Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Diploma Thesis No. 3734

Extension of the Nefolog Decision Support System for the Cloud

Jinhui Huang

Course of Study: Informatik

Examiner:	Prof. Dr. Frank Leymann
Supervisor:	Dr. Vasilios Andrikopoulos, Santiago Gómez Sáez
Commenced:	01.12.2015
Completed:	01.06.2016

CR-Classification: D.2.1, D.2.9, H.3.3, H.5.2

Abstract

For the purpose to help users to make cloud migration choices more conveniently, this thesis extends a decision support system called Nefolog with better performance and more complex functionalities, which are exposed as RESTful web services. Based on these web services, a web application is created, during its implementation, a modern MVC framework AngularJS and a data visualization library D3 is used to produce a dynamic, interactive user interface.

Contents

List of Figures

List of Tables

List of Listings

1. Introduction

1.1. Motivation

As a part of her diploma thesis [Xiu13] [ARXL14], Xiu has developed a decision support system for helping to make decisions when we migrate the applications to the Cloud. As in Figure 1.1 shown, the JSP-based decision support system MiDSuS relies on a set of back-end RESTful Decision Support Web Services(Nefolog), these Web Services are generated by accessing the underlying Knowledge Base.

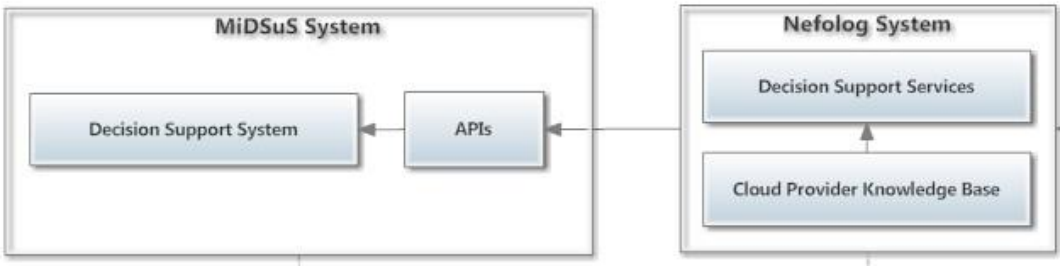


Figure 1.1.: Overview of Nefolog and MiDSuS [Xiu13]

The MiDSuS was designed for certain migration types [ABLS13], which are categorized by whether the application is partially or completely migrated to the Cloud. So the Nefolog are specified for these migration types, e.g. cost calculating, and they don't support more complex request such as finding cheapest configuration, i.e. when user has a certain configuration in mind, for example, she wants a Cloud virtual machine with certain number of CPU cores, certain amount of memory and so on, the Nefolog can only list all the possible offerings provided by different providers with the cost that may incur, this list can be very long, so its not easy for the users to choose the cheapest one. Furthermore, The User Interface of MiDSuS lacks also some functionalities, e.g. there is no way for the user to conveniently browse the offerings.

In this thesis, based on the existing Knowledge Base, we extend the Nefolog Decision Support System in both back-end and front-end, so that the system can provide more complex Web Services and also more convenient to use.

1.2. Outline

This thesis is structured in following chapters.

- **Chapter 2: Background & Related Work** introduces relevant fundamentals of this diploma thesis: the Nefolog system. This chapter also gives a brief State-of-the-Art about the UI provided by the main Cloud providers.
- **Chapter 3: Design** summarizes the requirements for both the new complex Web Services and UI, and the use cases build from these requirements. It also discusses the design of the new artifacts developed in this work.
- **Chapter 4: Implementation & Evaluation** introduces the technologies used during the implementation, and describes the implementation in detail. a brief evaluation is given in the end of this chapter.
- **Chapter 5: Conclusion** summarizes the entire thesis, lists the limitations of the work and some improvement possibilities.

2. Background & Related Work

2.1. Background

2.1.1. Fundamentals

Cloud computing meets the demands that people always want from IT: it is a way to increase capacity or add capabilities on the fly, and more important, without investing in new infrastructure, training new personnel, or licensing new software. According to the widely accepted definition of National Institute of Standards and Technology(NIST), it is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction[MG11]. The rest of this section will introduce five essential characteristics, three service models, and four deployment models that are proposed in the NIST cloud model.

Essential characteristics:

1. *On-demand self-service.* When needed, a consumer consumes the Cloud provider's computing capabilities, such as server time and network storage, automatically without human interaction.

2. *Broad network access.* Computing capabilities are available via the network and can be accessed using standard mechanisms, so that both thin and thick client platforms can access the capabilities.

3. *Resource pooling.* Computing resources of provider are pooled using a multi-tenant model in order to serve multiple consumers. Generally, the consumer doesn't have control and knowledge over the exact location of provided resources but may specify location at a higher level of abstraction.

4. *Rapid elasticity.* To the consumer, computing and storage resources appear to be unlimited, for they can be elastically provisioned and released at any time(in some cases automatically).

5. *Measured service.* The usage of resources is controlled and optimized automatically, furthermore, it can be monitored, controlled, and reported, so that transparency is provided both for the provider and consumer of the resources.

Service Models: NIST's definition proposed three service models, they define different access control level that the consumer has on the computer resources provided by a provider.

1. *Software as a Service (SaaS).* Consumer only has access of the provider's applications running on the cloud infrastructure. For all the three models, the underlying cloud infrastructure including network, servers, operating systems, storage can not be managed or controlled by consumer.
2. *Platform as a Service (PaaS).* Consumer can deploy consumer-created or acquired applications onto the cloud infrastructure. The consumer has control over the deployed applications and possibly configurations for the application-hosting environment.
3. *Infrastructure as a Service (IaaS).* Consumer has the access to provision processing, storage, networks, and other fundamental computing resources of the provider, the consumer can deploy and run arbitrary software, which can include operating systems and applications.

Deployment models: describes how the cloud infrastructure can be deployed.

1. *Private cloud.* The cloud infrastructure is provisioned exclusively by a single organization, the organization may own, manage and operate the infrastructure.
2. *Community cloud.* The cloud infrastructure is shared by a specific community of consumers that have same concerns.
3. *Public cloud.* The cloud infrastructure is provisioned for open use by the public.
4. *Hybrid cloud.* It combines two or more distinct cloud infrastructures(private, community and public), they remain unique entities, but are bound together by technology that enables data and application portability.

2.1.2. Nefolog

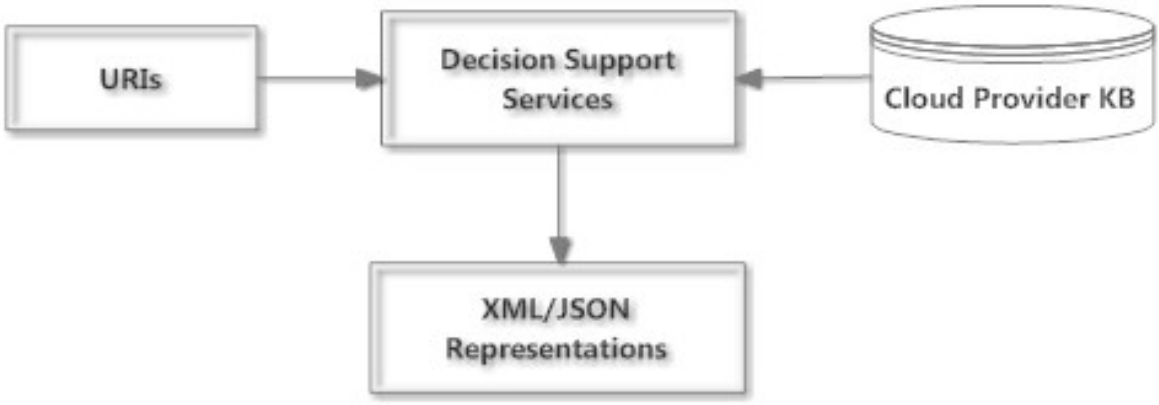


Figure 2.1.: Architecture of Nefolog [ARXL14]

The Nefolog Decision Support System is a significant extension of MDSS [Son13] [ASL13], Nefolog not only supports more Cloud providers(6 in Nefolog to 2 in MDSS) in the Cloud Provider Knowledge Base(Figure 2.2), but also provides a set of more general APIs, which are

implemented as a set of RESTful Web Services, the architecture of Nefolog is shown in the Figure 2.1. All the Web Services are exposed as a set of URIs, users can access the Knowledge Base through these URIs, the result can be in the form of JSON or XML.

Knowledge Base

The Data Model of Nefolog's Knowledge Base is shown in Figure 2.2. The basic information of all the offerings like Amazon EC2 [1] are abstracted into table `performance` (right top), whose columns contains all the possible performance characteristics. Table `configuration`(middle) connects all these basic informations like `performance`, `provider`, `offering`, `servicetype`.

Table `cost`, along with its relation with other tables like `coefficient`, `variable`, `location`, `zone` and so on, plays the most important role when serving the costCalculator Web Service. For a given performance, e.g. a virtual machine with 8 CPU cores, 1.2GHz CPU speed, 10 Gb memory and Windows operating system, the pricing policy from different Cloud providers are different, it differs from how and where the user want to deploy this virtual machine, along with the discount that the providers offer. The formulas used to calculate a certain cost for a given configuration with a certain performance, a given location and given usage information are stored in table coefficient. The cost table consists of 21856 rows which are combinations of 3686 upfront costs, 185 data transfer cost formulas, and 4733 service cost formulas.

Web Services

The Web Services in Nefolog are implemented by using Restlet [2] framework. The Web Services can be divided into two types, one is for simply accessing the content of the Knowledge Base, for example:

- ~/serviceTypes [3] will return all the service types declared in Knowledge Base, i.e. in table `servicetype`. The returned information contains the URI that leads to the Web Service for viewing each certain service type.
- ~/serviceTypes/{serviceTypeName}s will return a certain service type that the user demands. The returned information contains the URI that leads to the Web Service for viewing the provider that provides the service type.
- ~/offerings/{offeringName} returns all the configuration that belong to the offering user inputted.
- ~/offerings/{offeringName}/configuration_{configId} returns the performance information of the configuration that the user demands.

[1] https://aws.amazon.com/ec2/
[2] https://restlet.com/
[3] ~ means the absolute path of the deployed Web Services, e.g. localhost:8080/nefolog-webapp

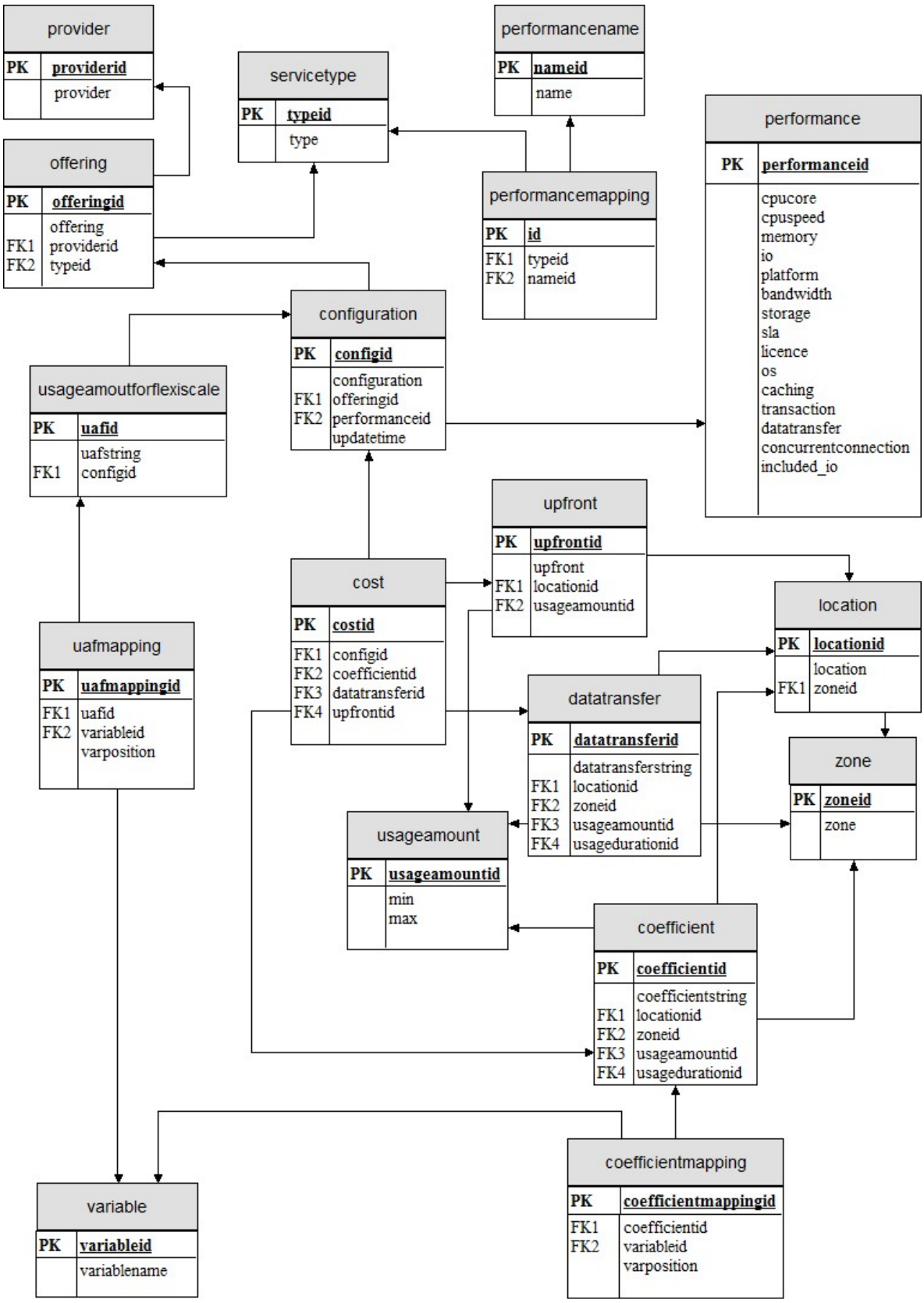


Figure 2.2.: Data Model of Nefolog's Knowledge Base [ARXL14]

This kind of Web Service are implemented by accessing table `configuration`, `performance`, `provider`, `offering`, `servicetype`.

The other type of Web Services are called decision support services [Xiu13], for example:

- ~/candidateSearch returns all the performance names that are stored in table `performance` with three more string "servicetype", "offering" and "provider".
- ~/candidateSearch?{query} will return candidate configurations which satisfy the user demands. For example, users can input the following URL: ~/candidateSearch?servicetype=infrastructure&cpuCores=8&cpuSpeed=1.2&memory=10&os=Windows to indicate the performance the user want, part of the result in form of XML is listed in the following Listing.

```
...
  <configuration>
    <name>
      30GB(Windows)
    </name>
    <uri>
      /offerings/cloudServers/configuration_487
    </uri>
  </configuration>

  <configuration>
    <name>
      m2.2xlarge Light Utillization High-Memory On-Demand
   Instances

  ...

```

Listing 2.1: Example result for url ~/candidateSearch?{query} [ARXL14]

- ~/costCalculator?{query} returns all the cost information that the user demands, for example, ~/costCalculator?configid=316&Hour=240&GBStorage=500&usage_pattern=(Hour,start=1,end=12,rate=10) has the following result:

```
...
<querycollection>
  <staticquery>Hour=240</staticquery>
  <staticquery>configid=316</staticquery>
  <staticquery>GBExternalNetworkEgress=5000</staticquery>
  ...
</querycollection>
<result>
  <location_zone>Northern Virginia</location_zone>
  <cost>
    $5258.11
```

```
12      <upfront>$1450.0</upfront>
13      <service>
14        $3808.11
15        <Month_Service>1st Month=$178.08</Month_Service>
16        <Month_Service>2nd Month=$195.89</Month_Service>
17
18        ...
19
```

Listing 2.2: Example result for url ~/costCalculator?{query} [ARXL14]

2.2. Cloud Providers

This section surveys the User Interface provided by the leading Cloud Computing Service providers like Amazon Web Services(AWS) [awsb], Windows Azure [mica], Google Cloud Platform [gooa]. The focus is on how they let the users browse their products and how their cost calculator look like.

2.2.1. Amazon Web Services

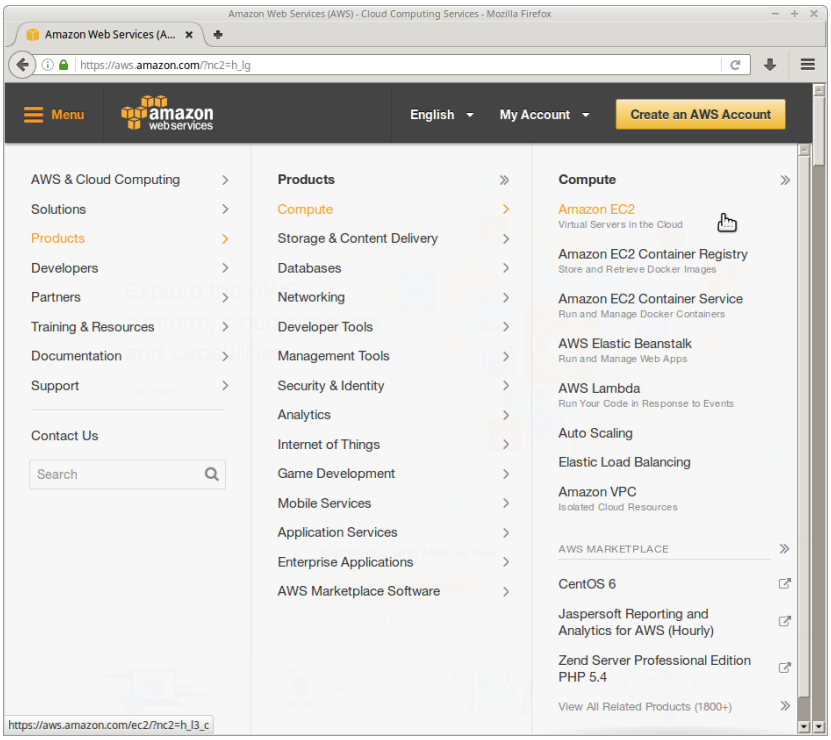


Figure 2.3.: Amazon Web Services products menu view [awsb]

As in Figure 2.3 shows, The HTML header element of the AWS's web page contains a drop down menu bar which can help the users to quickly navigate all the products and services that the AWS provides, for example, the users can navigate the product Amazon EC2 through the path: Products -> Compute -> Amazon EC2, this looks also like a collapsable tree structure.

AWS also provides matrix view for their products, right at their home page, or any time when the users click the Products link in the header element, the users can view the products as the following Figure 2.4 shows, The products are categorized, and displayed grouped by row and column header fields.

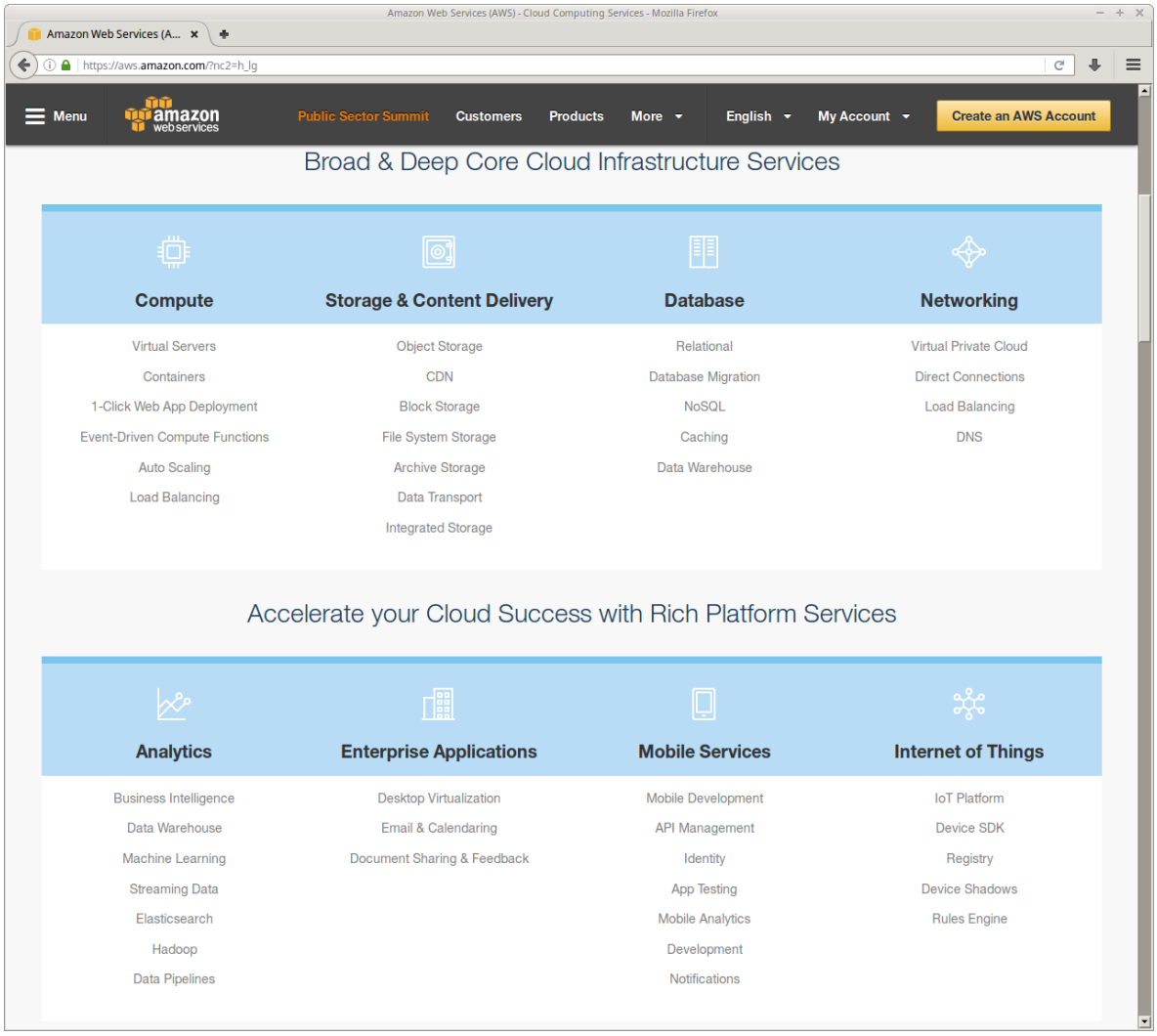


Figure 2.4.: Amazon Web Services products matrix view [awsc]

After the users browsed the products they are interested in, they can choose the individual product, view the product information and the pricing information. Then the users can get the monthly cost for the product by using the cost calculator that AWS provides. As shown in Figure 2.5, the users need to choose the product they want, e.g. Amazon EC2, in the left

navigation bar, then choose the region that the instance will be deployed, the instance type, for example t1.micro with Linux operating system, the instance number and the time usage amount in a month.

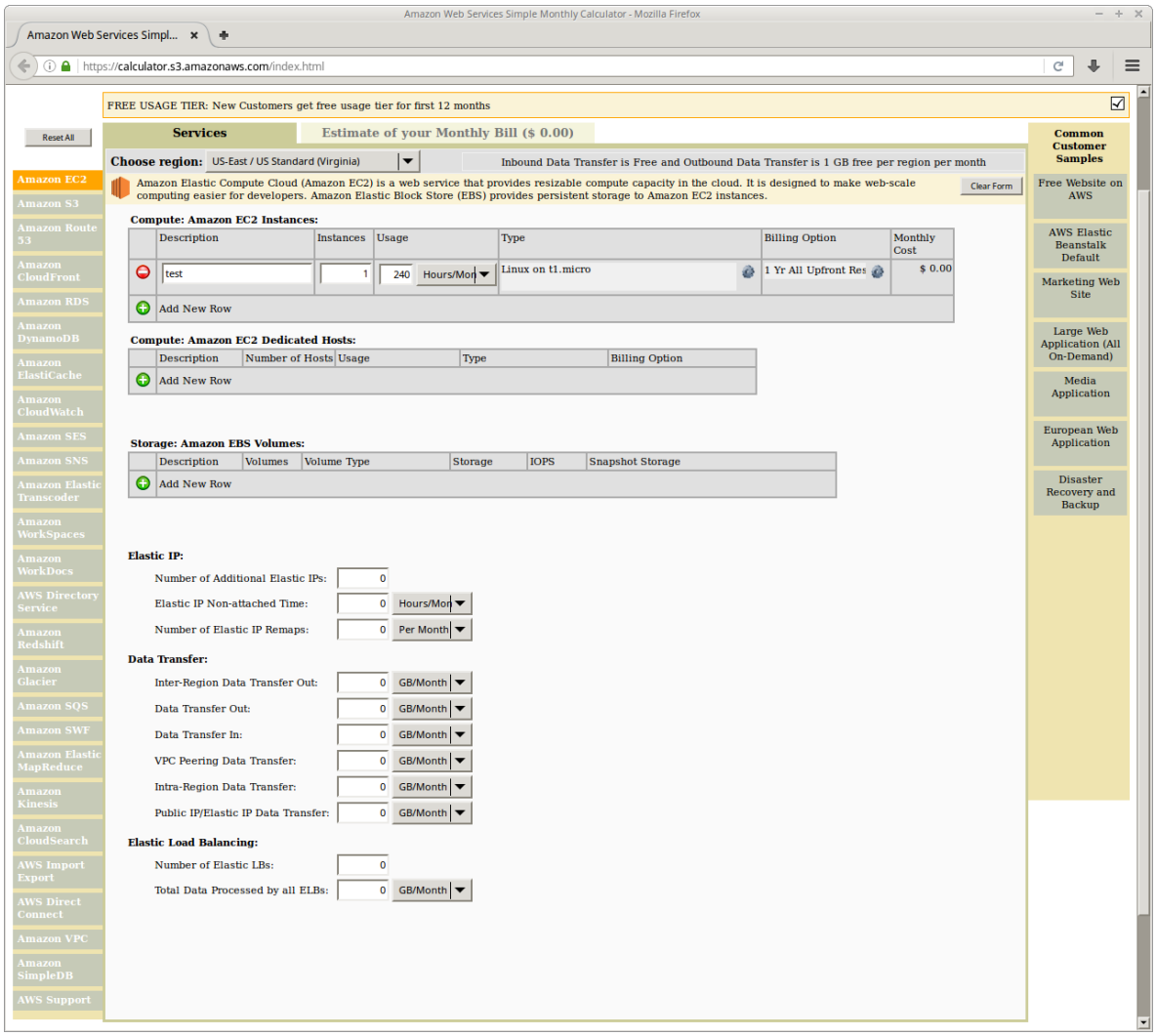


Figure 2.5.: Amazon Web Services Simple Monthly Calculator [awsa]

The users can also indicate more options that during the utilization of the instance they chose, for example the storage amount, the data transfer amount, and network related options like elastic IP and elastic load balancing.

AWS also provides another cost calculator called AWS Total Cost of Ownership (TCO) Calculator (as shown in Figure 2.6), which helps the users to compare the estimated cost when using AWS and on-premises hardware/software, The result is generated in form as a report, which contains a lot of diagrams and tables as in Figure 2.7, which can help the users to get more intuitive impression about the result.

Figure 2.7 shows a bar chart to show the difference, in the rest of the report, there are more

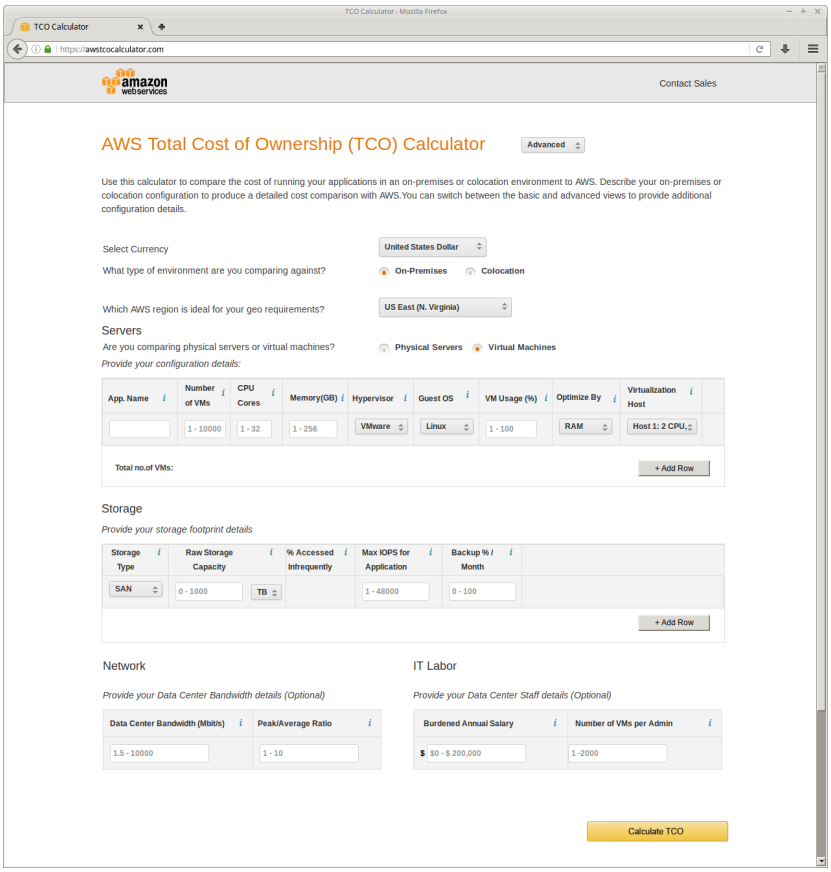


Figure 2.6.: Amazon Web Services TCO Calculator [awsd]

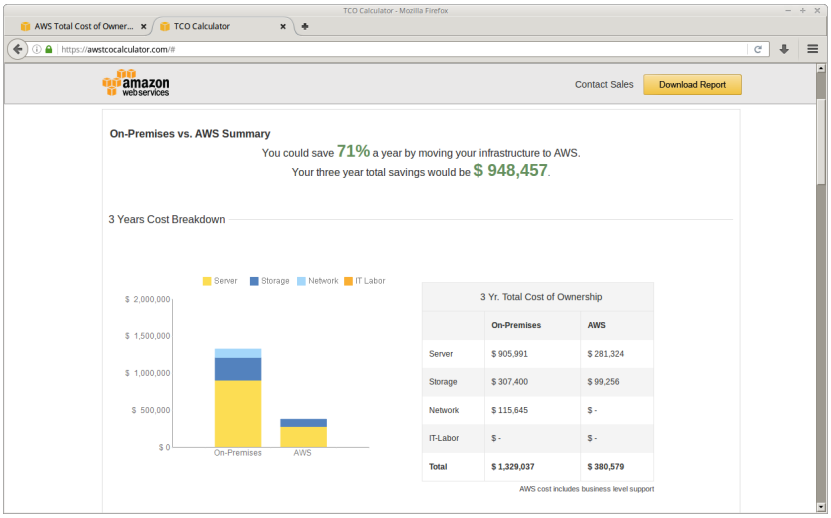


Figure 2.7.: Amazon Web Services TCO Calculator Report [awsd]

pie charts that illustrate the numerical proportion of different costs.

2.2.2. Microsoft Azure

Microsoft Azure also provides a drop down menu navigator in their web page's header element, as in Figure 2.8 shown.

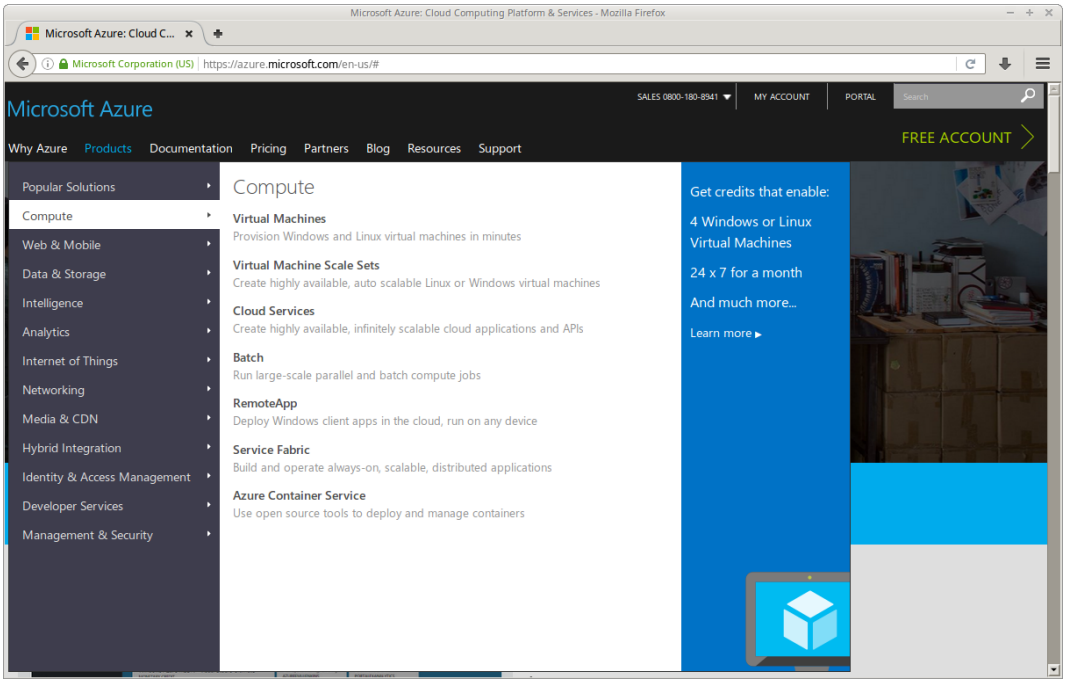


Figure 2.8.: Microsoft Azure products menu view [mica]

Unlike AWS, Azure only provides a products matrix view on their home page, which is not categorized, but listed by the products' popularity. But the pricing information for each product is shown in categorized matrix view [4].

The cost calculator provided by Azure is the most easy-to-use one in the 3 providers that are surveyed in this section. As in Figure 2.9 shown, the users can navigate to the product they want in the top half of the page(part 1.), after the users clicked one of the product, e.g. Virtual Machines, a new div element [5] will be dynamically added to the page(part 2.), the users can configure the detail of the instance they chose, and the users can repeatedly add and remove more product dynamically. The users can also configure support options(part 3.), the final result of the cost will be shown in part 4.

[4] https://azure.microsoft.com/en-us/pricing/
[5] https://www.w3.org/TR/html5/grouping-content.html#the-div-element

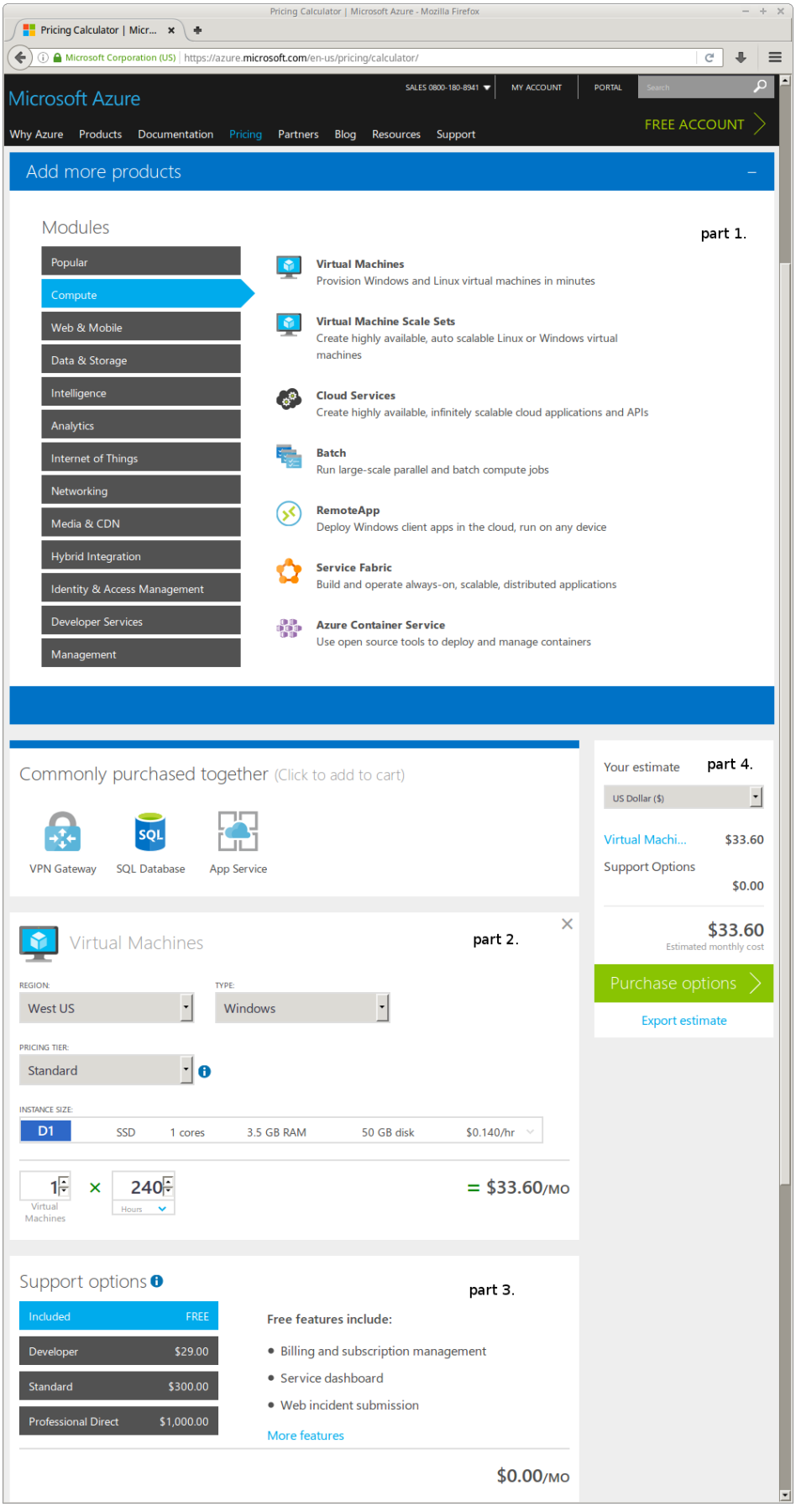


Figure 2.9.: Microsoft Azure products menu view [micb]

2.2.3. Google Cloud Platform

Google Cloud Platform provides a categorized products matrix view on their home page, as in Figure 2.10 shown.

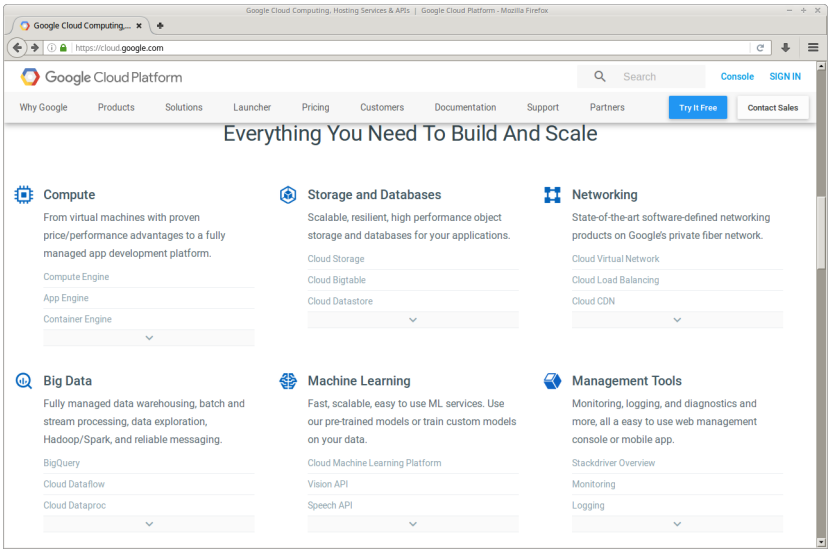


Figure 2.10.: Google Cloud Platform products matrix view [gooa]

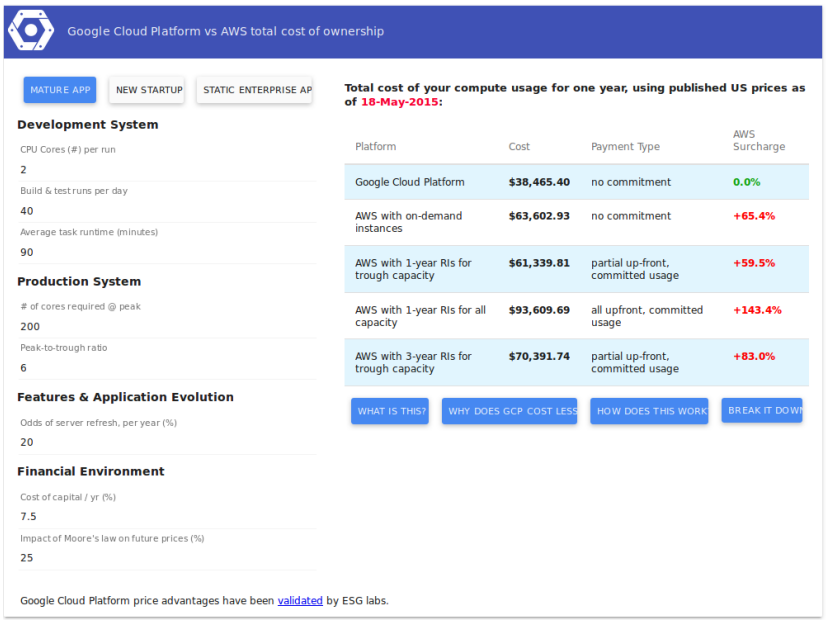


Figure 2.11.: Google Cloud Platform TCO Pricing Calculator [goob]

Google Cloud Platform also provides a TCO calculator (Figure 2.11) to compare the estimate cost with AWS, because it is difficult to correctly anticipate which instance type, size, location, and software configuration will be used by the user, the parameters (left side) here are

abstracted performance like how many CPU cores will be needed when the traffic hit the peak.

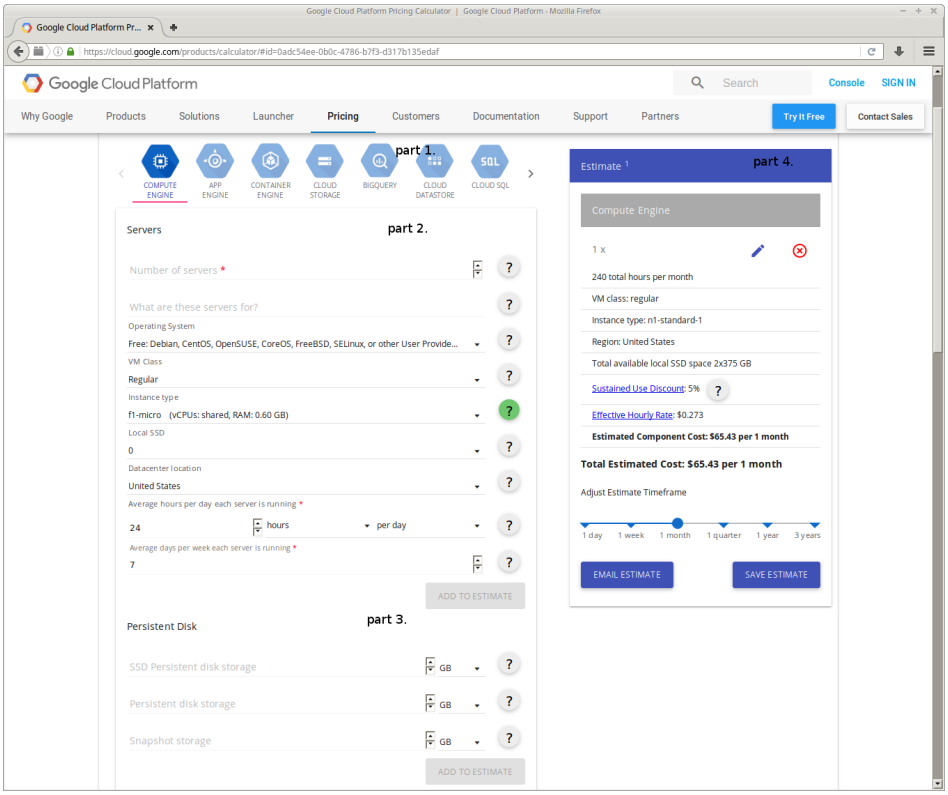


Figure 2.12.: Google Cloud Platform pricing calculator [gooc]

GCP provides a Pricing Calculator (Figure 2.12) as well, the layout is approximately the same with the calculator provided by Microsoft Azure (Figure 2.9), products navigator in part 1, required parameter form in part 2, optional parameter form in part 3, and the result in part 4.

3. Design

3.1. Requirements

Nefolog has provided two decision support services: `candidateSearch` and `costCalculator`, both of them support multiple types of parameters, the main shortcoming of service `costCalculator` is that it can only calculate the cost for one configuration at one time, but the result of `candidateSearch` for a given set of performance requirement can be quite a long list, so it is not convenient for the users to check the cost for each candidate manually. Starting from this, this thesis intended to extend the existing Nefolog with more complex Web Services, and a more user-friendly User Interface that can help users make their decisions more easily.

3.1.1. Functional Requirements

The functionalities that the extended Nefolog should provide contains:

- For a given set of performance and usage parameters, the system can search and return the cheapest configuration that is contained by the Knowledge Base.
- For a given service type, a provider, budget and usage information, the system can search and return all possible configurations.

The functionalities that the UI should provide:

- The user can browse the content of the Knowledge Base through a hierarchical structure for quick navigation.
- A search interface that accesses `candidateSearch`, the users can express their requirements more efficiently.
- A search interface that accesses `costCalculator`, the users can express their requirements more efficiently, and the result should be represented in both text and graphic format.
- A search interface that accesses new Web Service `cheapestConfig`, the users can express their requirements about the performance and usage, and the result should be the cheapest configuration that is possible.
- A search interface that accesses new Web Service `searchInBudget`, the users can express their requirements about the service type, usage and budget, and the result should be all the possible configurations.

3.1.2. Non-functional Requirements

- Performance. The Web Services should be able to return result in a acceptable period of time.
- Efficiency. The users should be able to access the Web Services with minimal effort.

3.1.3. Design Constraints

The work in [Xiu13] has been done in early 2013. Since that time, the way how the Cloud Computing providers provide their services has changed radically. For example, the HP Helion Public Cloud, was "sunset" on Jan. 31, 2016, as announced by a blog post [1] by senior vice president of HP Cloud.

For the other providers, their products and services are still continuously evolving, take Windows Azure as example, in the Knowledge Base from [Xiu13], there is an offering called web site, but in the current offerings provided by Microsoft Azure, this offering no longer exists (Figure 3.1).

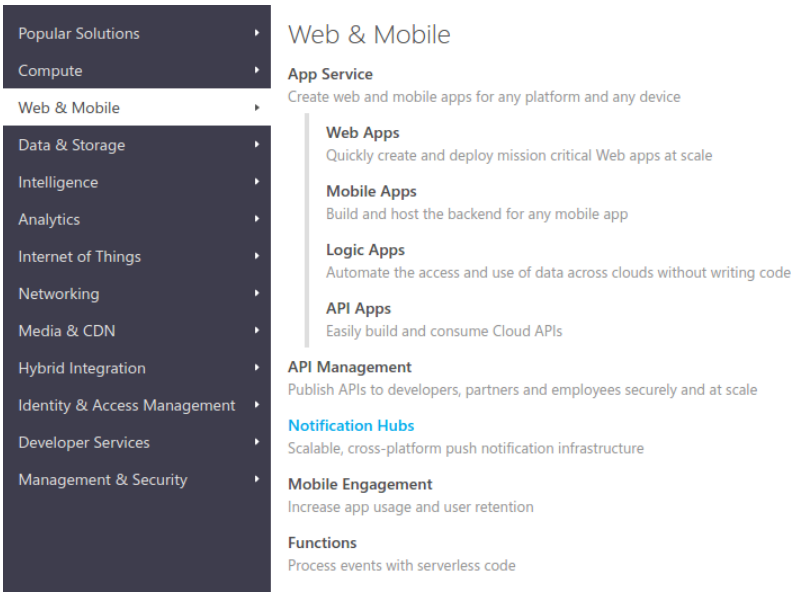


Figure 3.1.: Microsoft Azure Web & Mobile products [mica]

Table 3.1 summarizes part of the current offerings. In this table, I have surveyed the current offerings from three leading Cloud Computing providers: Amazon, Microsoft and Google, categorized the products which all three of them provide. Because of the tremendous evolution of the Cloud Computing market, It becomes more and more difficult to abstract a common model for the offerings from different providers, so updating the existing database is out of

[1] http://community.hpe.com/t5/Grounded-in-the-Cloud/A-new-model-to-deliver-public-cloud/ba-p/6804409#.V0XE4mF9603

the scope of this work, this thesis only uses the out-of-date Knowledge Base in [Xiu13], which is quite a challenge to dealing with the entries that no longer exist. The functionalities are therefore somehow constrained both in design and implementation phase, these constraints will be explained in the following when necessary.

Service Catagory	Service Type	AWS Offering	Azure Offering	GCP Offering
Compute	Virtual Machine	Amazon EC2	Virtual Machines	Google Compute Engine
	Container	Amazon EC2 Container Service	Azure Container Service (Azure Resource Manager)	Google Container Engine
	Web Application	AWS Elastic Beanstalk	App Service/ Cloud Services	Google App Engine
Networking	Virtual Network	Amazon VPC	Azure Virtual Network	Google Compute Engine Network
	Load Balancing	Amazon Elastic Load Balancing	Azure Load Balancer	Google Compute Engine Load Balancing
	Direct Connection	AWS Direct Connect	Azure ExpressRoute	Google Cloud Interconnect
	DNS	Amazon Route 53	Azure DNS	Google Cloud DNS
Storage	Object Storage	Amazon S3	Azure Blob Storage	Google Cloud Storage
	Block Storage	Amazon Elastic Block Storage (EBS)	Azure Premium Storage	Google Compute Engine
	Archive Storage	Amazon Glacier	Azure Backup	Google Cloud Storage Nearline
	CDN	Amazon CloudFront	Azure CDN	Google Cloud CDN
Database	RDMBS	Amazon RDS	Azure SQL Database	Google Cloud SQL
	NoSQL	Amazon DynamoDB	Azure DocumentDB	Google Cloud Datastore
	Cache	Amazon ElasticCache	Azure Redis Cache	Google App Engine Memcache
Monitoring	Resource Monitoring	Amazon CloudWatch	Azure Application Insights	Google Cloud Monitoring
	API Monitoring	Amazon CloudTrail	Azure Operational Insights	Google Cloud Monitoring

Table 3.1.: Service Category for current offerings

3.2. Specifications

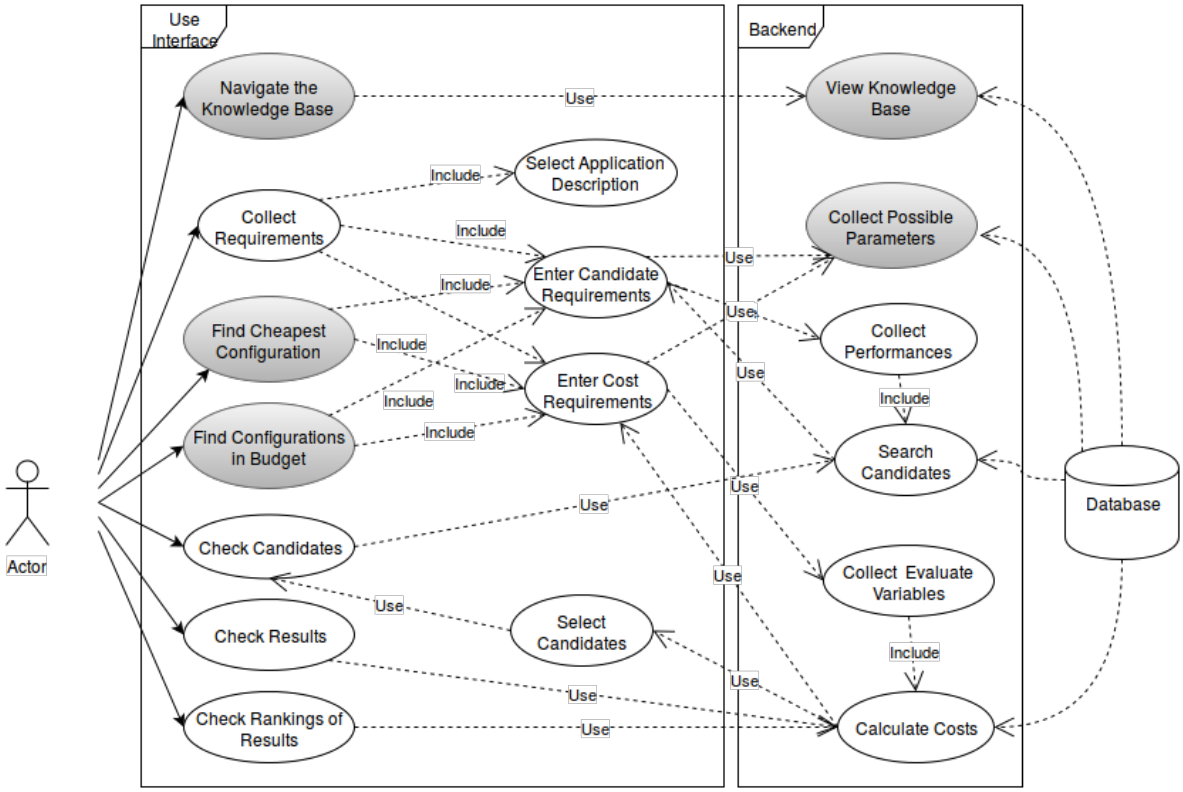


Figure 3.2.: Envisioned Use Cases

The envisioned functionalities are illustrated in Figure 3.2, which is also an extension of the work in [Xiu13], the newly added use cases are drawn in gray eclipses, they all rely on the existing use cases or database, the detailed description is listed in the following.

Use Case:	**Find Cheapest Configuration**
Goal:	The user wants to get the cheapest possible configuration for a given set of performance and usage information
Actor:	User
Precondition:	The possible parameters for a given service type, i.e. performances and variables are initialized in user interface for the user to input the value she desires.
Postcondition:	The cheapest possible configuration is shown in the user interface.

Normal Case:	1. The user initializes the input form by interaction with the navigator. 2. The user inputs the performances, variables and optionally the usage patterns in the form, and clicks the submit button. 3. The System shows the possible result in user interface. 4. The System shows the possible report with diagram in user interface.
Special Case:	2a. The user has inputted invalid parameter value. a) The system shows an error message and waits the user to correct.

Table 3.2.: Description of Use Case Find Cheapest Configuration

Use Case:	**Find Configurations in Budget**
Goal:	The user wants to get all the possible configurations for a given budget and usage information, under the scope of a service type of a provider
Actor:	User
Precondition:	The possible variable parameters for a given service type are initialized in user interface for the user to input the value she desires.
Postcondition:	All the possible configurations are shown in the user interface.
Normal Case:	1. The user initializes the input form by interaction with the navigator. 2. The user inputs the variables and optionally the usage patterns in the form, and clicks the submit button. 3. The System shows the possible result in user interface. 4. The System shows the possible report with diagram in user interface.
Special Case:	2a. The user has inputted invalid parameter value. a) The system shows an error message and waits the user to correct.

Table 3.3.: Description of Use Case Find Configurations in Budget

The remaining of the use cases are discussed in the following sections.

3.3. Design for Back-end

3.3.1. Knowledge Base

To provide a more hierarchical view for the content of the Knowledge Base, all the existing service types are categorized into certain categories This approach is used by all the Cloud providers that are surveyed in chapter 2. As shown in Figure 3.3, a new entity called Category has been added to the existing Entity-Relation Diagram, each category can have multiple service types and each service type can only belong to one category.

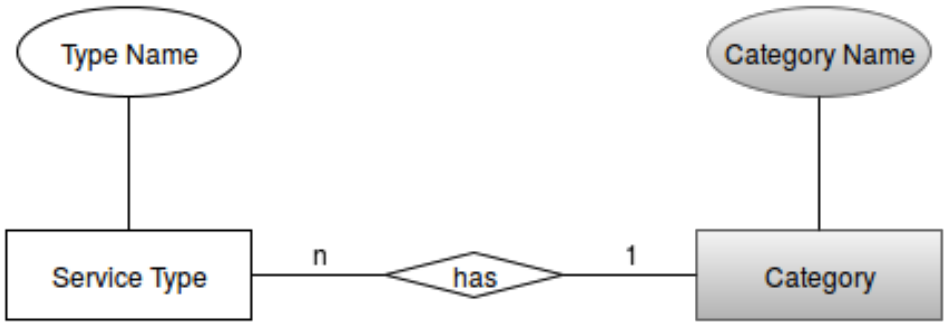


Figure 3.3.: Entity-Relation Diagram

The Knowledge Base should be updated following Table 3.1.

3.3.2. Web Services

Based on the design and implementation approach in [Xiu13], many Web Services are newly added or modified.

~/categories

For the purpose to view the content of Knowledge Base more hierarchically, a new Web Service with URI ~/categories should be able to return all the categories existing in the Knowledge Base, the result in form JSON should be like Listing 3.1:

```
{
  "name": "categories",
  "content": [
    {
      "name": "compute",
      "link": "/categories/compute"
    },
    {
      "name": "networking",
      "link": "/categories/networking"
    },
    {
      "name": "storage",
      "link": "/categories/storage"
    },
    {
      "name": "database",
      "link": "/categories/database"
    },
    {
      "name": "monitoring",
      "link": "/categories/monitoring"
    }
  ]
}
```

Listing 3.1: Example result for uri ~/categories

The "link" element in each category object represents the relative URI of the Web Service ~/categories/{category}, which contains the corresponding information for each category.

~/categories/{category}

This new created Web Service should return all the service types that belong to the given category, for example, the result of ~/categories/compute should be like:

```
{
  "name": "compute",
  "content": [
    {
      "name": "infrastructure",
      "link": "/serviceTypes/infrastructure",
      "deliveryModel": "IaaS"
    },
```

```
    {
      "name": "webSite",
      "link": "/serviceTypes/webSite",
      "deliveryModel": "IaaS"
    },
    {
      "name": "application",
      "link": "/serviceTypes/application",
      "deliveryModel": "IaaS"
    }
  ]
}
```

Listing 3.2: Example result for uri ~/categories/{category}

~/serviceTypes/{serviceType}

This modified Web Service should return all the offerings that belong to the given service type, for example, the result of ~/serviceTypes/infrastructure (partially) should be like:

```
{
  "content": [
    {
      "name": "elasticComputeCloud",
      "provider": "AmazonWebServices",
      "category": "compute",
      "link": "/offerings/elasticComputeCloud"
    },
    {
      "name": "VM",
      "provider": "WindowsAzure",
      "category": "compute",
      "link": "/offerings/VM"
    },

    ......

  ]
}
```

Listing 3.3: Example result for uri ~/serviceTypess/{serviceType}

~/offerings/{offering}

This modified Web Service should return all the configurations with performance information belong to the given offering, for example, the result of ~/serviceTypes/infrastructure (partially) should be like:

```
{
  "offering": {
    "name": "elasticComputeCloud",
    "provider": "AmazonWebServices",
    "link": "/offerings/elasticComputeCloud"
  },
  "configurations": [
    {
      "name": "m1.medium Light Utillization Standard Reserved
   Instances",
      "performance": {
        "cpuCores": 2,
        "cpuSpeed": 1.2,
        "memory": 3.75,
        "pio": "moderate",
        "platform": 3264,
        "storage": "410",
        "sla": 0.9995,
        "os": "Windows"
      }
    },

  ......

  ]
}
```

Listing 3.4: Example result for uri ~/offerings/{offering}

By accessing these 4 Web Services sequentially, i.e. ~/categories, ~/categories/{category}, ~/serviceTypes/{serviceType} and ~/offerings/{offering}, the users should be able to view the content of the Knowledge Base in a tree structure, when the users reach the leafs of the tree, they can get the all the configurations with performance information that each offering provides. These 4 Web Services are used by the use case View Knoledge Base (Figure 3.2).

~/parameters/{serviceType}

This URI should return all the possible parameters, e.g. performances and variables for the given service type, for example, the result of ~/parameters/infrastructure could be like:

```
{
  "performances": [
    "cpuCores",
    "cpuSpeed",
    "memory",
    "io",
    "platform",
    "bandwidth",
    "storage",
    "sla",
    "licence",
    "os"
  ],
  "variables": [
    "Hour"
  ],
  "locations": [
    "location_zone"
  ]
}
```

Listing 3.5: Example result for uri ~/parameters/{serviceType}

This Web Service is used by use case Collect Possible Parameters (Figure 3.2), there are 16 possible performances, 49 possible variables defined in the database from [Xiu13], when the users input these parameters, it is more convenient for the users to only input the relevant parameters, or only have the possible parameter input field in their view.

~/parameters/{offering}

This URI should return all the possible parameters, e.g. performances and variables for the given offering, this Web Service is also used by use case Collect Possible Parameters, and the result should also look like Listing 3.5.

~/cheapestConfig?{query}

This URI should return the cheapest possible configuration for the given query, the query should contain 2 part of parameters:

- The first part of the parameter should indicate the required information for how to find the configurations that meet the users' demand. This part of parameters can also be divided into two part:

  - Which group of configurations, e.g. the group of configurations that belong to the same provider, or the same service type, should be queried. If this information is not given by the user, the query should be performed on all configurations. This part should look like: servicetype=infrastructure.
  - The minimal performances the user demands. This part should look like: cpuCore=32&cpuSpeed=1.2&memory=60&io=very high&platform=64&storage=240&sla=0.9995&os=Linux.

- The second part of the parameter should indicate the user desired usage information, including where the product should be deployed, the variable and the related usage pattern. This part should look like: location_zone=Oregon&Hour=240&usage_pattern=(Hour,start=1,end=12,rate=10).

A separator between the first and the second part of parameters is put into the entire URI for the purpose of easier implementation, for example, the entire URI that combines all the parts above should be like: ~/cheapestConfig?servicetype=infrastructure&cpuCore=32&cpuSpeed=1.2&memory=62&io=very high&platform=64&storage=240&sla=0.9995&os=Linux&separator&location_zone=Oregon&Hour=240&usage_pattern=(Hour,start=1,end=12,rate=10). The result should look like the following Listing 3.6.

```
{
  "variables": [
    {"Hour": 240}
  ],
  "usagePatterns": [
    {"HourPattern": "start=1, end=12, rate=10"}
  ],
  "location_zone": "Oregeon",
  "result": {
    "configuration": {
      "name": "hi1.4xlarge Medium Utillization High-I/O On-Demand Instances",
      "performance": {
        "cpuCores": 35,
        "cpuSpeed": 1.2,
        "memory": 60.5,
        "pio": "very high",
        "platform": 64,
        "sla": 0.9995,
        "os": "Linux"
      }
    },
    "cost": 10638.20
  }
}
```

Listing 3.6: Example result for uri ~/cheapestConfig?{query}

Considering that for a given performance demand, there might be multiple candidate configurations that can meet the demand, to achieve a shorter query time, concurrency should be used when the cost for each configuration is calculated.

~/inBudgetConfigs?{query}

This URI should return all the possible configurations of a specific service type of a provider that satisfy a given budget and a given usage information. So the query should contain parameters that contains:

- The service type, e.g. servicetype=infrastructure.
- The provider, e.g. provider=AmazonWebServices.
- The usage information, e.g. location_zone=Oregon&Hour=240&usage_pattern=(Hour,start=1, end=12,rate=10).

The result should look like the following List 3.7.

```
{
  "serviceType": "infrastructure",
  "provider": "AmazonWebServices",
  "location_zone": "Oregon",
  "variables": [
    {"Hour": 240}
  ],
  "usagePatterns": [
    {"HourPattern": "start=1, end=12, rate=10"}
  ],
  "result": [
    {
      "configuration": {
        "name": "hi1.4xlarge Medium Utillization High-I/O On-
   Demand Instances",
        "performance": {
          "cpuCores": 35,
          "cpuSpeed": 1.2,
          "memory": 60.5,
          "pio": "very high",
          "platform": 64,
          "sla": 0.9995,
          "os": "Linux"
        },
```

```
24          "cost": 10638.20
25        }
26      ......
27
28  }
```

Listing 3.7: Example result for uri ~/inBudgetConfigs?{query}

This Web Service should also be implemented using concurrency.

3.4. Design for User Interface

3.4.1. Layout

Inspired by the cost calculator provided by Microsoft Azure [micb] and Google Cloud Platform [gooc], the envisioned layout of the User Interface is illustrated in the following Figure 3.4.

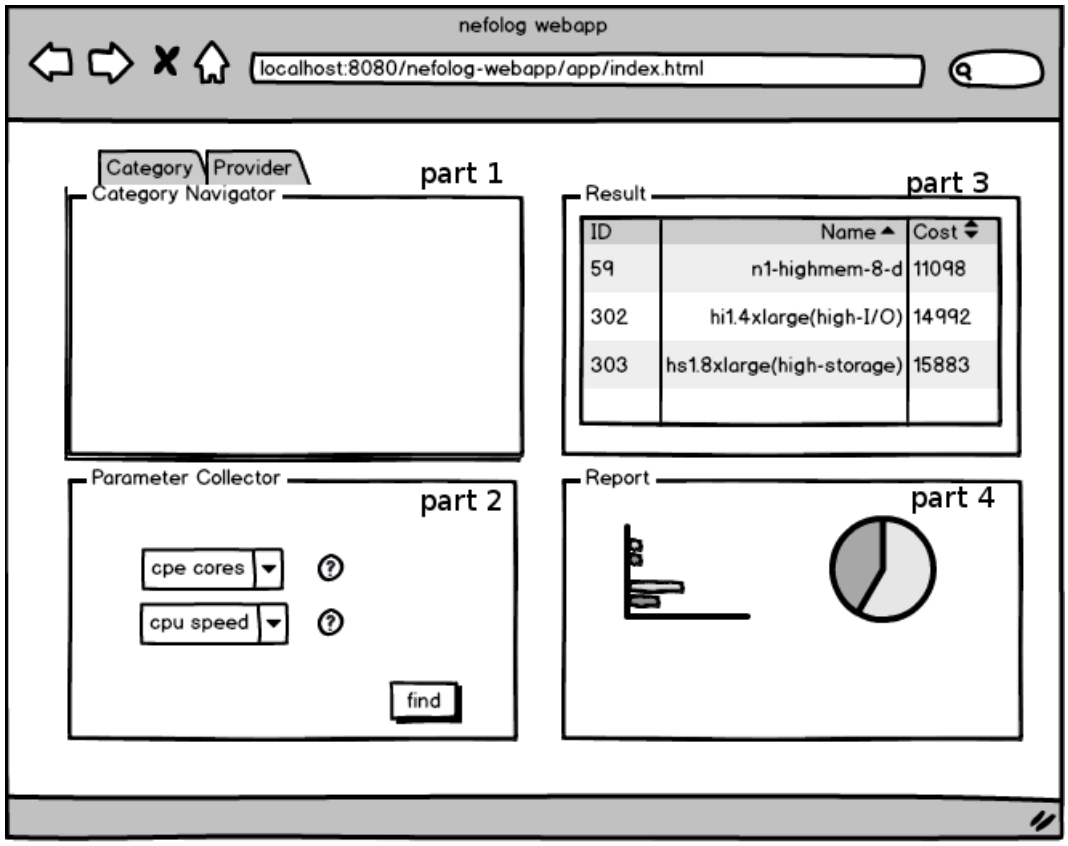


Figure 3.4.: Envisioned User Interface

The user should be able to view the content of the Knowledge Base in Navigator(part 1), the Navigator should provide two kinds of view, i.e. the category view and the provider view. Further more, the navigator should be able to give users the opportunity to acquire the access to Web Services like ~/cheapestConfig?{query} through certain user action. Then the user can input the desired parameters in the Parameter Collector(part 2). After certain user action like clicking the find button, the returned information should be shown in Result(part 3), and an autogenerated report containing explanatory diagrams should be shown in Report(part 4).

3.4.2. Design Pattern

The design of the User Interface follows the well known design pattern MVC(Model View Controller). MVC is a software structure that separates the system into three components: Model, View and Controller [Bur92], as in Figure 3.5 shown.

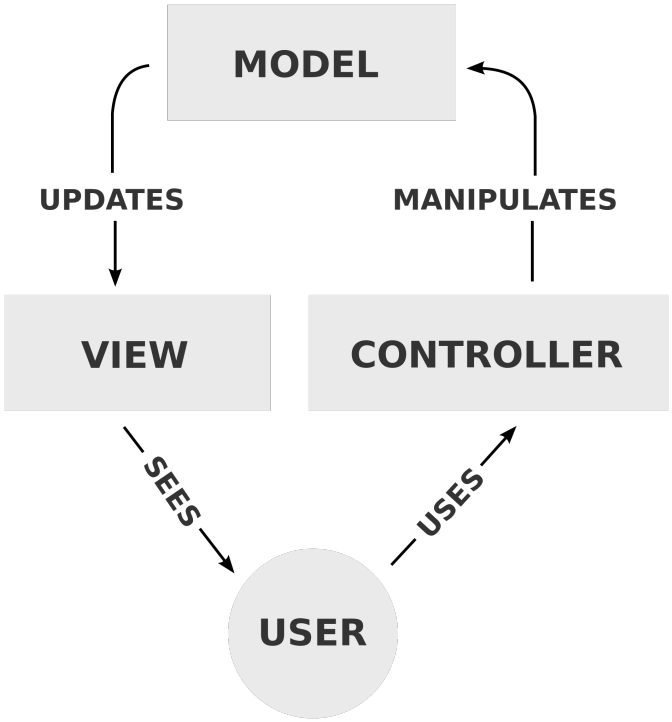


Figure 3.5.: A typical collaboration of the MVC components [mvca]

- The Model is responsible for managing the data in the system, it can respond to requests for information or user action. The model doesn't have to be aware of the existence of the other two components [mvcb].
- The View presents the content of the Model to the users and provides users the possibility to interact with the system, to further change the Model or the View.

- The Controller is the glue between the Model and View, it is in charge of changing the View when the Model changes, or the other way around.

This pattern decouples the Model from the View, i.e. how the data are manipulated from how they are displayed, this helps us for organizing the code more clean and makes the maintainability more possible.

Model

Most of the data that are needed to be shown to the user come from the database, i.e. from the Web Services described in Section 3.3.2, for the purpose of simplicity, this work follows design pattern Single Source of Truth [2], there is no cache designed for storing the data that are retrieved from the Web Services, in most cases, if a user action accessed the Web Services, the stored data will be repopulated with the result returned from the Web Service.

View

For the purpose to provide a user experience with homogeneous content, so that the user can expect the same content at the same position, all the components in Figure 3.4 should be able to dynamically change its content when the model changes.

Controller

Controller mainly takes the responsibility to handle the user actions, some user action might change more than one component in the Layout (Figure 3.4), e.g. the Result component and the Report component might change their content at the same time when the user clicked the find button, so the interaction between these components should also be the responsibility of the controller.

[2] https://en.wikipedia.org/wiki/Single_source_of_truth

4. Implementation & Evaluation

4.1. Technologies

The main technologies that are used during the implementation of this thesis are JDBC Connection Pool(Section 4.1.1), AngularJS(Section 4.1.2), Data-Driven Documents(D3) library(Section 4.1.3), I also used Restlet Framework to implement the Web Services, in a little different way than in [Xiu13] (Section 4.2.2).

4.1.1. JDBC Connection Pool

Introduction

To ensure the performance of the queries that are performed on the database during the implementation of the Web Services, JDBC Connection Pool was introduced. Each time a query through JDBC is performed on the database, there is a database connection [jdb] related to this task, and these connections are finite and scarce resource for the system [Hor13], therefore, it is a very costly way to obtain a connection for each query and close it after the query is done. The solution is to use Connection Pool, which is a mechanism that manages physical database connections in a queue, so that these connections can be reused without dragging down the performance of the system. However, the JDK and database vendors don't usually provide such a Connection Pool, the commonly used Connection Pool products are: JDBC Connection Pool [1], c3p0 [2], BoneCP [3] and so on. For the purpose of simplifying the dependency, JDBC Connection Pool was selected because the web application is deployed in Apache Tomcat.

Configuration

There are many ways to use the JDBC Connection Pool [toma], in this work, The JDBC Connection Pool is configured as a JNDI resource [jnd], and this resource is attached to a certain web application.

[1] https://tomcat.apache.org/tomcat-7.0-doc/jdbc-pool.html
[2] http://www.mchange.com/projects/c3p0/
[3] http://www.jolbox.com/

The developing environment is a Dynamic Web Project in Eclipse [4], the Web Application is deployed on Tomcat 7 [5]. The following Listing 4.1 contained configure file context.xml should be put into the folder Project_Root/WebContent/META-INF/.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Context>
3    <Resource
4      name="jdbc/CloudKB"
5      auth="Container"
6      type="javax.sql.DataSource"
7      factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
8      username="postgres"
9      password="123456789"
10     driverClassName="org.postgresql.Driver"
11     url="jdbc:postgresql://localhost:5432/PGCloudmigration"
12     initialSize="10"
13     maxActive="55"
14     maxIdle="21"
15     minIdle="13"
16     timeBetweenEvictionRunsMillis="33000"
17     minEvictableIdleTimeMillis="55000"
18     validationQuery="SELECT 1"
19     validationInterval="33000"
20     testOnBorrow="true"
21     removeAbandoned="true"
22     removeAbandonedTimeout="233"
23   />
24 </Context>
```

Listing 4.1: Configure file for JDBC Connection Pool ~/WebContent/META-INF/context.xml

Essential settings are listed in the first 8 lines(line 4 - line 11), the following settings are responsible for [tomb]:

- Sizing the Connection Pool(line 12 - line 17).
- Validating connections(line 18 - line 20).
- Dealing with connection leaks(line 21 - line 22).

The following configuration file(Listing 4.2) should be put in the web application's web.xml.

```
1    <resource-ref>
2      <description>Datasource for CloudKB</description>
3      <res-ref-name>jdbc/CloudKB</res-ref-name>
4      <res-type>javax.sql.DataSource</res-type>
```

[4] https://eclipse.org/
[5] http://tomcat.apache.org/

```
5     <res-auth>Container</res-auth>
6   </resource-ref>
```

Listing 4.2: Configure file for JDBC Connection Pool ~/WebContent/WEB-INF/web.xml

This registers the JNDI resource with name "jdbc/CloudKB" to the web application.

A JDBC Driver, e.g. postgresql-9.3-1104.jdbc41.jar, should be put into TOMCAT_ROOT_DIR/lib/.

After these configurations are done, a JDBC Connection Pool can be accessed by Servlets [6] or JSPs. As all the Web Services in this work are implemented by Restlet framework [7], and the Restlet application can be exposed as a Servlet [res], so we can implement the Web Services using the JDBC Connection Pool.

4.1.2. AngularJS

To implement the MVC design pattern described in Section 3.4.2, a brief survey has been made to choose a proper framework. There are a lot of choices, such as:

- .NET based ASP.NET MVC framework [8].
- Java based Apache Struts MVC framework [9].
- Javascript based MVC frameworks like backbone.js [10], sencha [11], AngularJS [12] and so on.

The popular Javascript MVC framework AngularJS is chosen, not only because the number of the documents available online for it, but also for the reason that the cost calculator provided by Google Cloud Platform in Figure 2.12 has used AngularJS.

Introduction

AngularJS framework is an open-sourced project that comes from Google, it offers the possibility to extend HTML vocabulary, i.e. new HTML tags for dynamic views in web-applications. Strictly speaking, it is called a Model-View-Whatever [anga] framework, the concepts that are essential to understanding the framework are [Gre13] :

- **Client-Side Templates**, the HTML template and the data are assembled by the browser, the server takes the responsibility to properly serve the data required by the templates.

[6] http://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html
[7] https://restlet.com/projects/restlet-framework/
[8] http://www.asp.net/mvc
[9] https://struts.apache.org/
[10] http://backbonejs.org/
[11] https://www.sencha.com/
[12] https://angularjs.org/

- **MVC**, in AngularJS applications, the View is the Document Object Model(DOM), the Controller is JavaScript objects, and the Model is stored in object properties.
- **Data Binding**, AngularJS' famous two-way data binding automatically synchronizes the data between the Model and View components.

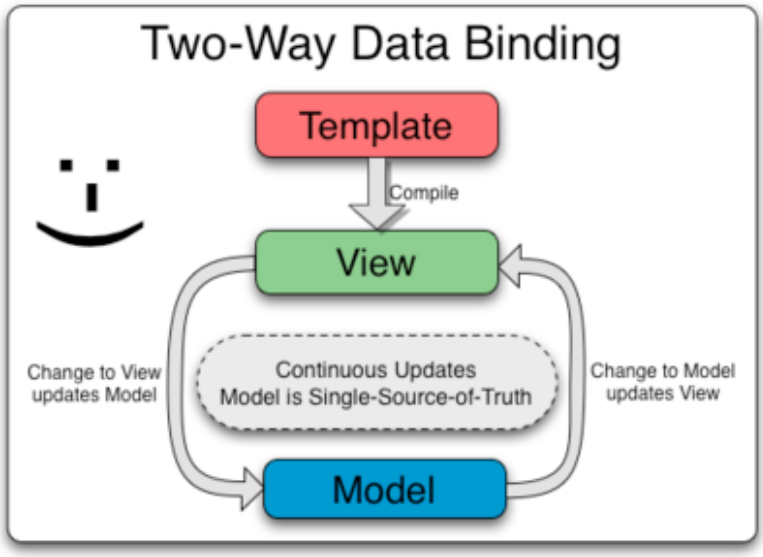


Figure 4.1.: Data Binding in Angular Templates [angb]

- **Dependency Injection**, the Angular injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested [angb], we can add other components very easily, for example, in the following code:

```
var app = angular.module("app", ["ngMaterial"]);
```

  I used an angular factory method `angular.module()` to add the libraries like `"ngMaterial"` [13] as dependency into the application `"app"`.

- **Directives**, directives can be transformed into DOM, AngularJS provides many powerful built-in directives that can help us to define the View, other than that, we can also write our own directives to do almost anything.
- **Scope**, when creating directives, we can choose to create a scope, which normally prototypically inherits from its parent scope, scopes are in hierarchical structure which mimic the DOM structure of the application. Scope refers to the Model of the application, which glues Controller and View. To make directive-to-directive communication, we can use `$broadcast` and `$emit` to dispatch events from directive to directive. As in Fiugre 4.2, all scopes are hierarchically children of the `$rootScope`, there is only one `$rootScope` in each AngularJS application, the events can be emitted from bottom to the top, or broadcasted from top to the bottom.

[13] https://material.angularjs.org/latest/

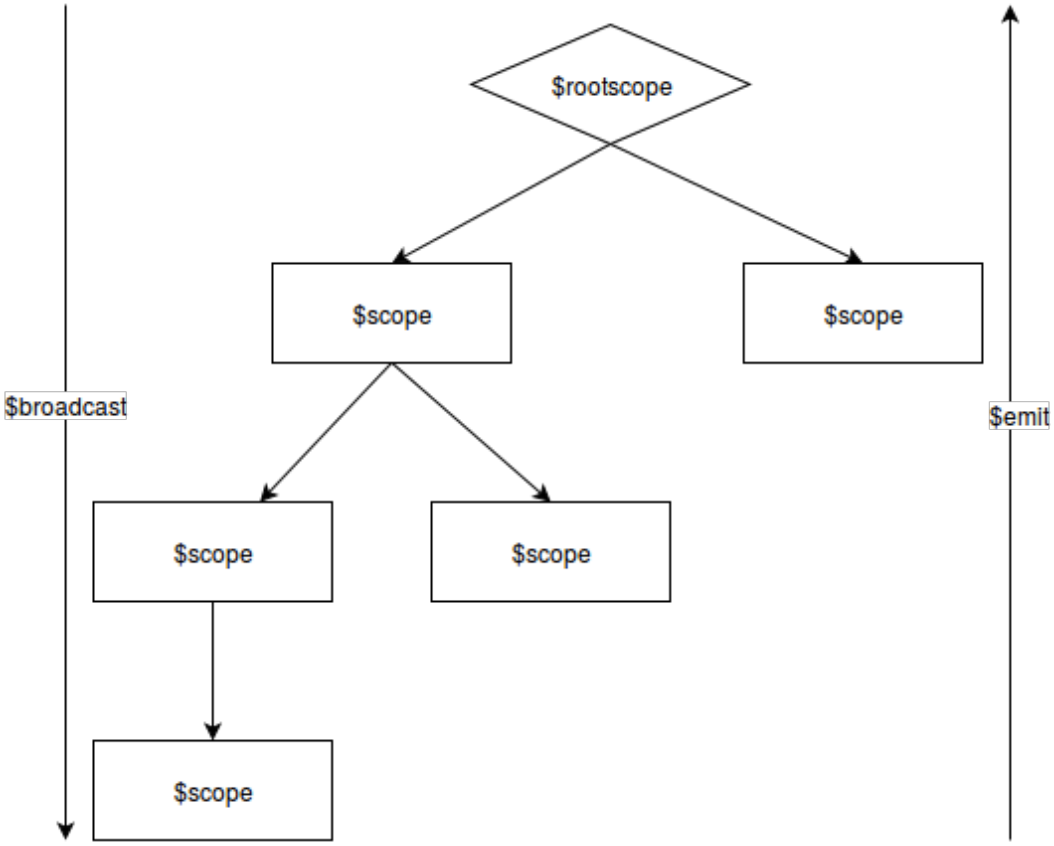


Figure 4.2.: `$broadcast` and `$emit` in AngulaJS

Denpendency

Our Web Application is implemented as a single-page application [14], the following libraries(dependencies) are needed by the AngularJS framework during the implementation of this thesis:

- CSS:

```
<link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.min.css">
<link rel="stylesheet" href="bower_components/angular-material/angular-material.css" />
```

- JS:

[14] https://en.wikipedia.org/wiki/Single-page_application

```
<script src="bower_components/angular/angular.js">
<script src="bower_components/angular-material/angular-material.js">
<script src="bower_components/angular-messages/angular-messages.js">
<script src="bower_components/angular-animate/angular-animate.js">
<script src="bower_components/angular-aria/angular-aria.js">
<script src="bower_components/angular-material/ngmaterial-svg-assets-cache.js">
<script src="bower_components/angular-smart-table/dist/smart-table.js"></script>

<!-- <script src="http://ngmaterial.assets.s3.amazonaws.com/svg-assets-cache.js"> -->
```

Note that the `svg-assets-cache.js` is needed for material-design icons but is not included in the original package, so I downloaded it and added it here locally.

The bower [15] library was used to manage the dependency in the implementation of Web Application. more detailed description is in appendix A.

4.1.3. Data-Driven Documents(D3.js)

D3.js was created for data visualization, there are plenty of libraries that are used for data visualization, such as:

- Java based JFreeChart [16], OpenChart2 [17] and so on.
- Javascipt based Highcharts [18], ECharts [19] and so on.

Although they are all very powerful and easy to use, yet D3.js is more powerful. D3.js provides us the ability to create rich interactive and animated content based on data and tie that content to existing web page elements(DOM) [Mee15]. Despite its steep learning curve, D3.js offers a very powerful tool for visualization of data.

Introduction

The core characteristics and functionalities of D3.js are:

- **SVG** D3.js allows us to create vector graphics [20] for charting and geospatial visualizations. The following Figure 4.3 shows a scenario that illustrates fluctuations since 2010 in California's 30-largest reservoirs [21].

  The blue circles represent the fluctuating storage levels for each reservoir, e.g. Lake Shasta, they will change their size automatically along with the time line in the top, and

[15] http://bower.io/
[16] http://www.jfree.org/jfreechart/
[17] http://approximatrix.com/products/openchart2/
[18] http://www.highcharts.com/
[19] https://ecomfe.github.io/echarts/index-en.html
[20] https://www.w3.org/Graphics/SVG/
[21] http://ww2.kqed.org/lowdown/2014/03/18/into-the-drought-californias-shrinking-reservoirs/

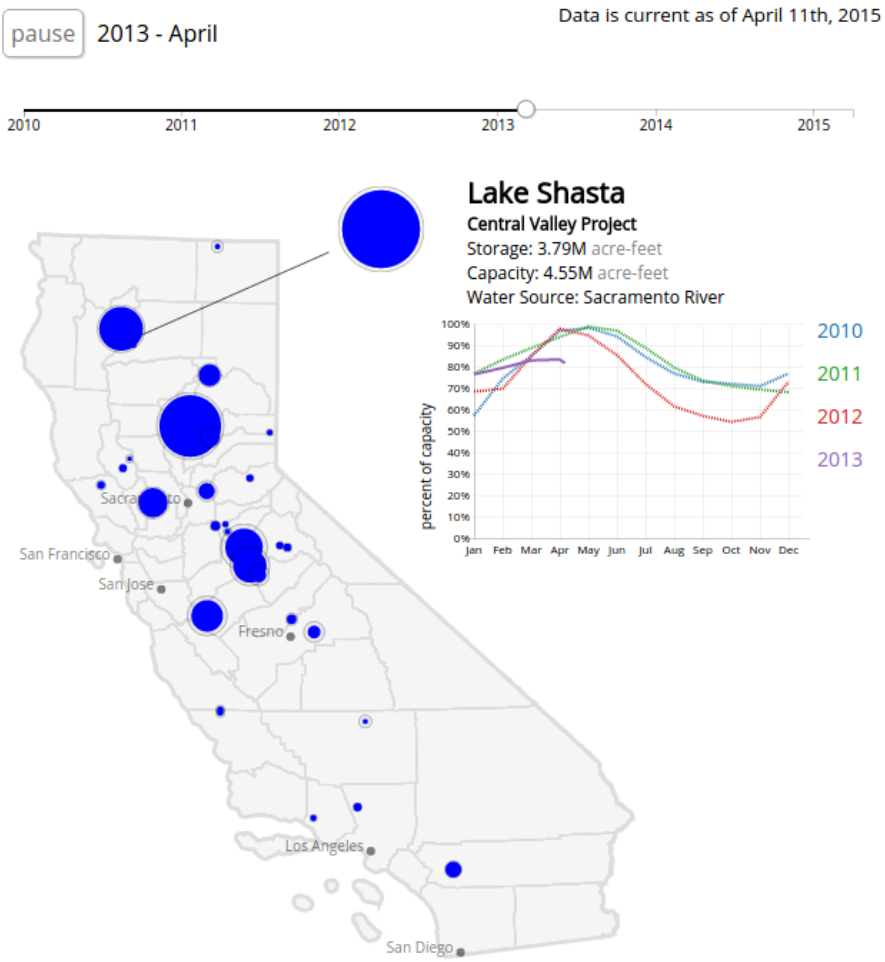


Figure 4.3.: D3.js svg example [Pow]

the multi-series line chart shows the trend for each year, which also changes along with the time line.

- **Selecting and Binding**, the real power of D3.js comes from these two functionalities, we can use selecting and binding to combine data and the web page elements. For example in the following code:

```
d3.selectAll("div.market").data([1, 4, 5, 2]);
```

We bind the elements in the array `[1, 4, 5, 2]` to `div` elements with the class of `"market"`. There are sometimes more DOM elements than data elements, or vice versa, D3.js has provided functions like `enter()` or `exit()` to create or remove DOM elements that are short or excess, so that the differences in the data can be reflected by the appearance of web page elements.

Dependency

D3 requires UTF-8 characters, so one way to ensure there's no strange error is to set:

```
<!DOCTYPE html><meta charset="utf-8">
```

The dependencies used by D3 in this thesis are:

```
<link rel="stylesheet" href="bower_components/d3-context-menu/css/d3-context-menu.css" />

<script src="bower_components/d3/d3.js">
<script src="bower_components/d3-context-menu/js/d3-context-menu.js">
```

4.2. Web Services

4.2.1. Accessing JDBC Connection Pool

In order to use the JDBC Connection Pool we configured in Section 4.1.1, a singleton class `DBUtil` is created, as shown in Listing 4.3:

```
1  public class DBUtil {
2
3    public static Connection getConnection() {
4      Connection con = null;
5      try {
6        Context initContext = new InitialContext();
7        Context envContext = (Context) initContext.lookup("java:/
     comp/env");
8        DataSource datasource = (DataSource) envContext.lookup("
     jdbc/CloudKB");
9        con = datasource.getConnection();
10     }
11     catch (NamingException ex) {
12
13       ......
14
15     }
16     return con;
17   }
18 }
```

Listing 4.3: `DBUtil.class`

The `Connections` used by `ServerResources` are all managed by JDBC Connection Pool.

4.2.2. Content-Viewing Web Services

The implementation of Web Services is a little different than in [Xiu13], take ~/categories/{category} as an example,

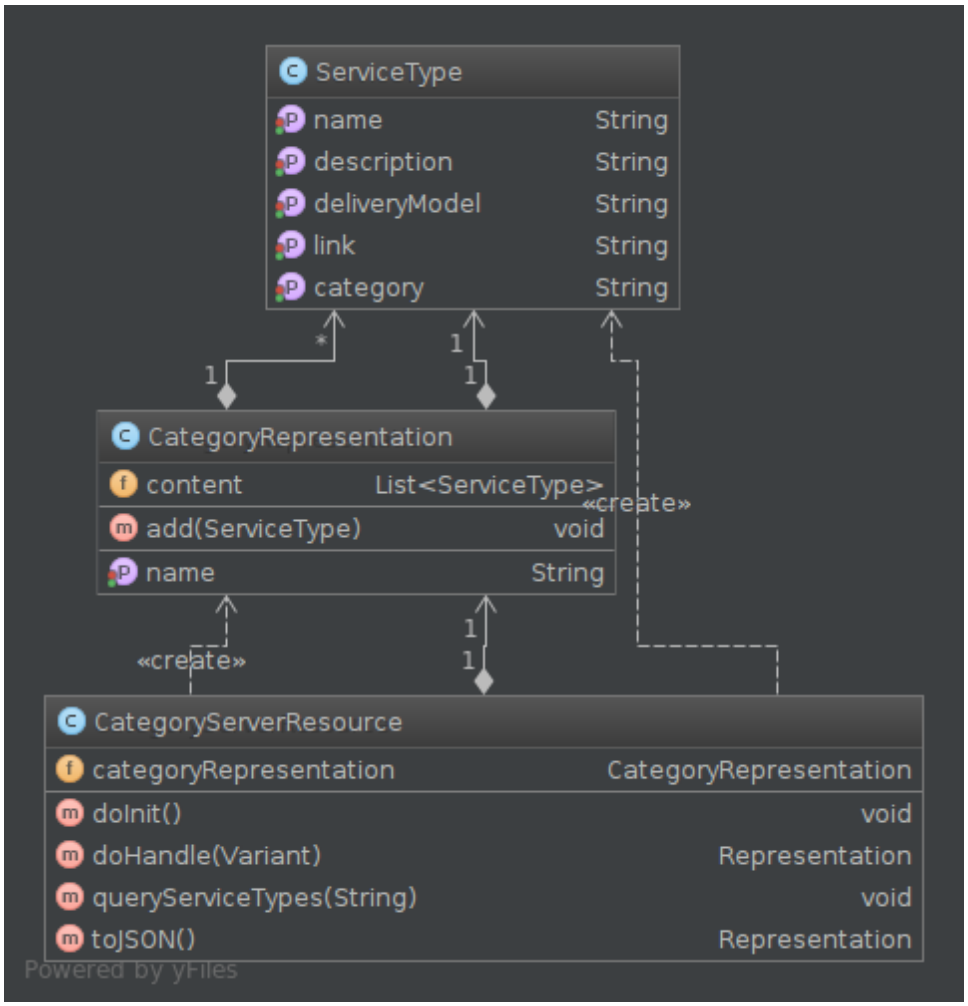


Figure 4.4.: Java Class Diagram for Web Service ~/categories/{category}

As in Figure 4.4 shows, a `CategoryRepresentation` class is used to generate the `Representation` of the result, which in this example, is a `List` of `ServiceTypes`, `ServiceType` is only a plain old Java object(POJO) [22] that will be created and populated during the life cycle of a `ServerResource` object .

Gson [23] is used to convert the objects like `categoryRepresentation` to JSON, the method `toJSON()` in `CategoryServerResource` only contains two lines of code:

```
Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

[22] https://en.wikipedia.org/wiki/Plain_Old_Java_Object
[23] https://github.com/google/gson

```
return new JsonRepresentation(gson.toJson(this.categoryRepresentation));
```

I only need the result in JSON format, so the default result returned by all the Web Services are in JSON.

4.2.3. Decision Support Web Services

~/parameters/{serviceType}

To return the possible parameters for a given service type, we need to query all possible performances, variables and locations that are related to the service type, take the performances as an example, we can get the performance names from `java.sql.ResultSetMetaData`(line 21 - line 24), and iterate all the result from query string(line 11 - line 16) to check which performance name has value in at least one result row(line 25 - line 31), in the end, all possible performance names is passed to the `paramsRepresentation`(line 33 - line 35).

```
1  private void queryAllPerformances(String nodeName) {
2    Connection conn = DBUtil.getConnection();
3
4    boolean[] existFlags = new boolean[16];
5    for(int i=0; i<existFlags.length; i++) {
6      existFlags[i] = false;
7    }
8
9    String[] performanceNames = new String[16];
10
11   String query = "SELECT configuration.configid, performance.* "
12     + "FROM servicetype, offering, configuration, performance "
13     + "WHERE offering.typeid = servicetype.typeid "
14     + "AND offering.offeringid = configuration.offeringid "
15     + "AND configuration.performanceid = performance.
       performanceid "
16     + "AND servicetype.type = '" + nodeName + "';";
17
18   try {
19     Statement stmt = conn.createStatement();
20     ResultSet rs = stmt.executeQuery(query);
21     ResultSetMetaData rsmd = rs.getMetaData();
22     for(int i=0; i<performanceNames.length; i++) {
23       performanceNames[i] = rsmd.getColumnName(i+3);
24     }
25     while(rs.next()) {
26       for (int i=0; i<16; i++) {
27         if(rs.getString(i+3) != null) {
28           existFlags[i] = true;
```

```
29          }
30        }
31      }
32
33      for( int i=0; i<existFlags.length; i++) {
34        if(existFlags[i]) {
35          this.paramsRepresentation.addPerformance(performanceNames
      [i]);
36        }
37      }
38
39    } catch (SQLException e) {
40      e.printStackTrace();
41    }  finally {
42      if(rs!=null) try { rs.close(); } catch (Exception ignore){}
43      if(stmt!=null) try { st.close(); } catch (Exception ignore){}
44      if(conn!=null) try { con.close();} catch (Exception ignore){}
45    }
46  }
```

Listing 4.4: Method `queryAllPerformances()` in `ParamsServerResource.class`

~/cheapestConfig?{query}

The implementation of this Web Service relies on three modified classes from Xiu's work in [Xiu13], as shown in Figure 4.5, they are `ConfigurationCandidate` from Xiu's `CandidateParams`, `CallableCostCalculator` from `CostResource`, and `Cost`.

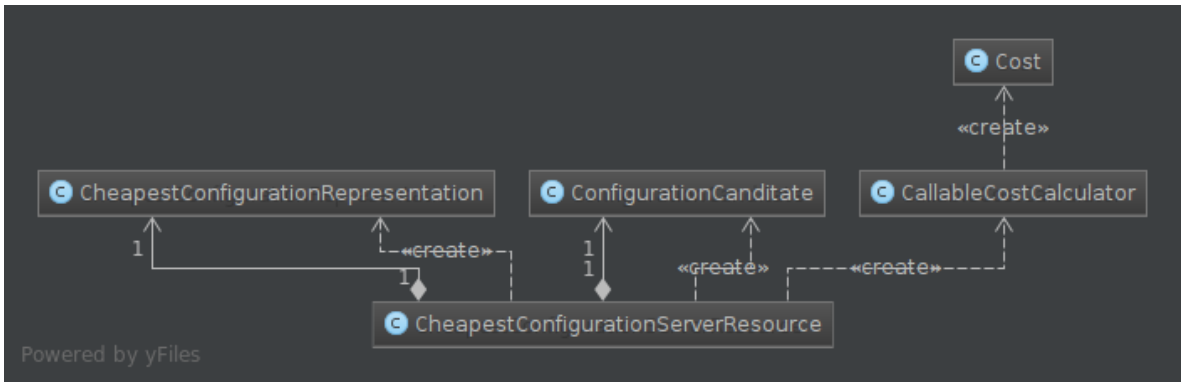


Figure 4.5.: Java Class Diagram for ~/cheapestCOnfig/{query}

As in Section 3.3.2 introduced, the user's input: ~/cheapestConfig?servicetype=infrastructure&cpuCore=32&cpuSpeed=1.2&memory=62&io=very high&platform=64&storage=240&sla=0.9995&os=Linux&separator&location_zone=Oregon&Hour=240&usage_pattern=(Hour,start=1,end=12,rate=10) will

first be divided into two part, the first part is used to find all the possible configuration candidates, the result is stored in `ConfigurationCandidate` object. The second part of the user input is used to instantiate a `List` of `CallableCostCalculator` that implements `java.util.concurrent.Callable`, concurrently, these instances will be used to calculate the cost for each configuration.

```
@Override
protected Representation get(Variant variant) {
  ......

  if (MediaType.APPLICATION_JSON.isCompatible(variant.
   getMediaType())) {

    try {
      Map<String, String> candidatesMap = candidateSearch();
      Set<String> configidSet = candidatesMap.keySet();

      List<FutureTask<Double>> futureTaskCalculatorList = new
   ArrayList<FutureTask<Double>>();

      Iterator<String> configItr = configidSet.iterator();

      while (configItr.hasNext()) {
        Form costFormWithConfigid = new Form();
        costFormWithConfigid.add("configid", configItr.next());
        costFormWithConfigid.addAll(costParamForm);

        CallableCostCalculator calculator = new
   CallableCostCalculator(costFormWithConfigid);

        FutureTask<Double> futureTask = new FutureTask<Double>(
   calculator);
        futureTaskCalculatorList.add(futureTask);
      }

      ExecutorService executor = Executors.newFixedThreadPool(16)
   ;

      for (FutureTask<Double> futureTask :
   futureTaskCalculatorList) {
        executor.execute(futureTask);
      }

      boolean allDone = false;
      int iAllTasks = configidSet.size();
```

```
34
35      while (!allDone) {
36        int iDoneTasks = 0;
37
38        try {
39          for (FutureTask<Double> futureTask :
    futureTaskCalculatorList) {
40            if (futureTask.isDone()) {
41              iDoneTasks++;
42            }
43          }
44
45          if (iDoneTasks == iAllTasks) {
46            executor.shutdown();
47            allDone = true;
48            getCheapest();
49          }
50
51        } catch (InterruptedException | ExecutionException e) {
52          e.printStackTrace();
53        }
54      }
55    } catch (SQLException e) {
56      e.printStackTrace();
57    }
58  }
59
60  ......
61 }
```

Listing 4.5: Method `get()` in `CheapestConfigurationServerResource.class`

The `CallableCostCalculator` has a constructor that takes a `org.restlet.data.Form` as argument, which contains the second part of user input(line 16 - line 20),
a `List` of `java.util.concurrent.FutureTask<Double>` objects is created, each of them wraps an instance of `CallableCostCalculator` and are executed concurrently, until all of them are done, we can get the cheapest configuration(line 22 - line 48).

4.3. User Interface

4.3.1. Layout

As mentioned in Section 4.1.2, Angular Material(ngMaterial) is used as UI Component framework that implements Google's Material Design Specification [24]. The following Listing 4.6 creates the layout shown in Figure 4.6.

```
1  <div ng-app="nefolog-app">
2  <entire-dir>
3
4  <md-content layout="column" layout-padding="">
5  <md-card class="md-default-theme">
6
7  <!-- Title card --> ...
8
9  </md-card>
10
11 <div ng-cloak="" >
12   <md-content class="md-padding" layout-xs="column" layout="row">
13     <div flex-xs="" flex-gt-xs="50" layout="column">
14       <md-card>
15        <md-card-title>
16          <md-card-title-text>
17            <span class="md-title">Navigator</span>
18          </md-card-title-text>
19        </md-card-title>
20        <navigator></navigator>
21      </md-card>
22
23     ......
24   </md-content>
25 </div>
26
27 </md-content>
28 </entire-dir>
29 </div>
```

Listing 4.6: HTML template for layout

The entire Web Application lives inside `<div ng-app="nefolog-app">`, each html element start with `md` is a directive defined by ngMaterial, such as `md-content`, our directives such as Navigator(line 20) is put in the corresponding `md-card` with the corresponding title.

[24] https://www.google.com/design/spec/material-design/introduction.html

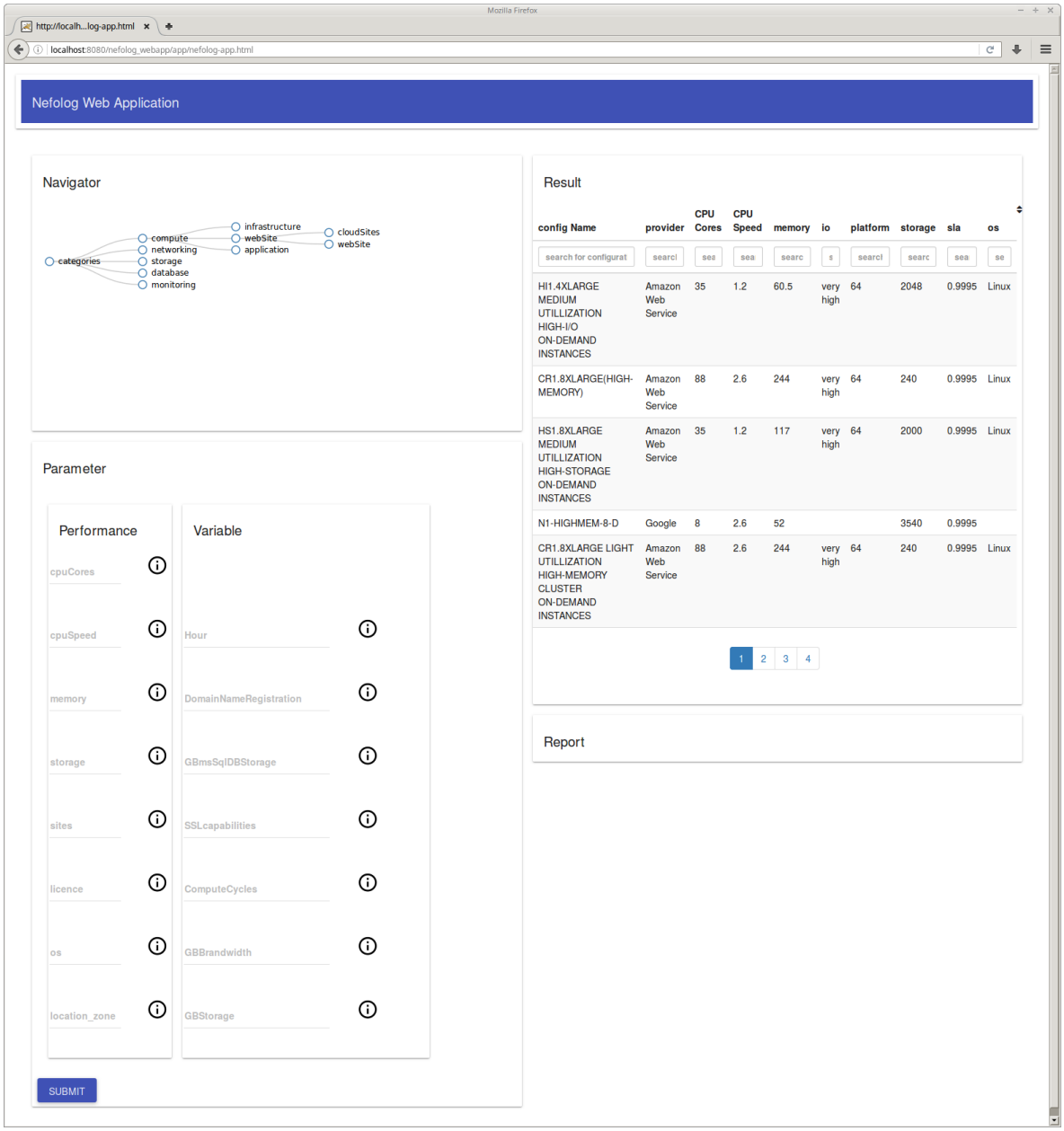


Figure 4.6.: Layout for User Interface

4.3.2. Navigator View

The Navigator is implemented as an AngularJS directive, which mainly contains a tree-layout D3 diagram , originated from the author of D3.js [Bos]. At initial state, it only contains one root node, i.e. categories in Figure 4.7, the related Model object is:

```
root = {
  "name": "categories",
```

```
3   "link": "/categories",
4   "linkFetched": false,
5   "children": []
6 };
```

Listing 4.7: Initial Model object of root node

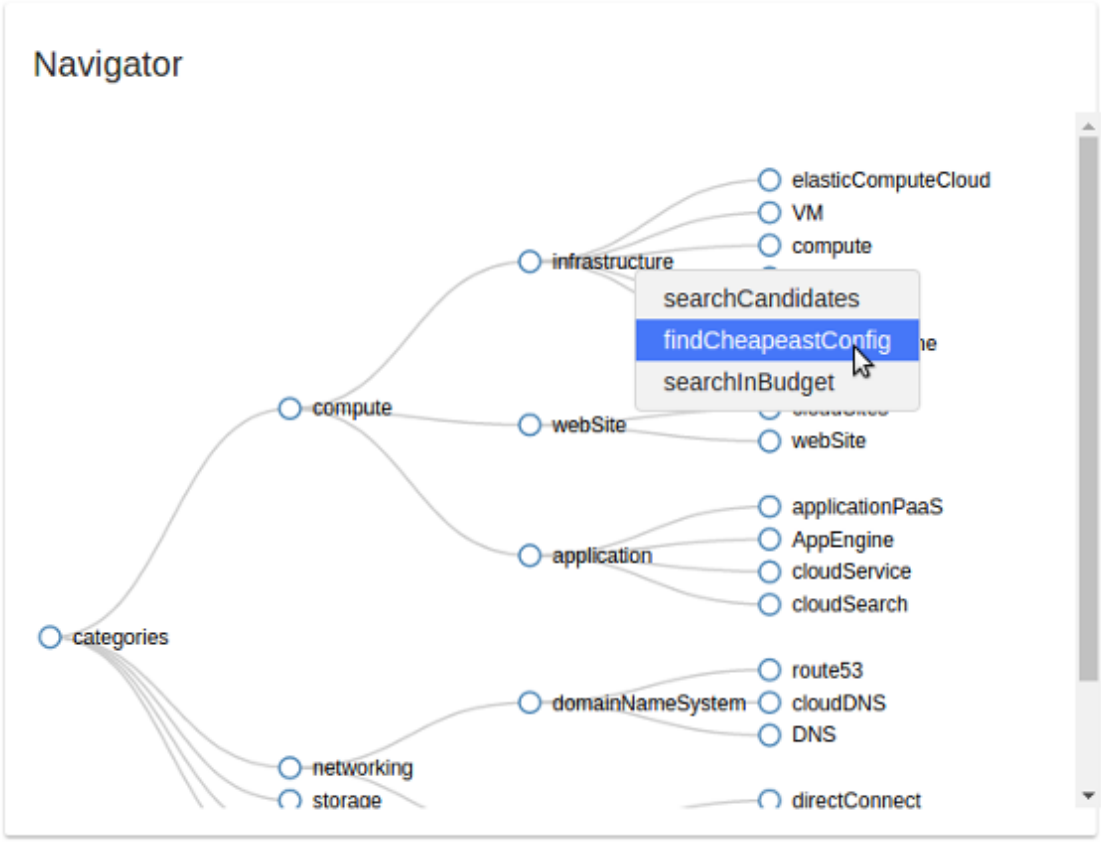


Figure 4.7.: Navigator View

When the user left-clicks any node in the tree, if the node that is being clicked is not a leaf node, a URI is generated by the `link`'s value, and method `d3.json()` will fetch the result from the corresponding Web Service(line 3), as shown in Listing 4.8. The returned data is used to populate the tree data, and refresh the View with the new Model object(line 6 - line 14).

```
1 function loadChildren(d) {
2
3   d3.json('http://localhost:8080/nefolog_webapp/api' + d.link,
     function (error, data) {
4     if (error) throw error;
5
6     var content = data.content;
```

```
7
8       d.children = [];
9       content.map(function (obj) {
10        d.children.push(obj);
11      });
12      d.linkFetched = true;
13
14      update(d);
15    });
16  }
```

Listing 4.8: Method `loadChildren()` in Navigator directive

Note that the URI is generated starting with `http://localhost:8080/nefolog_webapp/api`, this is because the Web Services that are implemented in this work don't support Cross-Origin Resource Sharing [25], so both the Web Services and the Web Application are deployed on the same server, but in different folders, as in Figure 4.6 shows, the URI of the Web Application is `http://localhost:8080/nefolog_webapp/app/nefolog-app.html`.

As the number of offerings(leaf nodes) is 51, it will take too much space if the tree is entirely uncollapsed, a scrollbar is dynamically added when the height of the tree exceed 400 pixels.

The user can right-click the `service type` nodes and `offering` nodes, a corresponding context menu [26] will be popped out accordingly(Figure 4.7). For `service type` nodes, the context menu contains the entry points for Web Service `candidateSearch` [Xiu13], `cheapestConfig` and `inBudgetConfigs`. For `offering` nodes, the context menu contains the entry points for Web Service `candidateSearch` and `cheapestConfig`. When an item in context menu is clicked, e.g. "findCheapestConfig", the event along with the data that is contained in the node is emitted to top most directive, as in Listing 4.9 shows.

```
1 {
2   title: 'findCheapeastConfig',
3   action: function(elm, d, i) {
4     scope.$emit('menuItemSelectedEvent', {node: d});
5   },
6 }
```

Listing 4.9: MenuItem `findCheapestConfig` in Navigator directive

The directive `entire-dir`(Listing 4.6) is in charge of making response to all the events that come from its children scope, it broadcasts the corresponding event back down to the directives that live inside it, the directive with corresponding listener will then react to the broadcast.

[25] https://www.w3.org/TR/cors/
[26] https://github.com/patorjk/d3-context-menu

4.3.3. Parameter View

The Parameter View is also a directive, it will respond to broadcast `broadcastSelectedServiceType`, that is when the user left-clicked a `service type` node, or `broadcastSelectedOffering`, when user left-clicked a `offerig` node. Accordingly, the content of Parameter View will change its content after its listener accessed the corresponding Web Service, ~/parameters/{serviceType}(Section 3.3.2) or ~/parameters/{offering}(Section 3.3.2), and populate the Model object with the returned data.

The user can then input the parameters in the input fields that are listed in two columns, the input fields on the left side are performances and location(Figure 4.8), on the right are variables. For variables, the user can input the usage pattern if necessary, for example, if user inputted `240,start=1,end=12,rate=10` in the blue input field, a query contains `usage_pattern=(Hour,start=1,end=12,rate=10)` will be generated for further accessing.

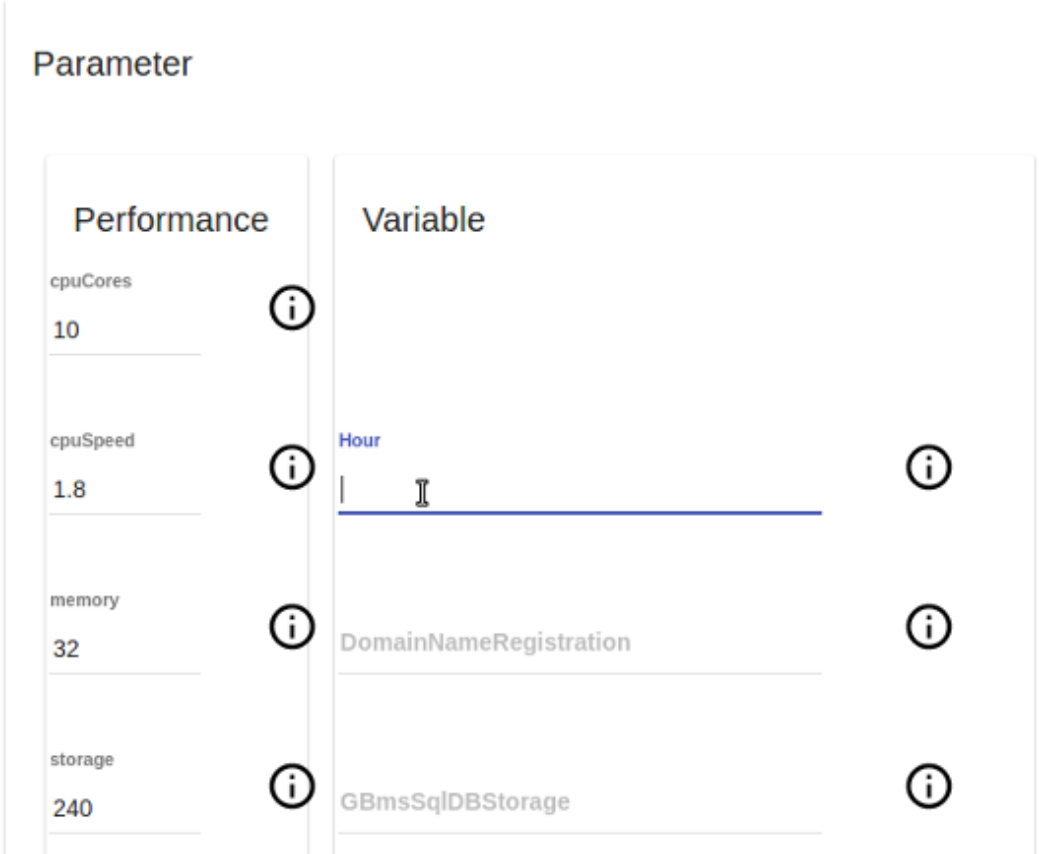
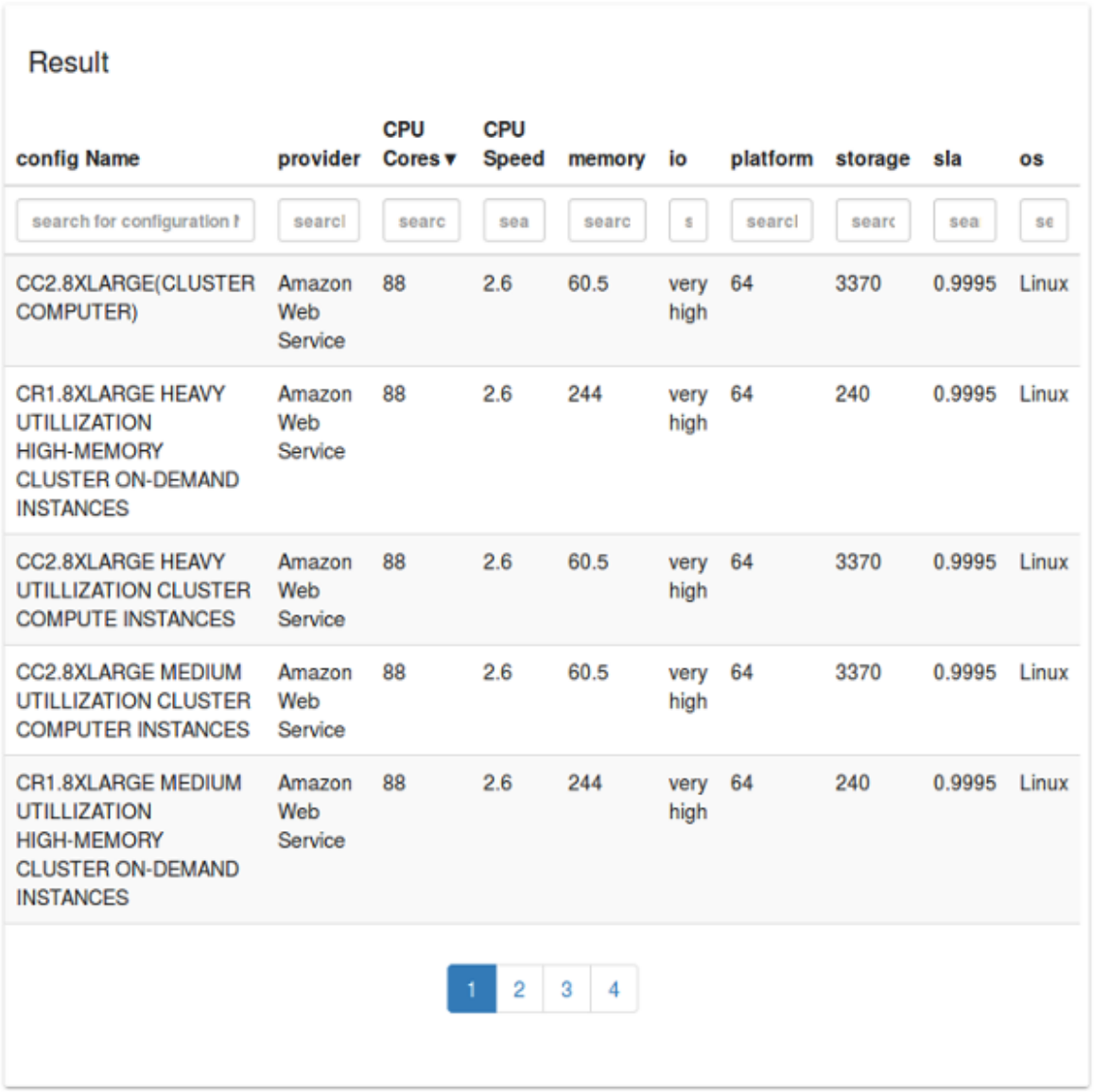


Figure 4.8.: Parameter View

It would be more convenient for the user, if each input field for performance can be changed to a `<md-select>`, and if the input field for variable can be changed to a set of `<md-select>`s, so that the user can pick the possible value from the drop-down list.

4.3.4. Result View

After the user finished the input and clicked the `SUBMIT` button, the corresponding Web Service is accessed and the result is shown in the Result View. Smart table [27] is used to generate tables. Currently, the users can click the table title to sort, for example in Figure 4.9, the result is sorted descending by the number of CPU cores, click one more time, it will become ascending. The search field for each column can be used to filter the result, for example, if `Google` is inputted in the search field under `provider`, only the configurations from Google will be listed.


Result

config Name	provider	CPU Cores ▾	CPU Speed	memory	io	platform	storage	sla	os
search for configuration f	searcl	searc	sea	searc	s	searcl	searc	sea	se
CC2.8XLARGE(CLUSTER COMPUTER)	Amazon Web Service	88	2.6	60.5	very high	64	3370	0.9995	Linux
CR1.8XLARGE HEAVY UTILLIZATION HIGH-MEMORY CLUSTER ON-DEMAND INSTANCES	Amazon Web Service	88	2.6	244	very high	64	240	0.9995	Linux
CC2.8XLARGE HEAVY UTILLIZATION CLUSTER COMPUTE INSTANCES	Amazon Web Service	88	2.6	60.5	very high	64	3370	0.9995	Linux
CC2.8XLARGE MEDIUM UTILLIZATION CLUSTER COMPUTER INSTANCES	Amazon Web Service	88	2.6	60.5	very high	64	3370	0.9995	Linux
CR1.8XLARGE MEDIUM UTILLIZATION HIGH-MEMORY CLUSTER ON-DEMAND INSTANCES	Amazon Web Service	88	2.6	244	very high	64	240	0.9995	Linux

1 2 3 4

Figure 4.9.: Result View

[27] http://lorenzofox3.github.io/smart-table-website/

4.3.5. Report View

Till the submission of this thesis, the Report View is still under development.

4.4. Evaluation

Performance of ~/cheapestConfig?{query}

To evaluate the performance, a simple approach for calculating the time difference is used. To be tested URL is: http://localhost:8080/nefolog_webapp/api/cheapestConfig?servicetype=infrastructure&cpuCore=8&cpuSpeed=1.2&memory=32&io=very high&platform=64&storage=240&sla=0.9995&os=Linux&separator&Hour=240&GBStorage=500&location_zone=Oregon&usage_pattern=(Hour,start=1,end=12,rate=10).

The first part of the query has the following result(simplified):

```
{
  hi1.4xlarge Medium Utillization High-I/O On-Demand Instances,
   cr1.8xlarge(high-memory),
  hs1.8xlarge Medium Utillization High-Storage On-Demand
   Instances,
  n1-highmem-8-d,
  cr1.8xlarge Light Utillization High-Memory Cluster On-Demand
   instances,
  hi1.4xlarge(high-I/O),
  cc2.8xlarge Light Utillization Cluster Compute Instances,
  hs1.8xlarge(high-storage),
  cr1.8xlarge Medium Utillization High-Memory Cluster On-Demand
   Instances,
  hi1.4xlarge Light Utillization High-I/O On-Demand Instances,
  double extra large,
  hs1.8xlarge Light Utillization High-Storage On-Demand Instances
   ,
  cc2.8xlarge Medium Utillization Cluster Computer Instances,
  cc2.8xlarge Heavy Utillization Cluster Compute Instances,
  cr1.8xlarge Heavy Utillization High-Memory Cluster On-Demand
   Instances,
  hs1.8xlarge Heavy Utillization High-Storage On-Demand Instances
   ,
  hi1.4xlarge Heavy Utillization High-I/O On-Demand Instances,
  cc2.8xlarge(cluster computer)
}
```

Listing 4.10: Simplified result of `candidateSearch`

As shown in Listing 4.10, there are 18 configurations, the `configid` of each one of them will be passed to a `callableCostCalculator`, then a `List` of `futureTasks` will concurrently execute the query for each configuration in the database(Listing 4.5).

The start time `long lStartTime = System.currentTimeMillis();` is counted right after the `executor` instance of `java.util.concurrent.ExecutorService` executes all the `futureTasks` (line 31 in Listing 4.5), the end time `long lEndTime = System.currentTimeMillis();` is counted when all `futureTasks` are done (line 59 in Listing 4.5). the following console output shows the result of one test run(Figure 4.10):

```
Tomcat v7.0 Server at localhost [Apache Tomcat] /opt/java/jdk1.7.0_79/bin/java (Apr 12, 2016, 9:08:51 PM)
13728.194148453873
 ------ futureTask.get() ------
10377.631280360818
 ------ futureTask.get() ------
15909.907122804249
 ------ futureTask.get() ------
6401.534206133883
 ------ futureTask.get() ------
0.0
 ------ futureTask.get() ------
10730.972136841276
 ------ futureTask.get() ------
10156.300909800604
 ------ futureTask.get() ------
12946.655476626049
 ------ futureTask.get() ------
0.0
 ------ futureTask.get() ------
6917.4031762304185
 ------ futureTask.get() ------
6852.734345591075
 ------ futureTask.get() ------
10401.981424560849
 ------ futureTask.get() ------
0.0
 ------ futureTask.get() ------
10467.113652664983
 ------ futureTask.get() ------
12317.347449912964
Elapsed milliseconds: 309
Apr 12, 2016 9:09:18 PM org.restlet.engine.log.LogFilter afterHandle
INFO: 2016-04-12        21:09:18        127.0.0.1       -       127.0.0.1       8080
```

Figure 4.10.: Performance of `/cheapestConfig?query`

All the `futureTasks` are done in less than 1/3 second, the test was on my laptop(Intel® Core™ i5-2520M processor (dual-core, 2.50GHz, 3MB Cache)).

5. Conclusion

5.1. Summary

There should be no doubt that Cloud Computing, especially the public Cloud services becomes more and more important across all industry sectors. For the purpose of helping customers to make their cloud migration choices, a system called Nefolog was created by Xiu. Based on a Cloud provider knowledge base, Nefolog exposes RESTful web services that are used by a JSP-based user interface MidSuS.

In order to provide more functionalities and more user-friendly experience to customers, after studied the Cloud provider knowledge base and the Nefolog, the current state of Cloud Computing services is surveyed. The web applications provided by 3 leading Cloud Computing providers are studied.

Some of the web services in Nefolog are refactored in a more object-oriented way, new web service for calculating multiple costs for different candidate configurations is created. To ensure that the customer won't wait too lang for a result, JDBC connection pool and concurrency is used during the implementation.

Based on MVC design pattern, a modern front-end framework AngularJS is used to implement the new single page web application that is inspired from the cost calculator of Google and Microsoft. Now the user can first interact with a navigator that is created by D3.js, the dynamically generated parameter view then gathers more requirements from the user, and finally, the result is represented to the user with more opportunities to make further decision.

5.2. Future work

The Cloud provider knowledge base needs a thorough update.

During the implementation of this work, there are many limitations need to be improved:

- The ~/inBudgetConfigs?{query} web service is not yet implemented. The idea is basically similar with the ~/cheapestConig?{query}, which is to concurrently calculate multiple configurations that fulfill certain demand, but it can help users to deal with the common scenario: budget.

- The Report View which is designed to show the result graphically, is not yet finished. Considering the result of `costCalculator` contains geographical information, monthly cost for data transfer and so on, these data would be more intuitive with graphical expression by using D3.
- The Navigator View should provide a provider tree, or a products matrix view, which can help users to browse the knowledge base more flexible.
- The Parameter View need more refinement, for example, when the user inputted an invalid parameter, there should be a error message, and the submit button should be disabled, until the user has inputted all the necessary parameters. The usage pattern should be gathered more user friendly, not just in plain text.
- There are also more improvements can be done in Result View, for example, when a user selected two configurations in the result table, a report is generated in the Report View, which shows the comparison between the two configurations in detail.

Appendix A.

Bower Dependency

bower.json:

```
{
  "name": "nefolog-webapp",
  "authors": [
    "huangjinhui <whocodeworld@gmail.com>"
  ],
  "description": "diplomarbeit",
  "main": "",
  "moduleType": [],
  "license": "MIT",
  "homepage": "",
  "ignore": [
    "**/.*",
    "node_modules",
    "bower_components",
    "test",
    "tests"
  ],
  "dependencies": {
    "angular-material": "1.0.6",
    "d3-context-menu": "*",
    "angular-smart-table": "^2.1.8",
    "angular-bootstrap": "^1.3.3"
  }
}
```

Listing A.1: bower.json

`angular-material` relies on `angular`, `angular-animate`, `angular-aria`, `angular-messages`, `d3-context-menu` relies on `d3`, bower will automatically install the proper version of them.

Bibliography

[ABLS13] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch. How to adapt applications for the Cloud environment. *Computing*, 95(6):493–535, 2013.

[anga] AngularJS : MVW framework - Model-View-Whatever. https://plus.google.com/+AngularJS/posts/aZNVhj355G2. Accessed: 2016-05-30.

[angb] Guide to AngularJS Documentation. https://docs.angularjs.org/guide. Accessed: 2016-05-30.

[ARXL14] V. Andrikopoulos, A. Reuter, M. Xiu, and F. Leymann. Design Support for Cost-efficient Application Distribution in the Cloud. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 697–704. IEEE, 2014.

[ASL13] V. Andrikopoulos, Z. Song, and F. Leymann. Supporting the migration of applications to the cloud through a decision support system. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 565–572. IEEE, 2013.

[awsa] Amazon Web Services Simple Monthly Calculator. https://calculator.s3.amazonaws.com/index.html. Accessed: 2016-05-30.

[awsb] Amazon Web Services(AWS) - Cloud Computing Services. https://aws.amazon.com. Accessed: 2016-05-30.

[awsc] Cloud Products & Services Amazon Web Services. https://aws.amazon.com/products/. Accessed: 2016-05-30.

[awsd] TCO Calculator. https://awstcocalculator.com/. Accessed: 2016-05-30.

[Bos] M. Bostock. Collapsible Tree. https://bl.ocks.org/mbostock/4339083. Accessed: 2016-05-30.

[Bur92] S. Burbeck. Applications programming in smalltalk-80 (tm): How to use model-view-controller (mvc). *Smalltalk-80 v2*, 5, 1992.

[gooa] Google Cloud Computing, Hosting Services & APIs. https://cloud.google.com. Accessed: 2016-05-30.

[goob] Google Cloud Computing TCO Pricing Calculator. https://cloud.google.com/pricing/tco/. Accessed: 2016-05-30.

[gooc] Google Cloud Platform Pricing Calculator. https://cloud.google.com/products/calculator/. Accessed: 2016-05-30.

[Gre13] B. Green. *Angularjs*. O'Reilly Media, Sebastopol, CA, 2013.

[Hor13] C. Horstmann. *Core Java*. Prentice Hall, Upper Saddle River, NJ, 2013.

[jdb] The Java™ Tutorials - JDBC Basics. https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html. Accessed: 2016-05-30.

[jnd] The Java™ Tutorials - Java Naming and Directory Interface. https://docs.oracle.com/javase/tutorial/jndi/. Accessed: 2016-05-30.

[Mee15] E. Meeks. *D3.js in action*. Manning Publications, Shelter Island, NY, 2015.

[MG11] P. Mell and T. Grance. The NIST definition of cloud computing. 2011.

[mica] Microsoft Azure: Cloud Computing Platforms & Services. https://azure.microsoft.com. Accessed: 2016-05-30.

[micb] Pricing Calculator | Microsoft Azure. https://azure.microsoft.com/en-us/pricing/calculator/. Accessed: 2016-05-30.

[mvca] Model–view–controller. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. Accessed: 2016-05-30.

[mvcb] MVC Architecture. https://developer.chrome.com/apps/app_frameworks. Accessed: 2016-05-30.

[Pow] V. Powell. How the Drought is Shrinking California's Reservoirs [Visualization]. http://ww2.kqed.org/lowdown/2014/03/18/into-the-drought-californias-shrinking-reservoirs/. Accessed: 2016-05-30.

[res] Restlet Framework - User Guide. https://restlet.com/technical-resources/restlet-framework/guide/2.3/introduction/overview. Accessed: 2016-05-30.

[Son13] Z. Song. A decision support system for application migration to the Cloud. http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-3381&engl=0, 2013. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany.

[toma] The Tomcat JDBC Connection Pool. https://tomcat.apache.org/tomcat-7.0-doc/jdbc-pool.html. Accessed: 2016-05-30.

[tomb] Tomcat JDBC Connection Pool configuration for production and development. http://www.codingpedia.org/ama/tomcat-jdbc-connection-pool-configuration-for-production-and-development/. Accessed: 2016-05-30.

[Xiu13] M. Xiu. Decision support for different migration types of applications to the Cloud. http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-3472&engl=0, 2013. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany.

All links were last followed on June 1, 2016

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, June 1, 2016 _____

(Name)