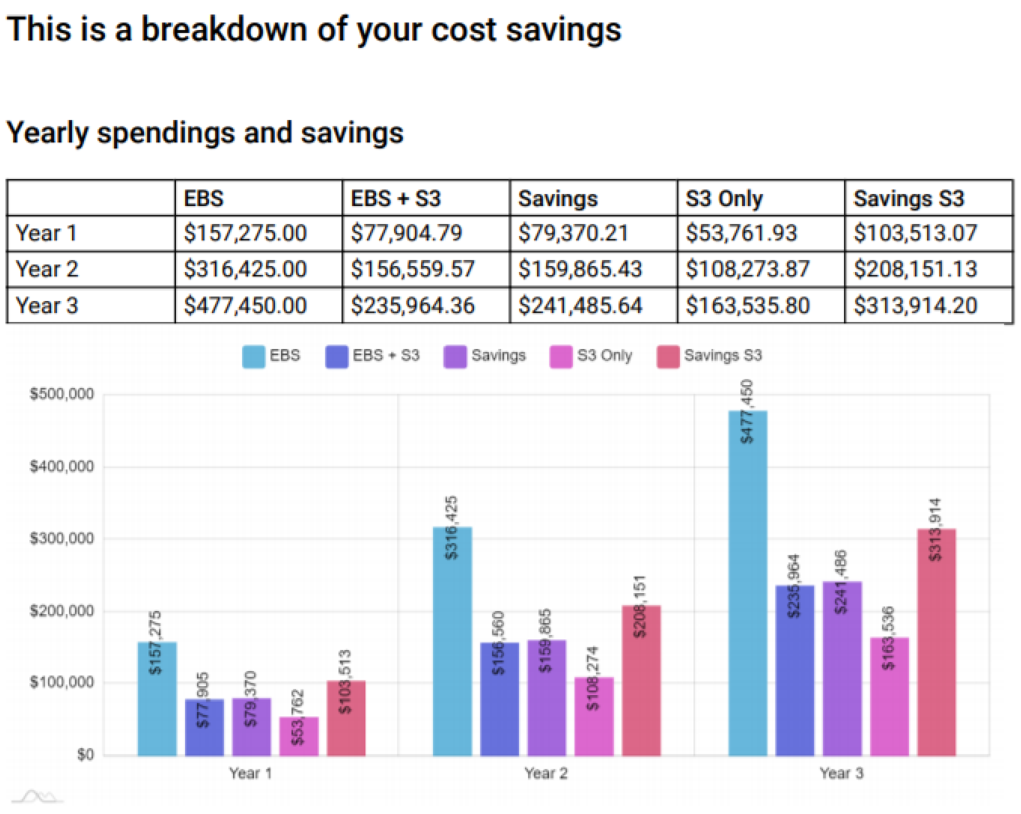
After the backend is done calculating it will return the calculations and with this data the frontend converts it to JSON and starts creating the charts with help of the AMCharts dependency. Afterwards it takes a screenshot of the chart and transform the received data and chart into a workable PDF that can be downloaded if necessary.

Examples



Intance count	300
Advanced settings	
Volume size (GB)	250
Retention daily snapshots	6
Retention weekly snapshots	12
Retention monthly snapshots	12
Retention yearly snapshots	7
Daily change rate (%)	2
Weekly change rate (%)	7
Monthly change rate (%)	30
Yearly change rate (%)	50
Storage region	US East (N.Virginia)

11. Screenshot of HTML form with corresponding output chart.



This is a breakdown of your cost savings

Yearly spendings and savings

	EBS	EBS + S3	Savings	S3 Only	Savings S3
Year 1	$157,275.00	$77,904.79	$79,370.21	$53,761.93	$103,513.07
Year 2	$316,425.00	$156,559.57	$159,865.43	$108,273.87	$208,151.13
Year 3	$477,450.00	$235,964.36	$241,485.64	$163,535.80	$313,914.20

12. Screenshot of generated PDF.

Dependencies

AMCharts

Used to create the charts and PDF.

3. Back-end

Server

The backend will be hosted on AWS. AWS is a service that provides on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis.

Language/framework

On AWS we will be hosting a small python script on a Flask server. Flask is a micro web framework written in Python that does not require particular tools or libraries.

Calculations

With the data received from the front-end, the script will calculate:

totalYearlyCostsEbs: The total yearly costs of all EBS snapshots.

dailyCostsEbs: The daily costs of all snapshots if only saved on EBS.

weeklyCostsEbs: The weekly costs of all snapshots if only saved on EBS.

monthlyCostsEbs: The monthly costs of all snapshots if only saved on EBS.

yearlyCostsEbs: The yearly costs of all snapshots if only saved on EBS.

totalYearlyCostsEbsS3: The total yearly costs of all EBS and S3 snapshots.

dailyCostsEbsS3: The daily costs of all snapshots if saved on EBS and S3.

weeklyCostsEbsS3: The weekly costs of all snapshots if saved on EBS and S3.

monthlyCostsS3: The monthly costs of all snapshots if saved on EBS and S3.

yearlyCostsS3: The yearly costs of all snapshots if saved on EBS and S3.

baseCostS3: The base cost of storing snapshots on S3

totalYearlyCostsS3: The total yearly costs of all S3 snapshots.

totalMonthlyCostsEbs: The total monthly of the first year costs if only saved on EBS.

totalMonthlyCostsEbsS3: The total monthly costs of the first year if saved on EBS and S3.

totalMonthlyCostsS3: The total monthly costs if only saved on S3.

processingCostsEbsS3: The total processing costs of transferring EBS snapshots to S3.

processingCostsS3: The total processing costs of transferring snapshots directly to S3.

baseCostEbs: The base cost of storing snapshots on EBS.

dailyCostsS3: The daily costs of all snapshots if saved on S3.

effectiveDailySnapshots: The effective amount of daily snapshots.

effectiveWeeklySnapshots: The effective amount of weekly snapshots.

effectiveMonthlySnapshots: The effective amount of monthly snapshots.

effectiveYearlySnapshots: The effective amount of yearly snapshots.

adjustedWeeklySnapshots: The adjusted amount of weekly snapshots.

adjustedMonthlySnapshots: The adjusted amount of monthly snapshots.

adjustedYearlySnapshots: The adjusted amount of yearly snapshots.