LinkedUSDL Pricing API

Daniel Alberto Guedes Barrigas
barrigas@student.dei.uc.pt

Faculdade de Ciências e Tecnologia da Universidade de Coimbra
Departamento de Engenharia Informática

Summary

- Objectives
- Cloud Pricing CODEX
- Pricing API
- Conclusion

Introduction

- The current pricing scene of today’s Cloud market is complex. Pricing methods vary between providers, each with it’s own advantages and disadvantages for both provider and consumer. But, despite of their differences, they also have some features in common[1][2].

- 451Research.com[1] attemps to simplify the current pricing scene by identifying the similarities between the different pricing methods and grouping them according to those similarities.

- This survey produced The Cloud Pricing CODEX – A taxonomy that classifies all pricing methods into 1 of 8 diferente categories.

[1] https://451research.com/report-long?icid=2770
[2] http://thoughtsoncloud.com/2013/06/present-future-cloud-pricing-models/

Introduction (2)

- In order to prove the (conceptual and pratical) validity of the LinkedUSDL pricing model and the Pricing API, I proposed a challenge that takes into consideration the survey performed by 451Research. Through this challenge I'll provide a pratical and conceptual proof that both API and pricing model can tackle todays Cloud pricing scene.

- The challenge consists on trying to model at least one Service Offering from each of the eight identified pricing methods on the Pricing CODEX using both the API and the LinkedUSDL pricing model.

Objectives

- Prove that the current pricing model and API can deal with the complex pricing scene of today's cloud computing services.

- Provide an abstraction layer between developers and the LinkedUSDL model in order to ease and promote its usage.

- Present several exemples that introduce the current state of the API and it's functionalities.

The Pricing Codex

- 451Research – “ … 451 Research, a division of leading global analyst and data company The 451 Group, is focused on the business of enterprise IT innovation. “ [3]

- The Codex is an attempt to simplify the pricing scene of today’s Cloud Computing market. It tries to help both providers and consumers, providing them with a set of tools to help them make more informed decisions and develop improved strategies for their business.

- The CODEX is a Taxonomy that tries to label every existing pricing method under one of its eight possible pricing methods.

[3] https://451research.com/about

The Pricing Codex (2)

- The Codex’s eight methods:

1. **PrePaid VM Access** – The customer pays upfront the full cost of the service. Let’s say that the user needs a VM for 7 or 8 months and the cost of that service would amount to a total of 12,000Euros. With this method, the customer needs to pay the 12,000 Euros up-front and only after the payment has been done is the customer able to access the VM.
2. **Recurrent PrePaid VM Access –** With this method, the customer commits to paying a recurring fee (usually monthly), over a certain period of time. We can look at this like the PrePaid VM Access but, instead of paying everything upfront, we can pay it in arrears over the duration of a certain period of time.
3. **On-Demand –** Probably one of the most used pricing methods nowadays. With this method the customer only pays for what he used and, contrary to the previous two methods, the customer is only billed after the service has been used not before.
4. **Reserved Instance –** This method is similar to the On-Demand method but with a little difference, this method charges an initial flat fee to the customer, which in a way works has a form of commitment from the user, and consequently provides a lower consumption fee for the Virtual Machine over the duration of the contracted term (usually 1 or 3 years).
5. **Spot Pricing –** This method is different from the others in a very particular way. While the other methods work with a fixed fee, the price of this service varies over time. What this means is that today, the price for the instance may be 0,015EUR per hour but tomorrow or 2,3,4 days from now, it can either go up or down, depending on the current state of the market and the providers environment. For example: Amazon’s cloud platform may be underused and as such, the price of the spot instance will probably decrease. On the other hand, in case their platform is busy and there aren’t many resources available, the price of the instance can increase considerably. In order to access this type of instance, the customers usually place a bit stating the price they’re willing to pay for those resources; if their bid exceeds the current price, they’re granted the access to those resources.

The Pricing Codex (3)

- The Codex’s eight methods (2):

6. **PrePaid Credit VM Access** – This method is very similar to the PrePaid VM Access method where the customer pays upfront the full cost of the service but, aside from being granted access to the instance, the customer also receives a certain amount of a virtual currency from the provider; let’s call them credits. Costs of consumption are then debited from these credits.

7. **Recurrent PrePaid Credit VM Access –** Like Recurrent PrePaid VM Access, this method also requires a commitment from the consumer. The consumer commits to paying a set amount in advance, on a recurring basis (usually monthly). Costs of consumption are then debited from the pre-payed amount which was converted into credits.

8. **Recurrent Resource Pooling –** In a way, this method is similar to the Recurrent PrePaid Credit VM Access(RPPC) but, while RPPC converts the prepaid amount into credits, in this method the customer buys the quantity of resources he needs (e.g: CPU,RAM, Disk size, ..) and costs of consumption are then debited from the quantitity of purchased resources.

The Pricing Codex (4)

- The Codex’s bundling methods: Aside from these 8 different methods, the CODEX also labels the way the provider charges his customer for the provided services. “Bundling refers to the practice of combining several products for a single price.” [1]
- “In any bundled offering, consumers must choose from preconfigured bundles, which many include the virtual machine CPU and RAM, the local (or external) storage, and the network bandwidth. Consumers are charged a single unit price for consuming the virtual machine, rather than for consuming individual resources. In an unbundled offering, consumers choose the CPU, RAM and storage themselves, aggregating them into a VM; consumers are then charged a unit for each resource. “ [1]
- On their survey they identified 4 different types of bundling:
  - **Fully Bundled (FB)** – CPU,RAM,Disk size and bandwidth are all bundled as a single package and provided in exchange of a single fee. Consumers must choose from a number of virtual machines with pre-defined quantities of resources.
  - **VM Bundled (VB)** – Consumers must choose from a number of virtual machines with predefined quantities of CPU, memory and disk. Bandwidth is charged separately.
  - **Processor Bundled (PB)** - Consumers must choose from a number of virtual machines with predefined quantities of CPU and memory. Bandwidth and disk are charged separately.
  - **Unbundled (UB) –** CPU, memory, disk and bandwidth are all charged separately and are fully controlled by the consumer, subject to a minimum charge.

LinkedUSDL PricingAPI

- JAVA API whose main objective is to reduce the gap between developers and the LinkedUSDL pricing model by providing a mean to programmatically interact with it. Through the API, the modeling of services following a LinkedUSDL approach becomes relatively simple.

- To use the API, the developer just needs to add the API to his project's buildpath along with necessary third party libraries. Once this step is concluded, all that is necessary is to import and "populate" the JAVA classes with the service information as depicted on the examples provided on github [4].

- The API is composed by the following JAVA classes:
  - **LinkedUSDLModel –** Container that holds rest of the JAVA Classes. This class is also responsible for other operations from which we can highlight the readModel and writeToModel operations. These operations are the ones that "transform" our JAVA classes into a Semantic model that follows the LinkedUSDL data structure.

[4] https://github.com/jorgearj/USDLPricing_API

LinkedUSDL PricingAPI (2)

- The API is composed by the following JAVA classes (2):
  - **Offering–** JAVA Class that represents an instance of the ServiceOffering object from the LinkedUSDL core [5]. This class is the container that holds an/several object(s) of the type Service and an object of the PricePlan type. This class serves has the linking point between the LinkedUSDL Core and the LinkedUSDL Pricing model.
  - **Service –** JAVA Class that represents an instance of the Service object from the LinkedUSDL Core. This class is the container that holds the features of the Service. For example, the RAM size of the instance, software like Oracle or MySQL Databases, it's Operating System, etc… These features are represented by the QualitativeValue and QuantitativeValue classes. As their names imply, QuantitativeValue class deals with values that can be quantified like RAM,CPU speed,Disk size, etc while the QualitativeValue class deals with features whose value is a concept with some semantic meaning attached to it; for example, the type of disk used on the instances: it can either be the classic HDD hardware or the "new" and improved SSD.
  - **QuantitativeValue –** As just mentioned, this class is responsible for dealing with features whose values are quantifiable, like RAM,CPU,Disk size, etc.
  - **QualitativeValue –** This class is responsible for dealing with features whose values are concepts with some semantic value attached to them.
  - **PricePlan –** Represents an instance of the PricePlan object from the LinkedUSDL Pricing model. Class responsible for holding every piece of information necessary to calculate the cost the Offering's Service(s). Each PricePlan can be composed of by one or more PriceComponents; These PriceComponents can be seen as the billable attributes of the Service, for example, an instance where we have to pay a fee for including the OracleDB. This extra fee would be represented as a PriceComponent of the PricePlan.

[5] https://github.com/linked-usdl/usdl-core

LinkedUSDL PricingAPI (3)

- The API is composed by the following JAVA classes (3):
  - **PriceComponent–** JAVA Class that represents an instance of the PriceComponent object from the LinkedUSDL Pricing model[6]. This class represents the billable (or deductions) attributes of the Offering's Service(s). An important feature of this class is that it can model both **static** and **dynamic** pricing values. Modeling static values is simple, just set the pre-defined value and it's finished(**PriceSpec** class). In order to model the dynamic nature of the pricing methods the API supports the definition of mathematical expressions in the PriceComponent class to calculate the final price of a service based on the user's preferences (**PriceFunction** class).
  - **PriceSpec –** Represents an instance of the PriceSpec object from the GoodRelations ontology [7]. This class represents a static price value.
  - **PriceFunction –** JAVA Class that represents an instance of the PriceFunction object from the LinkedUSDL Pricing model[6]. Through this class we can define mathematical expressions that will calculate a personalized price based on some values provided by the user/customer (Usage variables) and some pre-defined values (Provider variables). There are two types of expressions that are currently supported:
    - **Simple Expressions –** A simple mathematical expression to calculate the price, e.g: numberOfMonths (set by the customer) * instanceFeePerMonth (set by the provider/developer).
    - **Composed Expressions –** Mathematical expressions whose behavior is controller by some conditions, e.g: IF (nrOfMonths > 10) ; BEHAVIOR ~ ELSEIF (nrOfMonths < 10) ; BEHAVIOR2 . As you can see, the condition is and must always be separated from its behavior by the ";" character. IF and ELSEIF conditions must be separated by the "~" character. This feature is explored on the examples provided on [4].

[6] https://github.com/linked-usdl/usdl-price
[7] http://semanticweb.org/wiki/GoodRelations

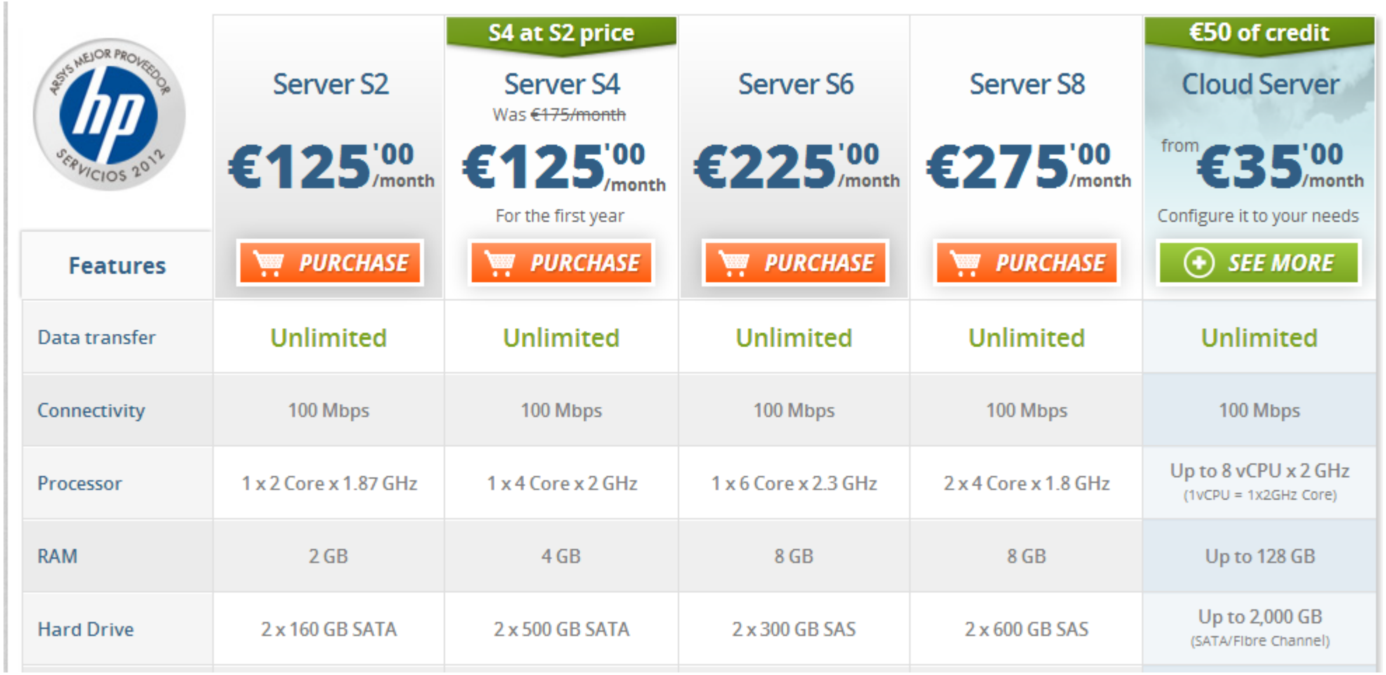
Challenge – Use Cases

Method	Provider	Bundling
Recurring Resource Pooling	VMWare' 'vCloud Hybrid'	PB
PrePaid Credit	Microsft' 'Azure Virtual Machines'	VB
PrePaid Subscription Credit	CloudSigma' 'Cloud'	UB
PrePaid VM	IDC Frontier' 'Cloud'	PB
On-Demand	Amazon' 'EC2'	VB
Spot Pricing	Amazon' 'EC2'	VB
Reserved Instance	Amazon' 'EC2'	VB
Recurring PrePaid VM	Arsys' 'Dedicated Server'	FB

Table.1 – Challenge use cases

Arsys Example

- As mentioned on the previous table, Arsys dedicated/VPS servers use a fully bundled Recurring PrePaid VM Access model. In our example we cover the dedicated servers however, the modeling method is exactly the same for the VPS servers.
- They provide four different offerings: S2,S4,S6,S8, each with it's pre-defined features and characteristics.

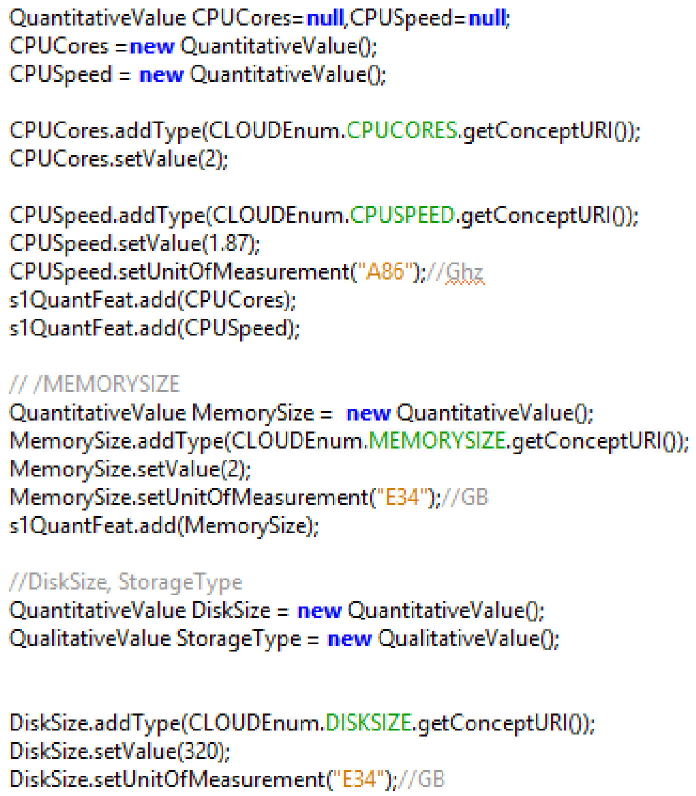


	Server S2	Server S4	Server S6	Server S8	Cloud Server
		S4 at S2 price			€50 of credit
		Was ~~€175/month~~			
	€125'00 /month	€125'00 /month	€225'00 /month	€275'00 /month	from €35'00 /month
		For the first year			Configure it to your needs
Features	PURCHASE	PURCHASE	PURCHASE	PURCHASE	SEE MORE
Data transfer	Unlimited	Unlimited	Unlimited	Unlimited	Unlimited
Connectivity	100 Mbps	100 Mbps	100 Mbps	100 Mbps	100 Mbps
Processor	1 x 2 Core x 1.87 GHz	1 x 4 Core x 2 GHz	1 x 6 Core x 2.3 GHz	2 x 4 Core x 1.8 GHz	Up to 8 vCPU x 2 GHz (1vCPU = 1x2GHz Core)
RAM	2 GB	4 GB	8 GB	8 GB	Up to 128 GB
Hard Drive	2 x 160 GB SATA	2 x 500 GB SATA	2 x 300 GB SAS	2 x 600 GB SAS	Up to 2,000 GB (SATA/Fibre Channel)

Arsys Example (2)

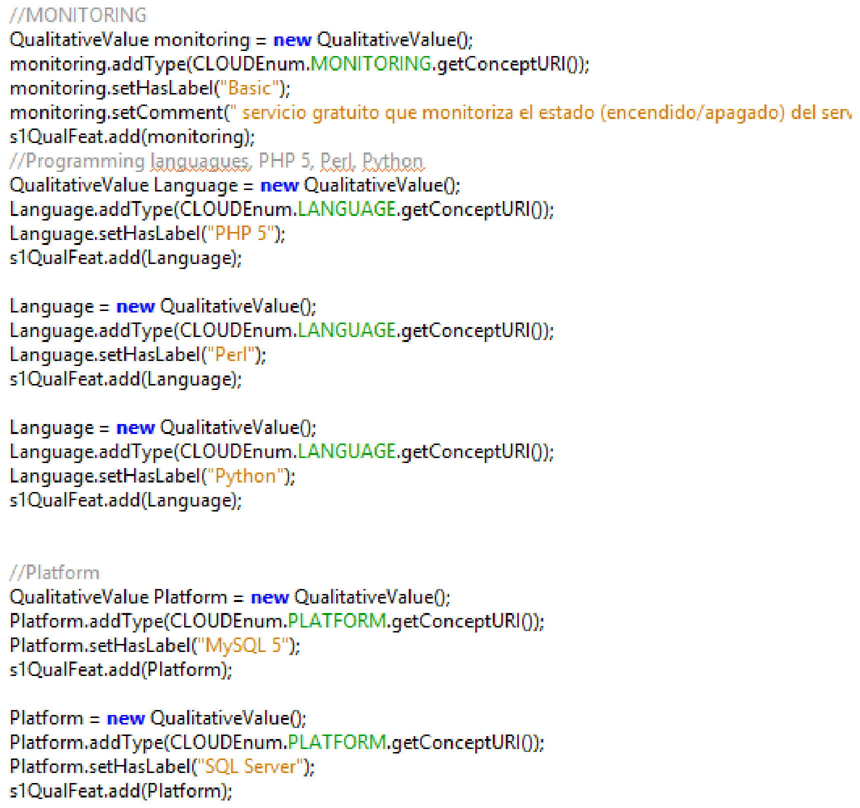
- Following the classes of the Pricing API, I created 1 service (to model the S2 Offering) and added its corresponding features as presented on their website. Each of its features were modeled using the QuantitativeValue or QualitativeValue classes.

E.g:QuantitativeValue Features:



```
QuantitativeValue CPUCores=null,CPUSpeed=null;
CPUCores =new QuantitativeValue();
CPUSpeed = new QuantitativeValue();

CPUCores.addType(CLOUDEnum.CPUCORES.getConceptURI());
CPUCores.setValue(2);

CPUSpeed.addType(CLOUDEnum.CPUSPEED.getConceptURI());
CPUSpeed.setValue(1.87);
CPUSpeed.setUnitOfMeasurement("A86");//Ghz
s1QuantFeat.add(CPUCores);
s1QuantFeat.add(CPUSpeed);

// /MEMORYSIZE
QuantitativeValue MemorySize =  new QuantitativeValue();
MemorySize.addType(CLOUDEnum.MEMORYSIZE.getConceptURI());
MemorySize.setValue(2);
MemorySize.setUnitOfMeasurement("E34");//GB
s1QuantFeat.add(MemorySize);

//DiskSize, StorageType
QuantitativeValue DiskSize = new QuantitativeValue();
QualitativeValue StorageType = new QualitativeValue();


DiskSize.addType(CLOUDEnum.DISKSIZE.getConceptURI());
DiskSize.setValue(320);
DiskSize.setUnitOfMeasurement("E34");//GB
```

E.g:QualitativeValue Features:



```
//MONITORING
QualitativeValue monitoring = new QualitativeValue();
monitoring.addType(CLOUDEnum.MONITORING.getConceptURI());
monitoring.setHasLabel("Basic");
monitoring.setComment(" servicio gratuito que monitoriza el estado (encendido/apagado) del serv
s1QualFeat.add(monitoring);
//Programming languagues, PHP 5, Perl, Python
QualitativeValue Language = new QualitativeValue();
Language.addType(CLOUDEnum.LANGUAGE.getConceptURI());
Language.setHasLabel("PHP 5");
s1QualFeat.add(Language);

Language = new QualitativeValue();
Language.addType(CLOUDEnum.LANGUAGE.getConceptURI());
Language.setHasLabel("Perl");
s1QualFeat.add(Language);

Language = new QualitativeValue();
Language.addType(CLOUDEnum.LANGUAGE.getConceptURI());
Language.setHasLabel("Python");
s1QualFeat.add(Language);


//Platform
QualitativeValue Platform = new QualitativeValue();
Platform.addType(CLOUDEnum.PLATFORM.getConceptURI());
Platform.setHasLabel("MySQL 5");
s1QualFeat.add(Platform);

Platform = new QualitativeValue();
Platform.addType(CLOUDEnum.PLATFORM.getConceptURI());
Platform.setHasLabel("SQL Server");
s1QualFeat.add(Platform);
```

Arsys Example (3)

- After modeling the service features, the next step is to create its corresponding Offering and PricePlan as well.
    - Offering of = new Offering(); of.addService(s1); PricePlan pp = new PricePlan(); of.setPricePlan(pp);

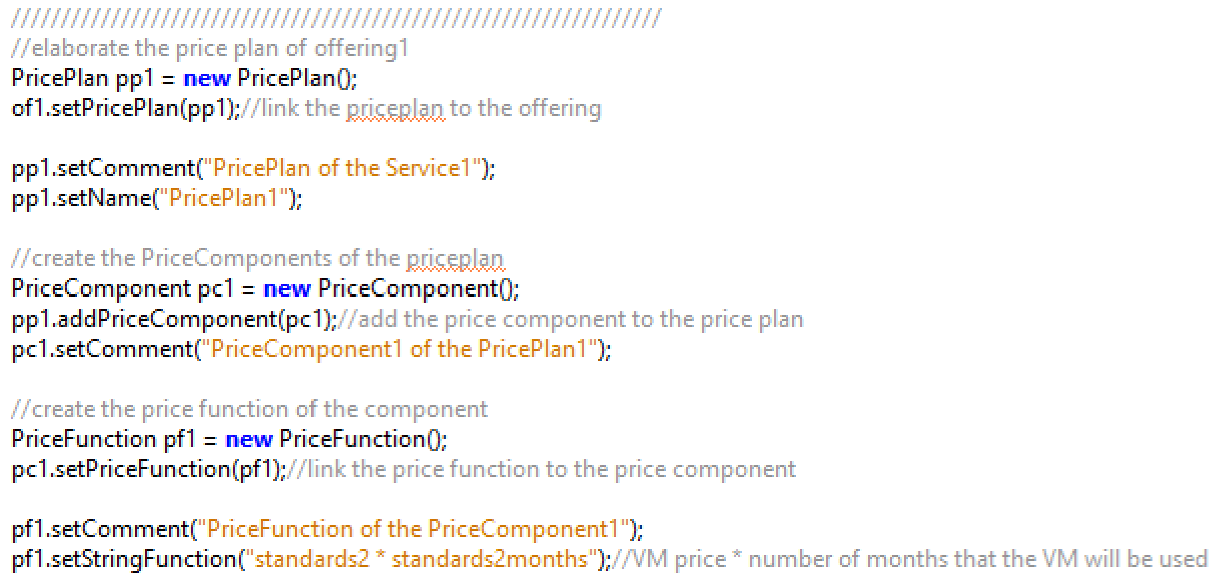
- Now we focus on one of the key points of the challenge: Model the Recurrent PrePaid model using the LinkedUSDL Pricing model.
- As we know, in order to access the instance/dedicated server, we need to pay in advance the cost of the usage of that service. Suppose we sign up a contract where we rent that service for 8 months. We would have to pay X (X being the cost of the service per month) at the beginning of each month, until the end of our contract. So, in our case, the total we would pay for the service would be X*8.

- Looking at the S2 service, if we signed a contract for 8 months, our total cost would mount up to 125*8 = 1000 euros. There's only one billable attribute for this service, which is the cost of the service itself per month. In order to model this using the API, we just needed to create a PricePlan with 1 PriceComponent. This PriceComponent was defined with a PriceFunction with the following mathematical expression: "standards2 * standards2months" where "standards2" is the price of the service per month and "standards2months" is the number of months that the customer will rent the service.

Arsys Example (4) - PricePlan

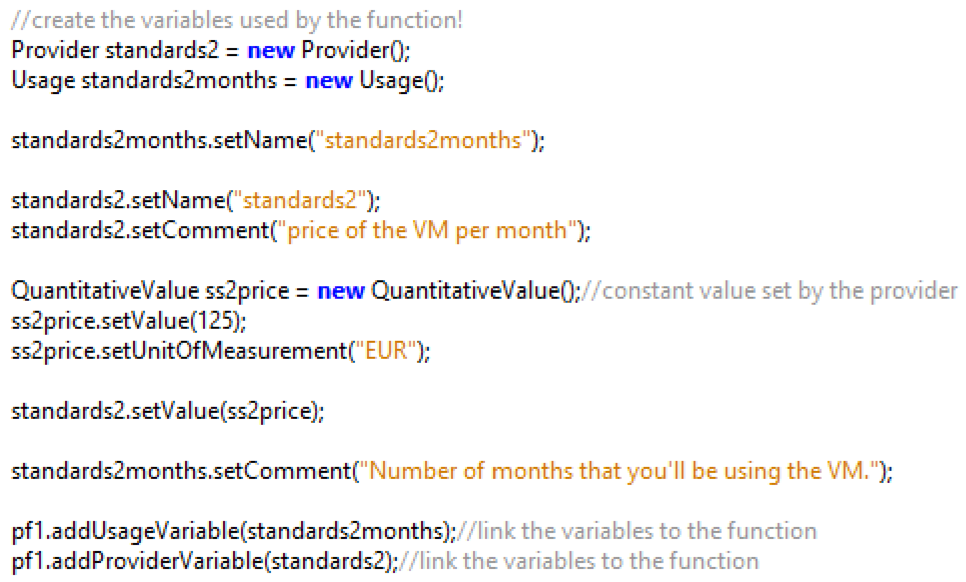
- This is the corresponding JAVA code:

(1)



```
////////////////////////////////////////////////////////////////
//elaborate the price plan of offering1
PricePlan pp1 = new PricePlan();
of1.setPricePlan(pp1);//link the priceplan to the offering

pp1.setComment("PricePlan of the Service1");
pp1.setName("PricePlan1");

//create the PriceComponents of the priceplan
PriceComponent pc1 = new PriceComponent();
pp1.addPriceComponent(pc1);//add the price component to the price plan
pc1.setComment("PriceComponent1 of the PricePlan1");

//create the price function of the component
PriceFunction pf1 = new PriceFunction();
pc1.setPriceFunction(pf1);//link the price function to the price component

pf1.setComment("PriceFunction of the PriceComponent1");
pf1.setStringFunction("standards2 * standards2months");//VM price * number of months that the VM will be used
```

(2)

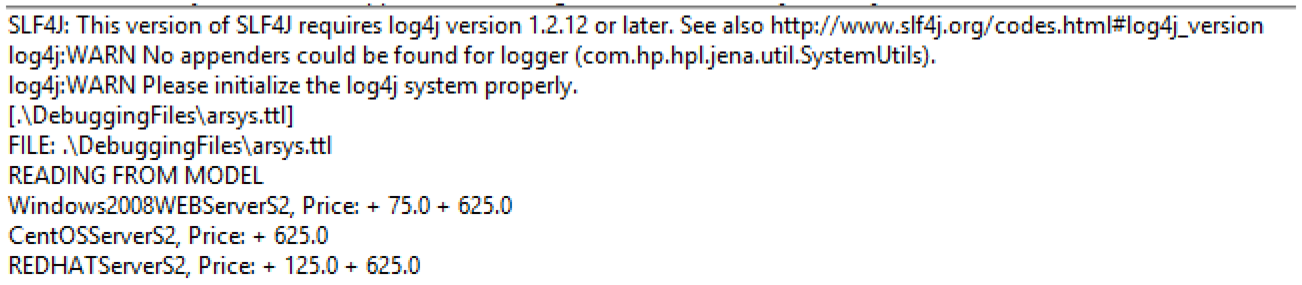


```
//create the variables used by the function!
Provider standards2 = new Provider();
Usage standards2months = new Usage();

standards2months.setName("standards2months");

standards2.setName("standards2");
standards2.setComment("price of the VM per month");

QuantitativeValue ss2price = new QuantitativeValue();//constant value set by the provider
ss2price.setValue(125);
ss2price.setUnitOfMeasurement("EUR");

standards2.setValue(ss2price);

standards2months.setComment("Number of months that you'll be using the VM.");

pf1.addUsageVariable(standards2months);//link the variables to the function
pf1.addProviderVariable(standards2);//link the variables to the function
```

- So far everything seems good. Once everything is set, we can call the writeToModel function in order to transform our JAVA Model into a Semantic Model:
  - ArrayList<Offering> offs = new ArrayList<Offering>();offs.add(of1); jmodel.setOfferings(offs);
  - jmodel.setBaseURI("http://PricingAPIArsysOfferings.com");
  - Model instance = jmodel.WriteToModel();//transform the java model to a semantic representation

Arsys Example (5)

- Once we have our semantic model, we can do whatever we want with it. We can import it into the TopBraid Composer for example or, we can **load it** with the **API**, define values for the Usage variables and then calculate the price of the Offerings based on the input from the user. On [4], I provide some examples that read the generated model, asks some values for the Usage variables of the model and then calculates the price of every offering.
- Running the ArsysReader Class and stating that we want to rent their service for 5months we get the following result:



```
SLF4J: This version of SLF4J requires log4j version 1.2.12 or later. See also http://www.slf4j.org/codes.html#log4j_version
log4j:WARN No appenders could be found for logger (com.hp.hpl.jena.util.SystemUtils).
log4j:WARN Please initialize the log4j system properly.
[.\DebuggingFiles\arsys.ttl]
FILE: .\DebuggingFiles\arsys.ttl
READING FROM MODEL
Windows2008WEBServerS2, Price: + 75.0 + 625.0
CentOSServerS2, Price: + 625.0
REDHATServerS2, Price: + 125.0 + 625.0
```

- As you can see, the ServerS2, which costs 125Euros per month, will have a total cost of 625Euros at the end of the 5months. It's also possible to see the cost of the same service but with other Operating Systems. While CentOS is free of charge, RedHat or Windows 2008 have an extra fee per month. For example, RedHat costs an extra 25Euros per month. 25*5 = 75Euros, as presented on the output.
- With this we can conclude that our API was successful modeling the Fully Bundled Recurrent PrePaid VM Access Model.
- PrePaid VM Access, the "brother" of the Recurrent PrePaid VM Access, has been replaced by other pricing methods (that also fall under the CODEX categories) so we were unable to model this type of pricing model.

Amazon Example

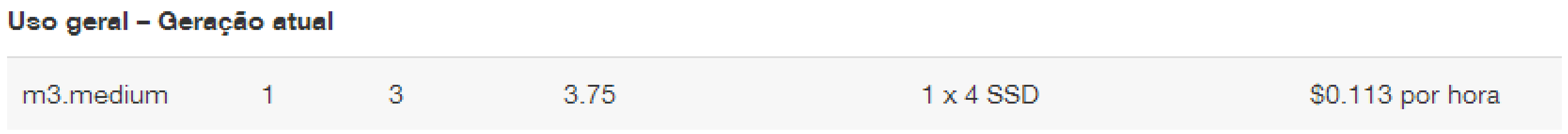
- Amazon EC2 was our next challenge, it provides instances with the following pricing methods:
  - On Demand
  - Reserved Instances
  - Spot Pricing

- As mentioned earlier, Amazon EC2 uses a VM Bundling approach on its services. As such, the customer has to choose the instance that suits his needs the best from the many available to him.

- For each of the pricing methods, a specific instance was chosen to be modeled using the API.

Amazon – On Demand

- The instance that I chose for this pricing method was the m3.medium for General Purposes instance, from North Virginia, running Linux.



Uso geral – Geração atual

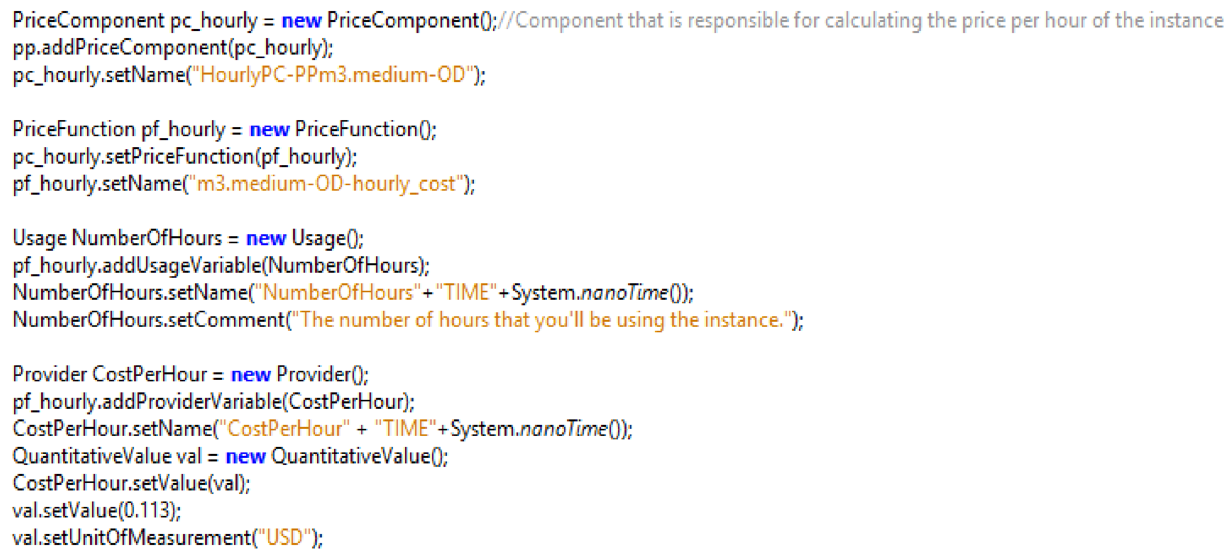
m3.medium	1	3	3.75	1 x 4 SSD	$0.113 por hora

- As explained earlier, On-Demand pricing method only charges users for what they use (usually at the end of the month). Aside from the cost of the instance, we have the extra fee for data transferals. Amazon EC2 only charges for outgoing traffic and, depending to where it is sent, different fee's apply however, in order to simplify the presentation, we only take into consideration traffic from AmazonEC2 to the internet since every other fee could be modeled as well using the same approach: adding more PriceComponents to the PricePlan.
- In order to calculate the total cost of the instance, we need to take into consideration the number of months that we'll be using the instance and the number of GB we expect to send out to the internet each month since these are the two billable components for the On-Demand Instances.

Amazon – On Demand (2)

- To model these two billable components (the cost per hour of the instance and the quantity of data sent to the internet) we need two PriceComponents for the PricePlan.

- One component will handle the cost of the Instance per hour while the other is in charge of calculating the total cost of data transfers:
  - PriceComponent – Hourly Cost:

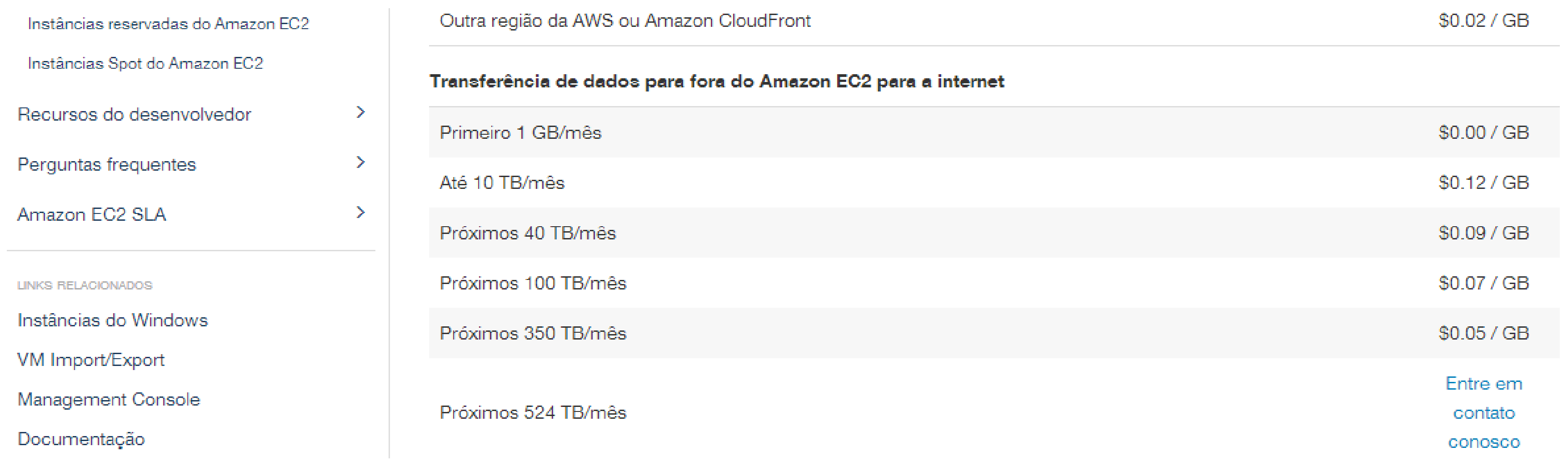


```
PriceComponent pc_hourly = new PriceComponent();//Component that is responsible for calculating the price per hour of the instance
pp.addPriceComponent(pc_hourly);
pc_hourly.setName("HourlyPC-PPm3.medium-OD");

PriceFunction pf_hourly = new PriceFunction();
pc_hourly.setPriceFunction(pf_hourly);
pf_hourly.setName("m3.medium-OD-hourly_cost");

Usage NumberOfHours = new Usage();
pf_hourly.addUsageVariable(NumberOfHours);
NumberOfHours.setName("NumberOfHours"+"TIME"+System.nanoTime());
NumberOfHours.setComment("The number of hours that you'll be using the instance.");

Provider CostPerHour = new Provider();
pf_hourly.addProviderVariable(CostPerHour);
CostPerHour.setName("CostPerHour" + "TIME"+System.nanoTime());
QuantitativeValue val = new QuantitativeValue();
CostPerHour.setValue(val);
val.setValue(0.113);
val.setUnitOfMeasurement("USD");
```

Amazon – On Demand (2)

- PriceComponent – Hourly Cost:

```
Provider price40 = new Provider();
data_cost_pf.addProviderVariable(price40);
price40.setName("price40"+"TIME"+System.nanoTime());
QuantitativeValue valc = new QuantitativeValue();
price40.setValue(valc);
valc.setValue(0.09);
valc.setUnitOfMeasurement("USD");

Provider price100 = new Provider();
data_cost_pf.addProviderVariable(price100);
price100.setName("price100"+"TIME"+System.nanoTime());
QuantitativeValue vald = new QuantitativeValue();
price100.setValue(vald);
vald.setValue(0.07);
vald.setUnitOfMeasurement("USD");

Provider price350 = new Provider();
data_cost_pf.addProviderVariable(price350);
price350.setName("price350"+"TIME"+System.nanoTime());
QuantitativeValue vale = new QuantitativeValue();
price350.setValue(vale);
vale.setValue(0.05);
vale.setUnitOfMeasurement("USD");

Usage gbout = new Usage();
data_cost_pf.addUsageVariable(gbout);
gbout.setName("gbout"+"TIME"+System.nanoTime());
gbout.setComment("Total GB of data that you expect to send out from Amazon EC2 to the internet.");
```



Instâncias reservadas do Amazon EC2
Instâncias Spot do Amazon EC2
Recursos do desenvolvedor
Perguntas frequentes
Amazon EC2 SLA

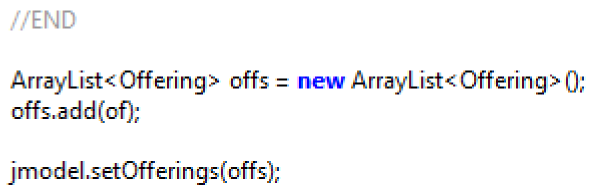
LINKS RELACIONADOS
Instâncias do Windows
VM Import/Export
Management Console
Documentação

Outra região da AWS ou Amazon CloudFront	$0.02 / GB
Transferência de dados para fora do Amazon EC2 para a internet	
Primeiro 1 GB/mês	$0.00 / GB
Até 10 TB/mês	$0.12 / GB
Próximos 40 TB/mês	$0.09 / GB
Próximos 100 TB/mês	$0.07 / GB
Próximos 350 TB/mês	$0.05 / GB
Próximos 524 TB/mês	Entre em contato conosco

```
data_cost_pf.setStringFunction("  IF ("+gbout.getName()+"<= 1) ; 1 * 0.00 ~"
    + " ELSEIF"+gbout.getName()+" > 1 && "+gbout.getName()+" <= 10*1024 ; 1*0.00 + ("+gbout.getName()+"-1) * "+price10.getName()+" ~ "
    + "ELSEIF ("+gbout.getName()+" > 10*1024) && ("+gbout.getName()+"<= 40*1024) ; 1*0.00 + 10*1024*"+price10.getName()+" + ("+gbout.getName()+"-10*1024-1)*"+price40.getName()+" ~ "
    + "ELSEIF ("+gbout.getName()+" >= 40*1024) && ("+gbout.getName()+" < 100*1024) ; 1*0.00 + 10*1024*"+price10.getName()+" + 40*1024*"+price40.getName()+" + ("+gbout.getName()+"-1-10*1024-40*1024)*"+price100.getName()+" ~ "
    + "ELSEIF ("+gbout.getName()+" >= 100*1024) && ("+gbout.getName()+" < 350*1024) ; 1*0.00 + 10*1024*"+price10.getName()+" + 40*1024*"+price40.getName()+" + 100*1024*"+price100.getName()+" + ("+gbout.getName()+"-1-10*1024-40*1024-100*1024)*"+price350.getName()+"");
```

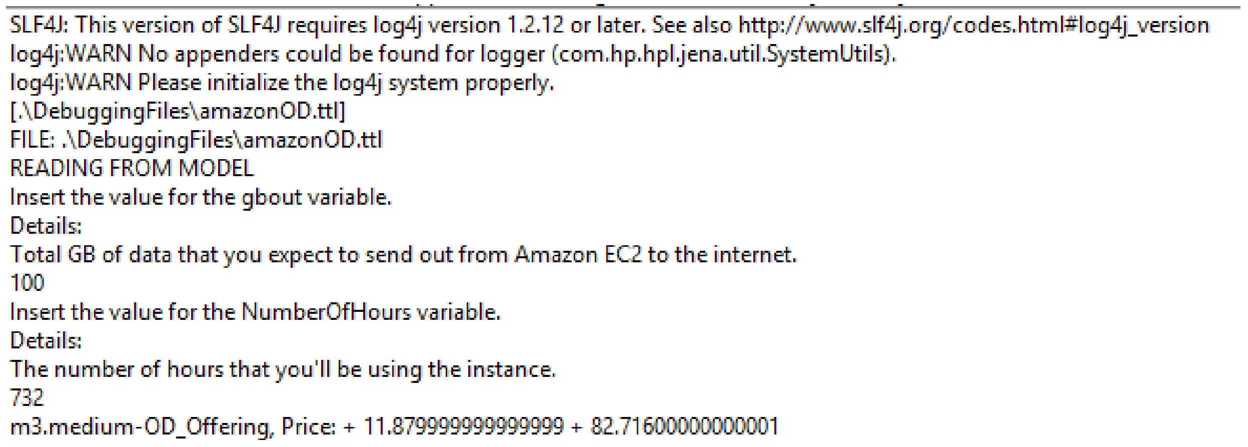
Amazon – On Demand (3)

- Once we're finished modeling the offering, like in the arsys example, we call the writeToModel() function to transform our JAVA model into a LinkedUSDL semantic model.



```
//END

ArrayList<Offering> offs = new ArrayList<Offering>();
offs.add(of);

jmodel.setOfferings(offs);
```

- Once we have our semantic model, we can test it with the AmazonODReader class. This class reads the model from a file, ask the user for the number of hours he's planning on using the instance and the number of GB he expects to send out, from the instance to the internet, per month. Then, it calculates and prints the total cost of the service.

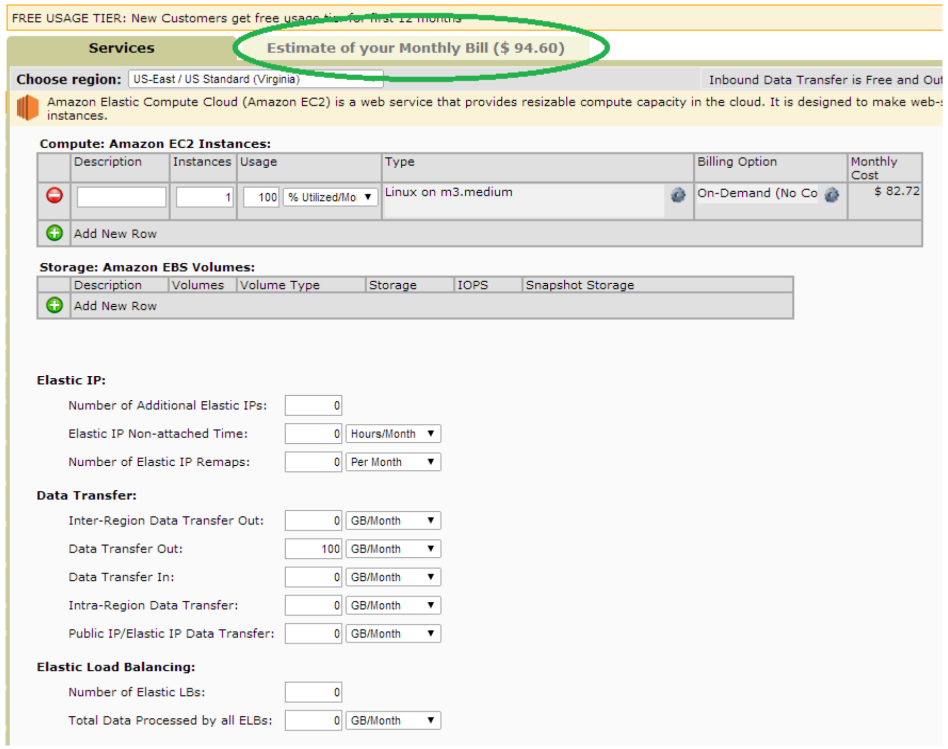


```
SLF4J: This version of SLF4J requires log4j version 1.2.12 or later. See also http://www.slf4j.org/codes.html#log4j_version
log4j:WARN No appenders could be found for logger (com.hp.hpl.jena.util.SystemUtils).
log4j:WARN Please initialize the log4j system properly.
[.\DebuggingFiles\amazonOD.ttl]
FILE: .\DebuggingFiles\amazonOD.ttl
READING FROM MODEL
Insert the value for the gbout variable.
Details:
Total GB of data that you expect to send out from Amazon EC2 to the internet.
100
Insert the value for the NumberOfHours variable.
Details:
The number of hours that you'll be using the instance.
732
m3.medium-OD_Offering, Price: + 11.879999999999999 + 82.71600000000001
```

Amazon – On Demand (3)

- On the previous image, we can see the two final costs for the two PriceComponents: 11.88$ for data transferals and 82.72$ for the 732hours we expect to use.

- After we use the m3.medium instance for 732 hours and send 100GB, we'll have to pay 94.60$. This result can be confirmed by amazon's calculator [8].



FREE USAGE TIER: New Customers get free usage tier for first 12 months

Services | Estimate of your Monthly Bill ($ 94.60)

Choose region: US-East / US Standard (Virginia) — Inbound Data Transfer is Free and Out

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-instances.

Compute: Amazon EC2 Instances:

	Description	Instances	Usage	Type	Billing Option	Monthly Cost
		1	100 % Utilized/Mo	Linux on m3.medium	On-Demand (No Co	$ 82.72
	Add New Row					

Storage: Amazon EBS Volumes:

	Description	Volumes	Volume Type	Storage	IOPS	Snapshot Storage
	Add New Row					

Elastic IP:

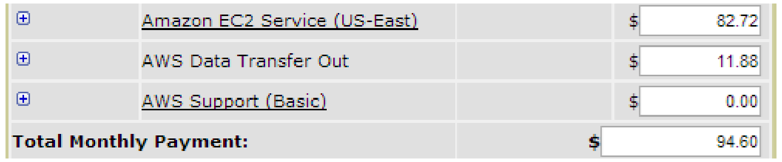
Number of Additional Elastic IPs:	0	
Elastic IP Non-attached Time:	0	Hours/Month
Number of Elastic IP Remaps:	0	Per Month

Data Transfer:

Inter-Region Data Transfer Out:	0	GB/Month
Data Transfer Out:	100	GB/Month
Data Transfer In:	0	GB/Month
Intra-Region Data Transfer:	0	GB/Month
Public IP/Elastic IP Data Transfer:	0	GB/Month

Elastic Load Balancing:

Number of Elastic LBs:	0	
Total Data Processed by all ELBs:	0	GB/Month

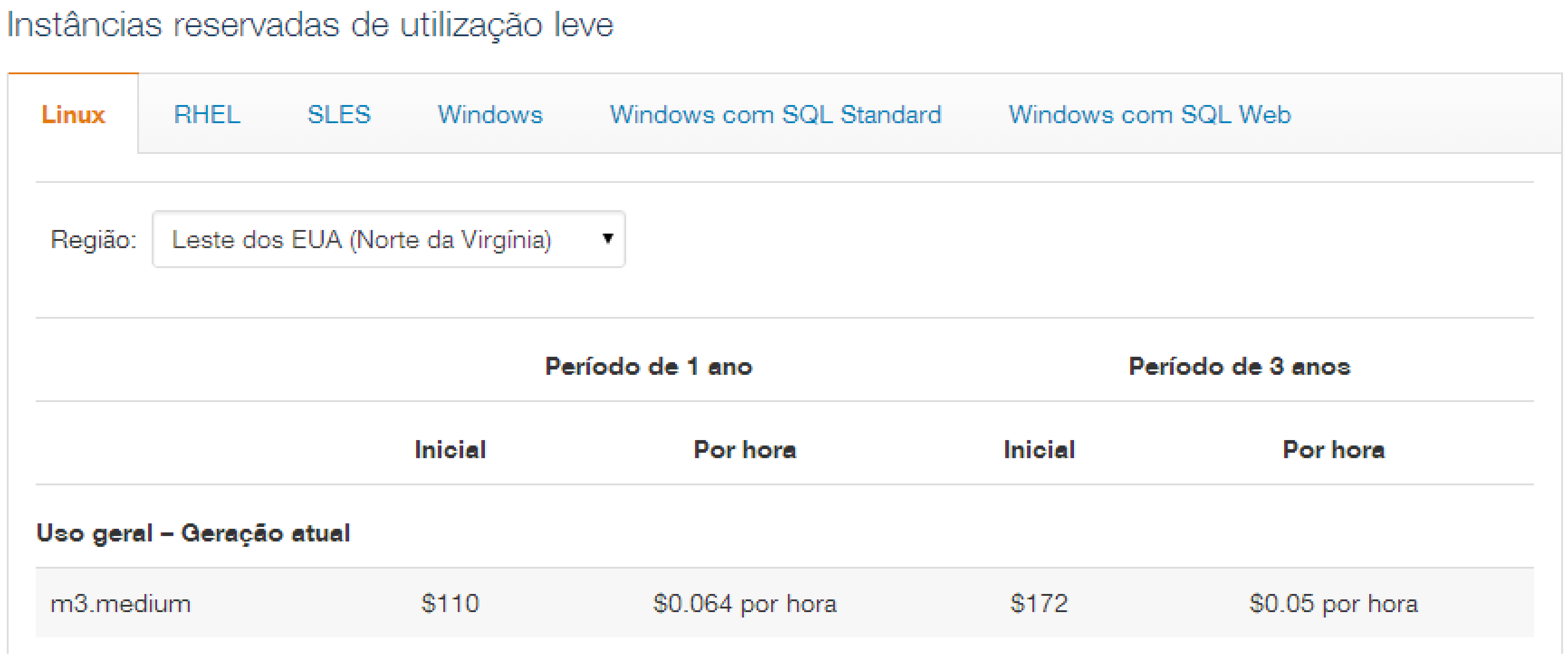


Amazon EC2 Service (US-East)	$ 82.72
AWS Data Transfer Out	$ 11.88
AWS Support (Basic)	$ 0.00
Total Monthly Payment:	$ 94.60

[8] http://calculator.s3.amazonaws.com/index.html

Amazon – Reserved Instance

- Amazon's EC2 Reserved Instance is very similar to their On-Demand instances but, require an up-front payment from the customer. This up-front payment provides a discount rate on the price of the service over a certain period of time.
- To model this pricing method I chose the m3.medium light usage with a 1 year subscription.

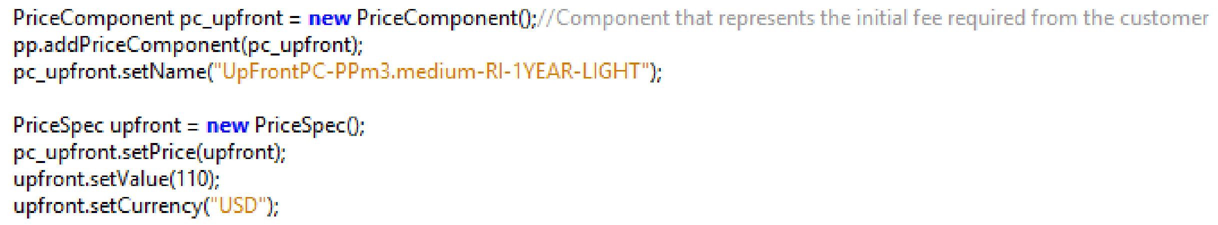

Instâncias reservadas de utilização leve

Região: Leste dos EUA (Norte da Virgínia)

	Período de 1 ano		Período de 3 anos	
	Inicial	Por hora	Inicial	Por hora
Uso geral – Geração atual				
m3.medium	$110	$0.064 por hora	$172	$0.05 por hora

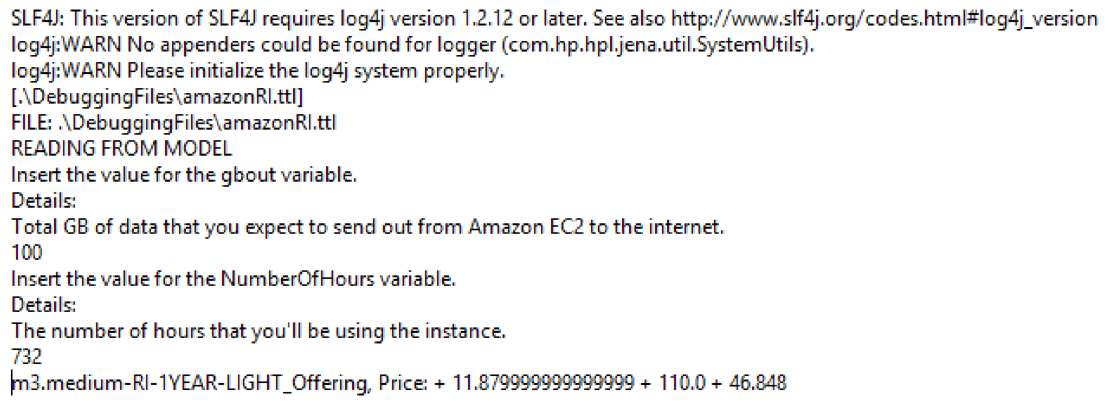
Amazon – Reserved Instance (2)

- As you can see, we need to pay 110$ up-front in order to use the m3.medium instance.
- To model this extra billable component of the service, a new PriceComponent with a static value was created:



```
PriceComponent pc_upfront = new PriceComponent();//Component that represents the initial fee required from the customer
pp.addPriceComponent(pc_upfront);
pc_upfront.setName("UpFrontPC-PPm3.medium-RI-1YEAR-LIGHT");

PriceSpec upfront = new PriceSpec();
pc_upfront.setPrice(upfront);
upfront.setValue(110);
upfront.setCurrency("USD");
```

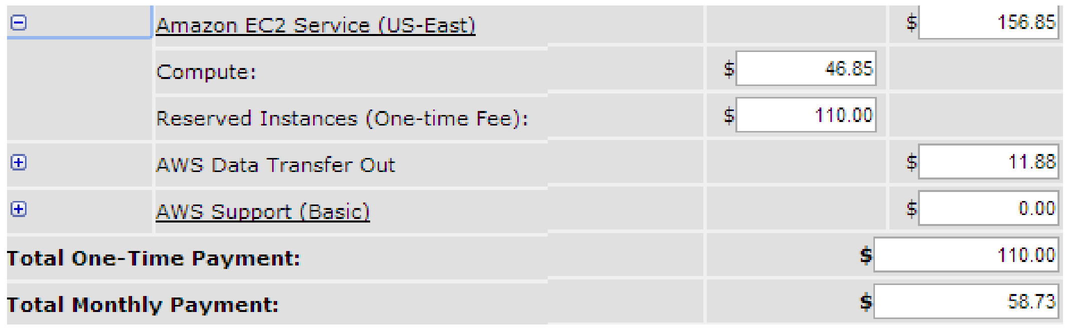
- In total, there are 3 PriceComponents required for this example: 1 to calculate the cost of the service per hour, another to calculate the cost of the data transfers and another to model the required up-front payment.
- Assuming we want to rent this service for 1 month (732Hrs) and expect to send out 100GB, the total cost would be: 110$ initial fee + 0.064$*732Hrs + Data transfer costs. Running the AmazonRIReader class and inserting the corresponding values we get the following output:



```
SLF4J: This version of SLF4J requires log4j version 1.2.12 or later. See also http://www.slf4j.org/codes.html#log4j_version
log4j:WARN No appenders could be found for logger (com.hp.hpl.jena.util.SystemUtils).
log4j:WARN Please initialize the log4j system properly.
[.\DebuggingFiles\amazonRI.ttl]
FILE: .\DebuggingFiles\amazonRI.ttl
READING FROM MODEL
Insert the value for the gbout variable.
Details:
Total GB of data that you expect to send out from Amazon EC2 to the internet.
100
Insert the value for the NumberOfHours variable.
Details:
The number of hours that you'll be using the instance.
732
m3.medium-RI-1YEAR-LIGHT_Offering, Price: + 11.879999999999999 + 110.0 + 46.848
```

Amazon – Reserved Instance (3)

- This result is validated using Amazon's web calculator [8].



⊟ Amazon EC2 Service (US-East)			$ 156.85
	Compute:	$ 46.85	
	Reserved Instances (One-time Fee):	$ 110.00	
⊞ AWS Data Transfer Out			$ 11.88
⊞ AWS Support (Basic)			$ 0.00
Total One-Time Payment:			**$** 110.00
Total Monthly Payment:			**$** 58.73

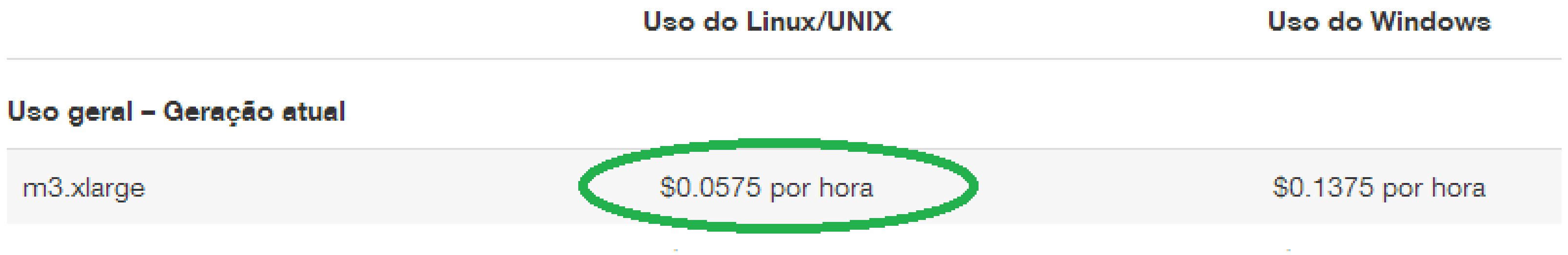
- Adding up the components: 46.85$ + 110$ + 11.88$ = 168.73$ which is the result returned by the AmazonRIReader class. Of course, I ran the example with one month (732hrs) because I wanted to confirm the result with the one obtained by Amazon.

Amazon – Spot instance

- Amazon's EC2 Spot Instances are the same as their On-Demand instances but, with a slight difference. While their On-Demand instances have "fixed" fee's (their price doesn't change that often when compare with Spot instances), Spot Instances have a variable fee which may depend on the current state of the market or the current state of their cloud platform. This means that the price I may be paying now may not be the same in 2 or 3 days from now.

- To tackle this issue, I combined the usage of our API with a little of JSON *parsing*. I use the API to model the service's data and fetch the price of the instance directly from a JSON file provided by Amazon: http://aws-assets-pricing-prod.s3.amazonaws.com/pricing/ec2/spot.js

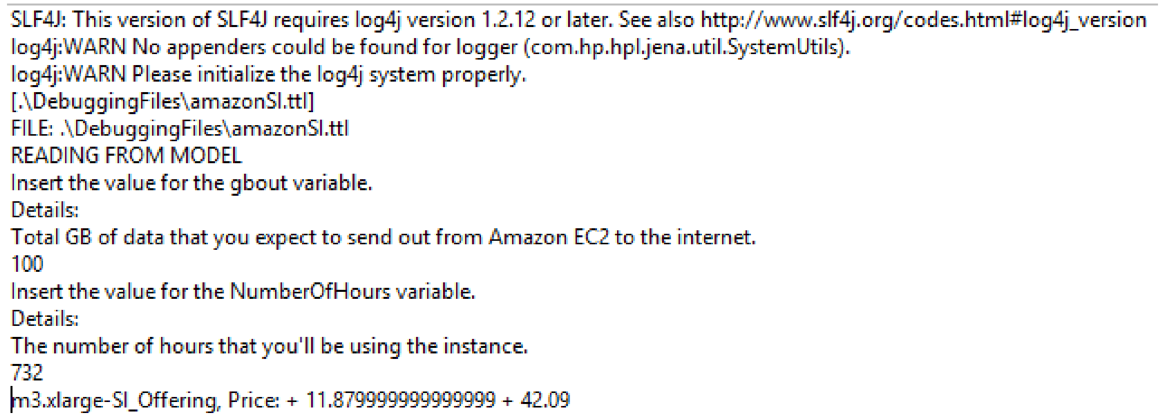
- To model this pricing method I chose the m3.xlarge instance, running Linux, located on North Virginia.



	Uso do Linux/UNIX	Uso do Windows
Uso geral – Geração atual		
m3.xlarge	$0.0575 por hora	$0.1375 por hora

Amazon – Spot Instance(2)

- Assuming we want to rent this service for 1 month (732Hrs) and expect to send out 100GB, the total cost would be: 0.0575$*732Hrs + Data transfer costs. Running the AmazonRIReader class and inserting the corresponding values we get the following output:



```
SLF4J: This version of SLF4J requires log4j version 1.2.12 or later. See also http://www.slf4j.org/codes.html#log4j_version
log4j:WARN No appenders could be found for logger (com.hp.hpl.jena.util.SystemUtils).
log4j:WARN Please initialize the log4j system properly.
[.\DebuggingFiles\amazonSI.ttl]
FILE: .\DebuggingFiles\amazonSI.ttl
READING FROM MODEL
Insert the value for the gbout variable.
Details:
Total GB of data that you expect to send out from Amazon EC2 to the internet.
100
Insert the value for the NumberOfHours variable.
Details:
The number of hours that you'll be using the instance.
732
m3.xlarge-SI_Offering, Price: + 11.879999999999999 + 42.09
```

- Unfortunately, Amazon's calculator doesn't calculate the price for their Spot instances but the result can be confirmed manually performing some simple calculations.

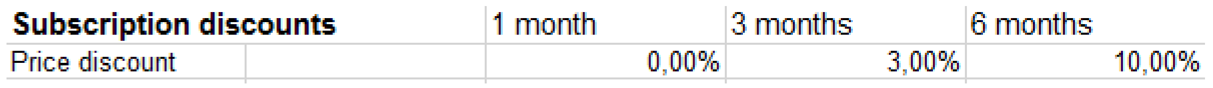
CloudSigma – PrePaid Credit Subscription Plans

- As presented on Table.1, their services use (but not only) an Unbundled PrePaid Credit Subscription pricing method. Very similar to the PrePaid Credit Plan but instead of paying everything upfront we're able to pay the service in arrears over a certain period of time.

- The subscription plans are only available for the following durations:
  - 1Month, 3Months, 6Months, 12 Months, 24Months and 36Months

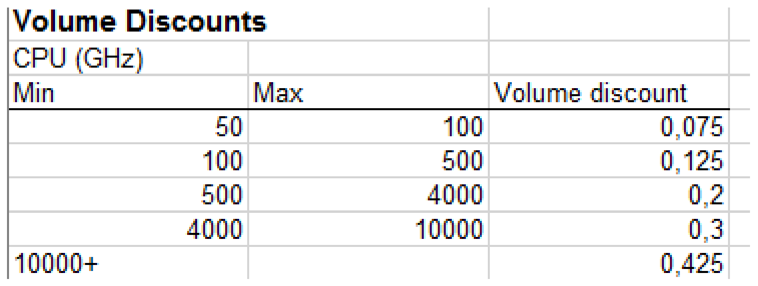
- They apply two types of discounts:
  1. The total price is reduced by a certain % depending on the number of months the customer is willing to subscribe. The longer the subscription plan is, the higher the discount.
  2. Depending on the quantity of purchased resources (CPU,RAM and DiskSize), the total price of that resource is reduced by a certain % as well.

1. Example – Subscription discount



Subscription discounts		1 month	3 months	6 months
Price discount		0,00%	3,00%	10,00%

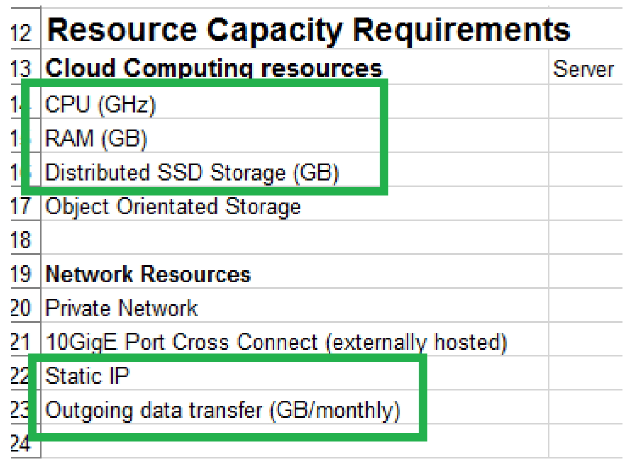
2. Example – CPU discount



Volume Discounts		
CPU (GHz)		
Min	Max	Volume discount
50	100	0,075
100	500	0,125
500	4000	0,2
4000	10000	0,3
10000+		0,425

CloudSigma–PrePaid Credit Subscription Plans (2)

- Given this information, I decided to model the 6Months subscription plan.
- Since CloudSigma already pre-defines the number of months (6), we don't need to ask the user how many months he desires to use the service.
- Due to the Unbundled approach, the PricePlan will have more PriceComponents than the examples presented so far since each of the attributes is billed separately, contrary to the other bundling methods where some of the attributes are combined into a single billable unit.
- After identifying the billable attributes of the service, I modeled some of them to test the API:
  - CPU Speed
  - Quantity RAM
  - Quantity Disk Storage
  - Number of Static IP
  - GB of outgoing data
- Every other billable component could be modeled like these.

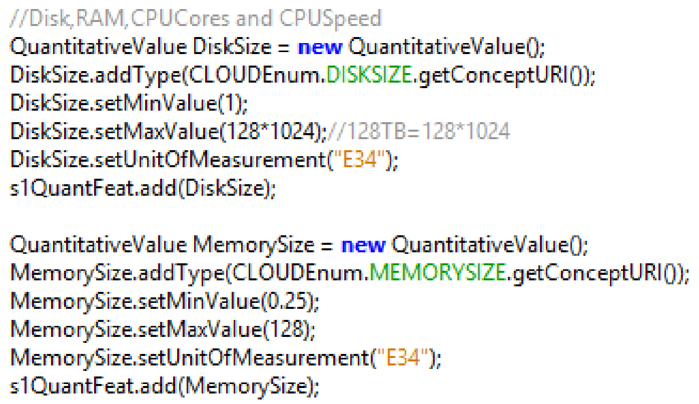


12	**Resource Capacity Requirements**	
13	**Cloud Computing resources**	Server
14	CPU (GHz)	
15	RAM (GB)	
16	Distributed SSD Storage (GB)	
17	Object Orientated Storage	
18		
19	**Network Resources**	
20	Private Network	
21	10GigE Port Cross Connect (externally hosted)	
22	Static IP	
23	Outgoing data transfer (GB/monthly)	
24		

CloudSigma–PrePaid Credit Subscription Plans (3)

- In total, there are 5 modeled billable components for the service.

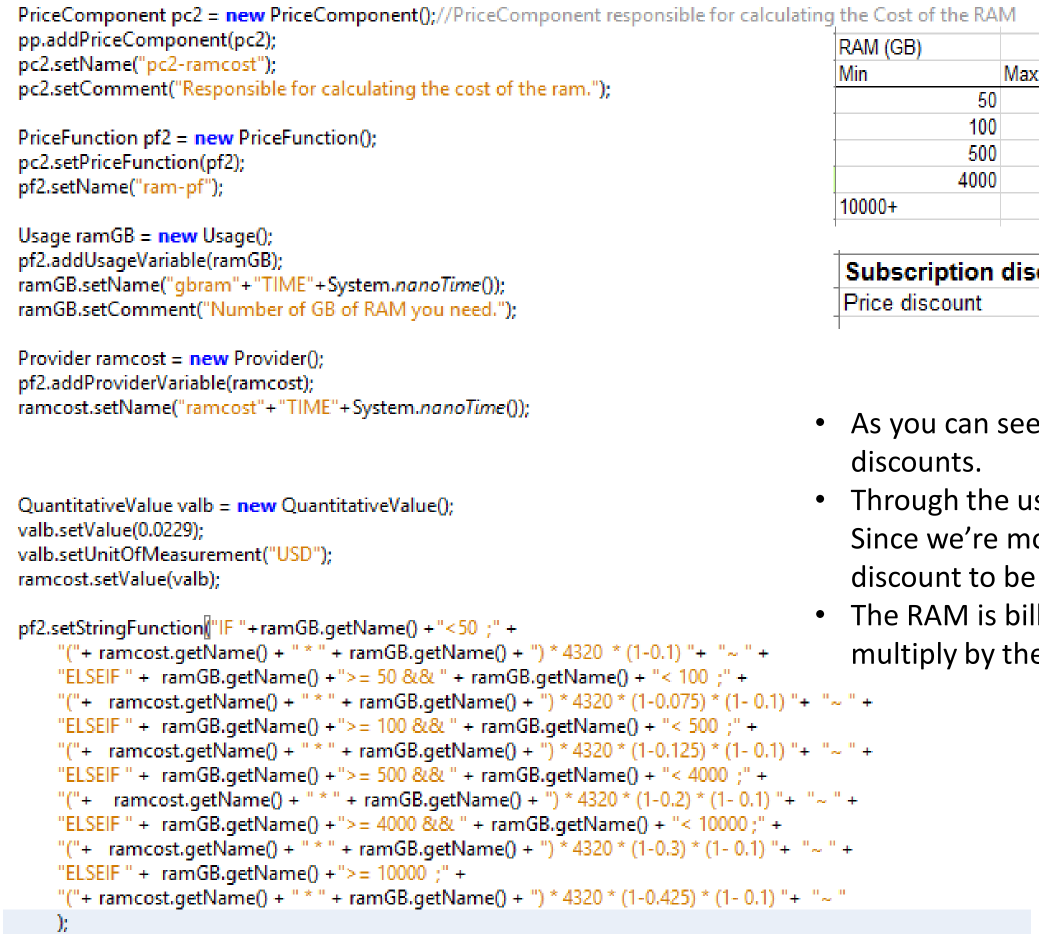
- Since there are no pre-defined values for these attributes of the service, only their Maximum and Minimum values, their features had to be modeled using different properties of the QuantitativeValue class:
    - setMinValue()
    - setMaxValue()



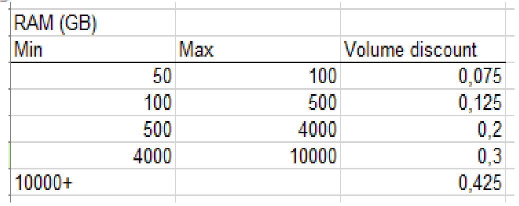
```
//Disk,RAM,CPUCores and CPUSpeed
QuantitativeValue DiskSize = new QuantitativeValue();
DiskSize.addType(CLOUDEnum.DISKSIZE.getConceptURI());
DiskSize.setMinValue(1);
DiskSize.setMaxValue(128*1024);//128TB=128*1024
DiskSize.setUnitOfMeasurement("E34");
s1QuantFeat.add(DiskSize);

QuantitativeValue MemorySize = new QuantitativeValue();
MemorySize.addType(CLOUDEnum.MEMORYSIZE.getConceptURI());
MemorySize.setMinValue(0.25);
MemorySize.setMaxValue(128);
MemorySize.setUnitOfMeasurement("E34");
s1QuantFeat.add(MemorySize);
```

CloudSigma–PrePaid Credit Subscription Plans (4)

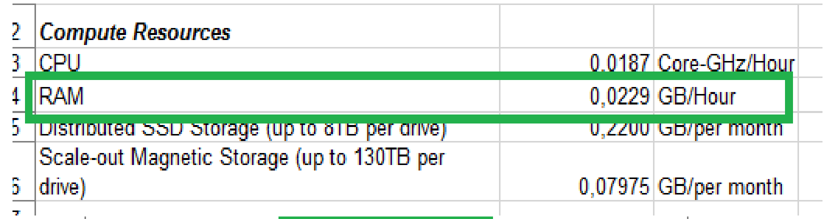
- Let’s take a look at the PriceComponent that defines the cost of the Memory attribute:



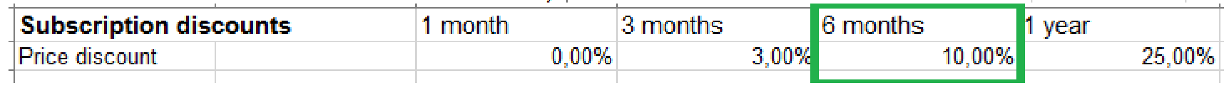
```
PriceComponent pc2 = new PriceComponent();//PriceComponent responsible for calculating the Cost of the RAM
pp.addPriceComponent(pc2);
pc2.setName("pc2-ramcost");
pc2.setComment("Responsible for calculating the cost of the ram.");

PriceFunction pf2 = new PriceFunction();
pc2.setPriceFunction(pf2);
pf2.setName("ram-pf");

Usage ramGB = new Usage();
pf2.addUsageVariable(ramGB);
ramGB.setName("gbram"+"TIME"+System.nanoTime());
ramGB.setComment("Number of GB of RAM you need.");

Provider ramcost = new Provider();
pf2.addProviderVariable(ramcost);
ramcost.setName("ramcost"+"TIME"+System.nanoTime());



QuantitativeValue valb = new QuantitativeValue();
valb.setValue(0.0229);
valb.setUnitOfMeasurement("USD");
ramcost.setValue(valb);

pf2.setStringFunction("IF "+ramGB.getName() +"<50 ;" +
     "("+ ramcost.getName() + " * " + ramGB.getName() + ") * 4320  * (1-0.1) "+  "~ " +
     "ELSEIF " +  ramGB.getName() +">= 50 && " + ramGB.getName() + "< 100 ;" +
     "("+   ramcost.getName() + " * " + ramGB.getName() + ") * 4320 * (1-0.075) * (1- 0.1) "+  "~ " +
     "ELSEIF " +  ramGB.getName() +">= 100 && " + ramGB.getName() + "< 500  ;" +
     "("+   ramcost.getName() + " * " + ramGB.getName() + ") * 4320 * (1-0.125) * (1- 0.1) "+  "~ " +
     "ELSEIF " +  ramGB.getName() +">= 500 && " + ramGB.getName() + "< 4000  ;" +
     "("+    ramcost.getName() + " * " + ramGB.getName() + ") * 4320 * (1-0.2) * (1- 0.1) "+  "~ " +
     "ELSEIF " +  ramGB.getName() +">= 4000 && " + ramGB.getName() + "< 10000 ;" +
     "("+   ramcost.getName() + " * " + ramGB.getName() + ") * 4320 * (1-0.3) * (1- 0.1) "+  "~ " +
     "ELSEIF " +  ramGB.getName() +">= 10000  ;" +
     "("+ ramcost.getName() + " * " + ramGB.getName() + ") * 4320 * (1-0.425) * (1- 0.1) "+  "~ "
     );
```



RAM (GB)		
Min	Max	Volume discount
50	100	0,075
100	500	0,125
500	4000	0,2
4000	10000	0,3
10000+		0,425



Compute Resources		
CPU	0,0187	Core-GHz/Hour
RAM	0,0229	GB/Hour
Distributed SSD Storage (up to 8TB per drive)	0,2200	GB/per month
Scale-out Magnetic Storage (up to 130TB per drive)	0,07975	GB/per month



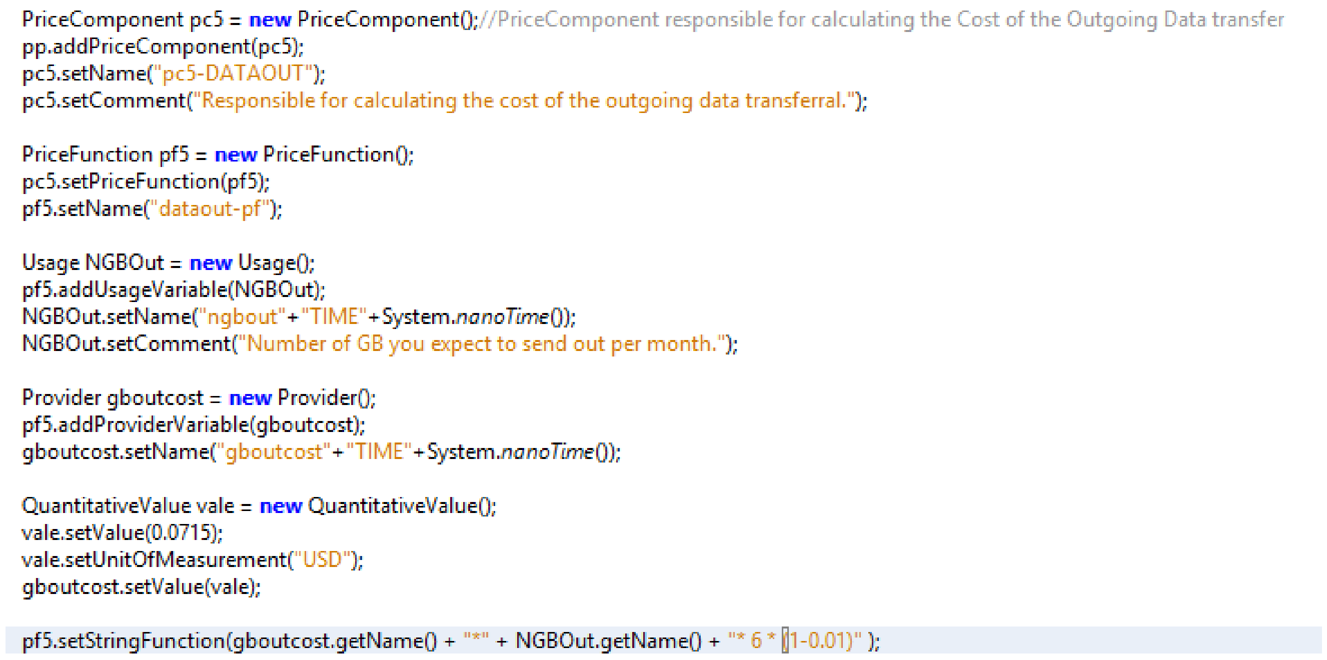
Subscription discounts	1 month	3 months	6 months	1 year
Price discount	0,00%	3,00%	10,00%	25,00%

- As you can see, the mathematical expression to calculate the cost of the Memory Size applies both discounts.
- Through the usage of IF-ELSEIF-ELSE conditions, we can calculate the correct price for the customer. Since we’re modeling the 6Months Subscription, the 10% discount is always applied. The quantity discount to be applied depends on the quantity of RAM requested by the customer.
- The RAM is billed by the hour thus, instead of multiplying by the number of months, we need to multiply by the number of hours in 6 months.

CloudSigma–PrePaid Credit Subscription Plans (5)

- This process is repeated for the other billable attributes whose cost is influenced by the quantity of purchased resources.

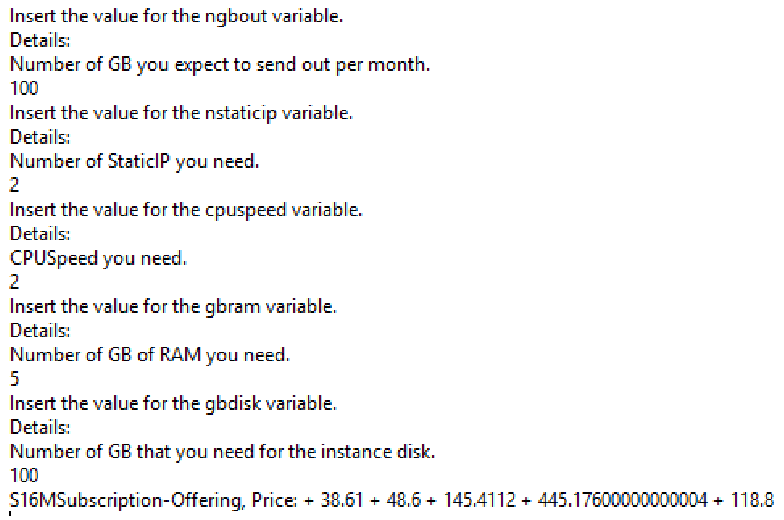
- Billable attributes that are not affected by the quantity discount have a much simpler mathematical expression where we simply multiply the fee of the attribute by the value inserted and apply the subscription discount :



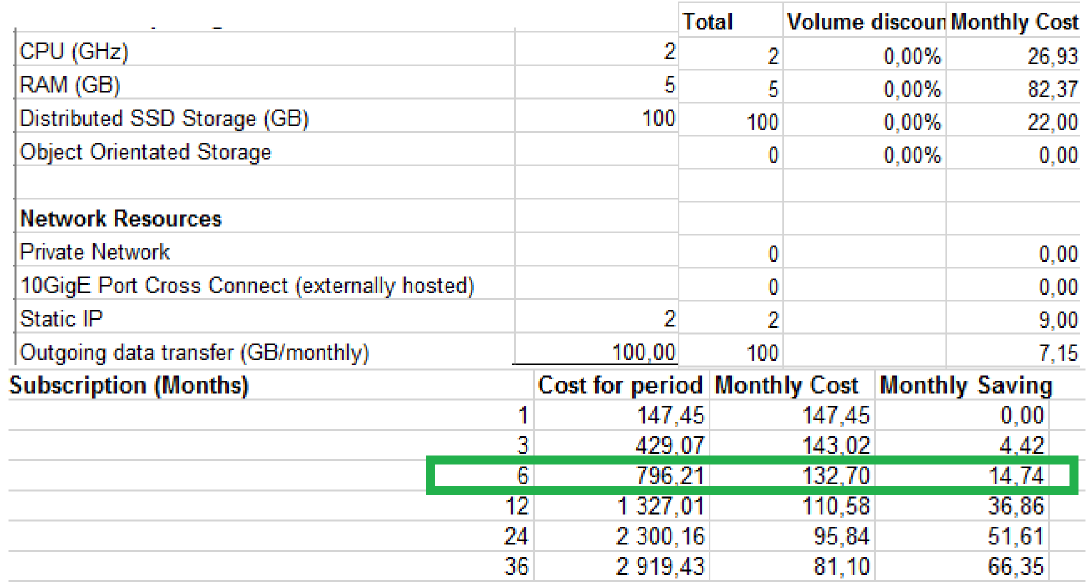
```java
PriceComponent pc5 = new PriceComponent();//PriceComponent responsible for calculating the Cost of the Outgoing Data transfer
pp.addPriceComponent(pc5);
pc5.setName("pc5-DATAOUT");
pc5.setComment("Responsible for calculating the cost of the outgoing data transferral.");

PriceFunction pf5 = new PriceFunction();
pc5.setPriceFunction(pf5);
pf5.setName("dataout-pf");

Usage NGBOut = new Usage();
pf5.addUsageVariable(NGBOut);
NGBOut.setName("ngbout"+"TIME"+System.nanoTime());
NGBOut.setComment("Number of GB you expect to send out per month.");

Provider gboutcost = new Provider();
pf5.addProviderVariable(gboutcost);
gboutcost.setName("gboutcost"+"TIME"+System.nanoTime());

QuantitativeValue vale = new QuantitativeValue();
vale.setValue(0.0715);
vale.setUnitOfMeasurement("USD");
gboutcost.setValue(vale);

pf5.setStringFunction(gboutcost.getName() + "*" + NGBOut.getName() + "* 6 * (1-0.01)" );
```

CloudSigma–PrePaid Credit Subscription Plans (6)

- When everything has been set, we can invoke the writeToModel function in order to create our Semantic model.
- With the CloudSigmaReader we can load and test the created model:



```
Insert the value for the ngbout variable.
Details:
Number of GB you expect to send out per month.
100
Insert the value for the nstaticip variable.
Details:
Number of StaticIP you need.
2
Insert the value for the cpuspeed variable.
Details:
CPUSpeed you need.
2
Insert the value for the gbram variable.
Details:
Number of GB of RAM you need.
5
Insert the value for the gbdisk variable.
Details:
Number of GB that you need for the instance disk.
100
S16MSubscription-Offering, Price: + 38.61 + 48.6 + 145.4112 + 445.17600000000004 + 118.8
```



		Total	Volume discoun	Monthly Cost
CPU (GHz)	2	2	0,00%	26,93
RAM (GB)	5	5	0,00%	82,37
Distributed SSD Storage (GB)	100	100	0,00%	22,00
Object Orientated Storage		0	0,00%	0,00
Network Resources				
Private Network		0		0,00
10GigE Port Cross Connect (externally hosted)		0		0,00
Static IP	2	2		9,00
Outgoing data transfer (GB/monthly)	100,00	100		7,15

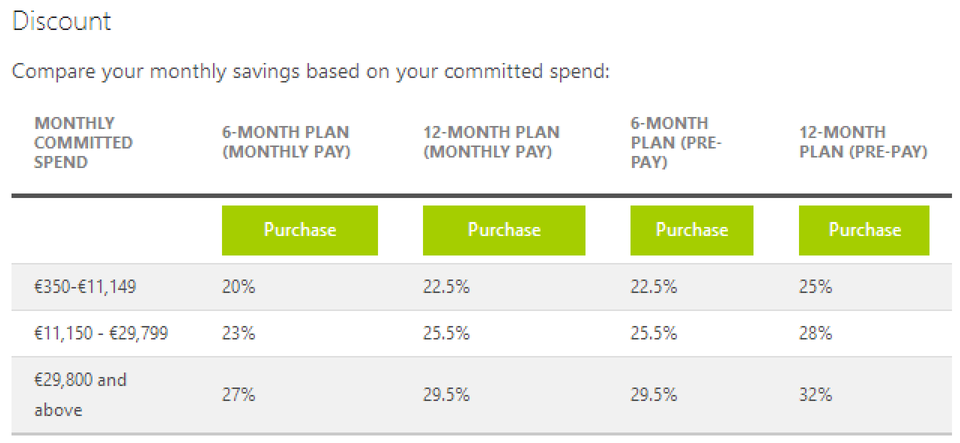
Subscription (Months)	Cost for period	Monthly Cost	Monthly Saving
1	147,45	147,45	0,00
3	429,07	143,02	4,42
6	796,21	132,70	14,74
12	1 327,01	110,58	36,86
24	2 300,16	95,84	51,61
36	2 919,43	81,10	66,35

- Adding the 5 pricing values calculated by the PriceComponents, we can verify that we reach the same result (0.4 error) as the one obtained by the subscription pricing sheet provided by their customer support: 38.61 + 48.6 + 145.41 + 445.18 + 118.8 = 796.6 USD, or 132.77 USD per month for 6 months.

Microsoft Azure – PrePaid Credit Plan

- As presented on Table.1, their services use (but not only) a VB Bundled PrePaid Credit Plan pricing method where the user has to pay upfront the full cost of the service. The upfront payment is converted into credits from which costs of usage will be deducted from. In case the customer doesn't use the expected quota of resources that month, the credits slide over to the next month until the end of the term. Once the term is over, the leftover credits become invalid and can't be consumed.

- Like CloudSigma, Microsoft also pre-defines the duration of the renting term:
  - 6Months or 12Months.
- They apply discounts based on the total cost of the service you're renting:



Discount

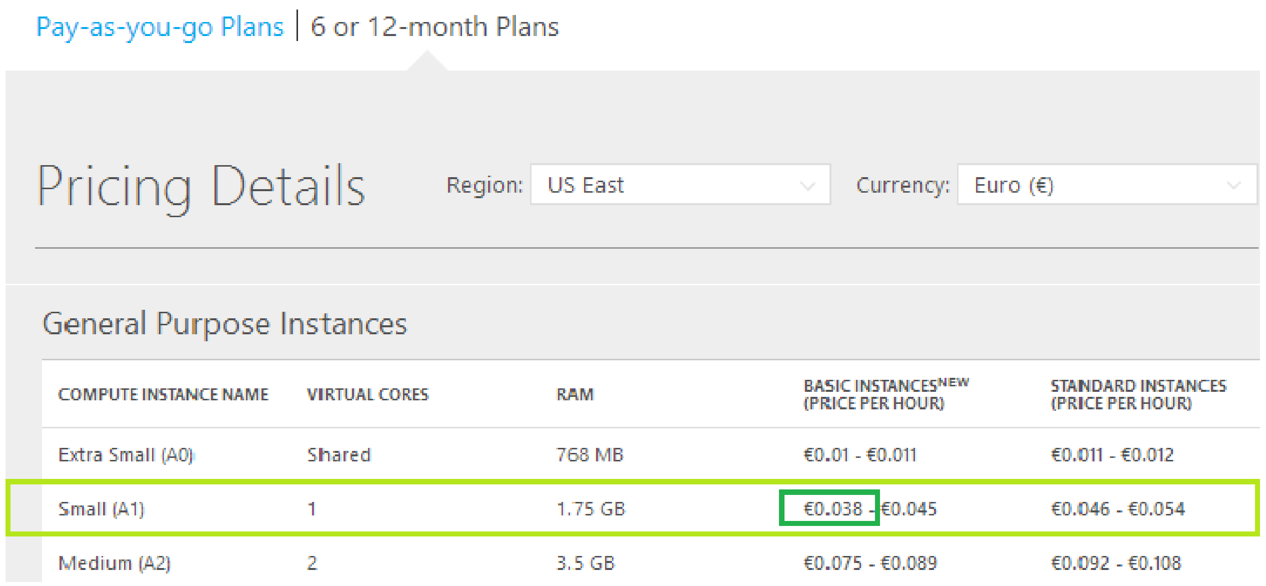
Compare your monthly savings based on your committed spend:

MONTHLY COMMITTED SPEND	6-MONTH PLAN (MONTHLY PAY)	12-MONTH PLAN (MONTHLY PAY)	6-MONTH PLAN (PRE-PAY)	12-MONTH PLAN (PRE-PAY)
	Purchase	Purchase	Purchase	Purchase
€350-€11,149	20%	22.5%	22.5%	25%
€11,150 - €29,799	23%	25.5%	25.5%	28%
€29,800 and above	27%	29.5%	29.5%	32%

http://azure.microsoft.com/en-us/offers/commitment-plans/

Microsoft Azure – PrePaid Credit Plan (2)

- To model this pricing method I chose Microsoft's A1 (Small) Virtual Machine on a 6Months pre-paid plan.



Pay-as-you-go Plans | 6 or 12-month Plans

Pricing Details

Region: US East

Currency: Euro (€)

General Purpose Instances

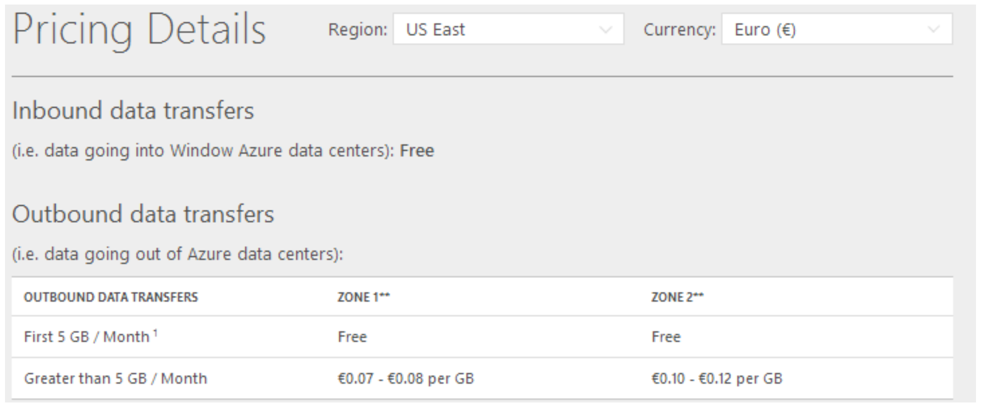
COMPUTE INSTANCE NAME	VIRTUAL CORES	RAM	BASIC INSTANCES NEW (PRICE PER HOUR)	STANDARD INSTANCES (PRICE PER HOUR)
Extra Small (A0)	Shared	768 MB	€0.01 - €0.011	€0.011 - €0.012
Small (A1)	1	1.75 GB	€0.038 - €0.045	€0.046 - €0.054
Medium (A2)	2	3.5 GB	€0.075 - €0.089	€0.092 - €0.108

http://azure.microsoft.com/en-us/pricing/details/virtual-machines/

- Since they adopted a VB Bundling method, the PricePlan of the offering will have two main PriceComponents: One for the cost of the VM per hour and another to calculate the cost of data transferals. For this example however, I decided to include the cost (and discount) of the backup feature.
- In total, there's 3 billable attributes in this example: VM cost, Data Transfers and Back-up. Each of these has a discount associated with it.

Microsoft Azure – PrePaid Credit Plan (3)

- PriceComponents and Discounts:
  - The total cost of the service has a discount which is presented on the image of slide 36.
  - Data Transferrals: The first 5GB of each month are free.
  - Back-up Costs: The first 5GB of each month are free.



Pricing Details Region: US East Currency: Euro (€)

Inbound data transfers

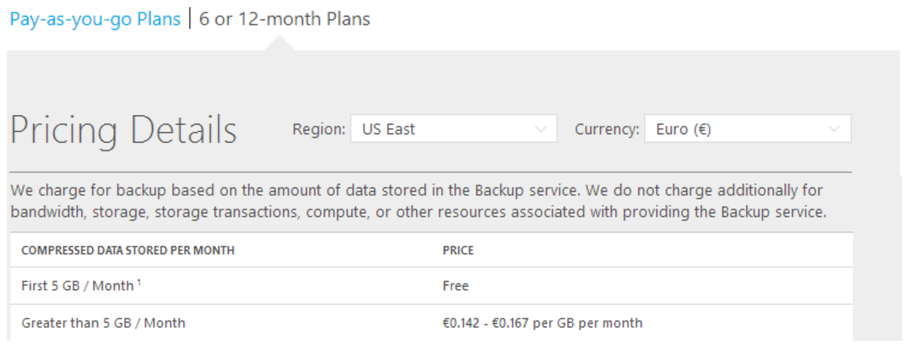
(i.e. data going into Window Azure data centers): **Free**

Outbound data transfers

(i.e. data going out of Azure data centers):

OUTBOUND DATA TRANSFERS	ZONE 1**	ZONE 2**
First 5 GB / Month [1]	Free	Free
Greater than 5 GB / Month	€0.07 - €0.08 per GB	€0.10 - €0.12 per GB

http://azure.microsoft.com/en-us/pricing/details/data-transfers/



Pay-as-you-go Plans | 6 or 12-month Plans

Pricing Details Region: US East Currency: Euro (€)

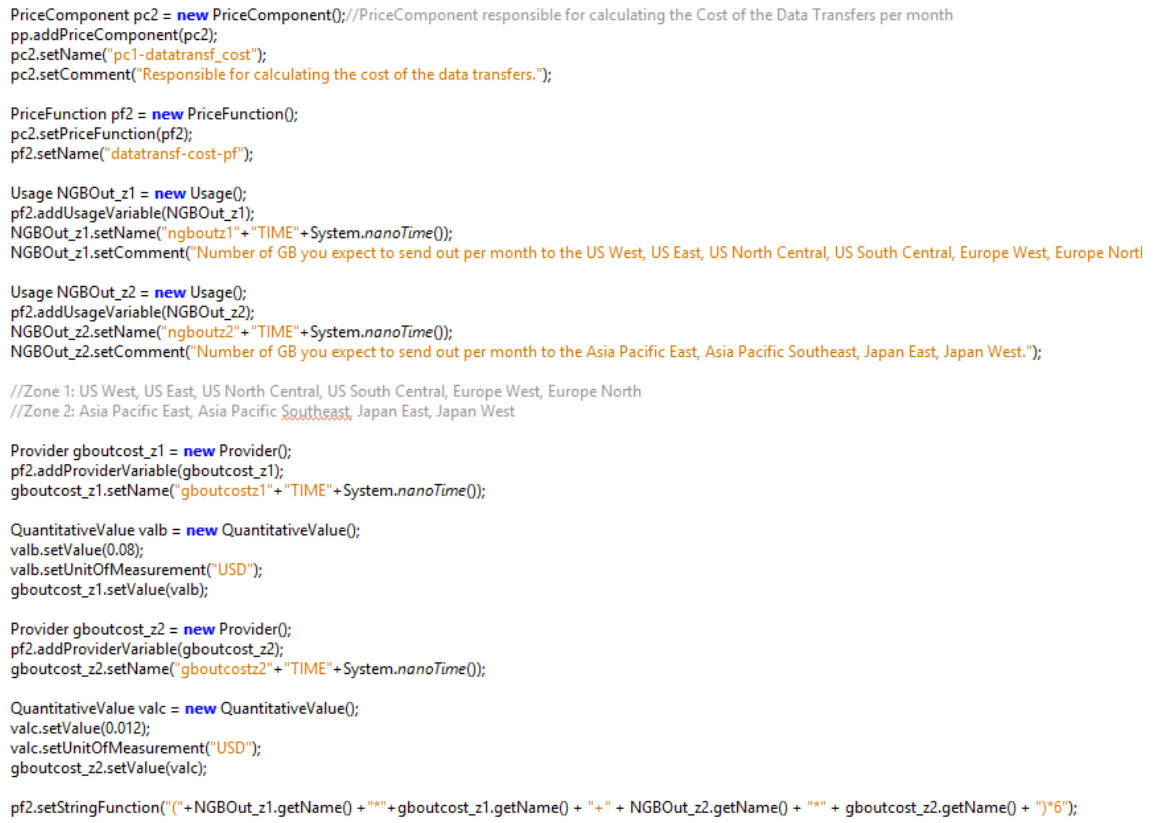
We charge for backup based on the amount of data stored in the Backup service. We do not charge additionally for bandwidth, storage, storage transactions, compute, or other resources associated with providing the Backup service.

COMPRESSED DATA STORED PER MONTH	PRICE
First 5 GB / Month [1]	Free
Greater than 5 GB / Month	€0.142 - €0.167 per GB per month

http://azure.microsoft.com/en-us/pricing/details/backup/

Microsoft Azure – PrePaid Credit Plan (4)

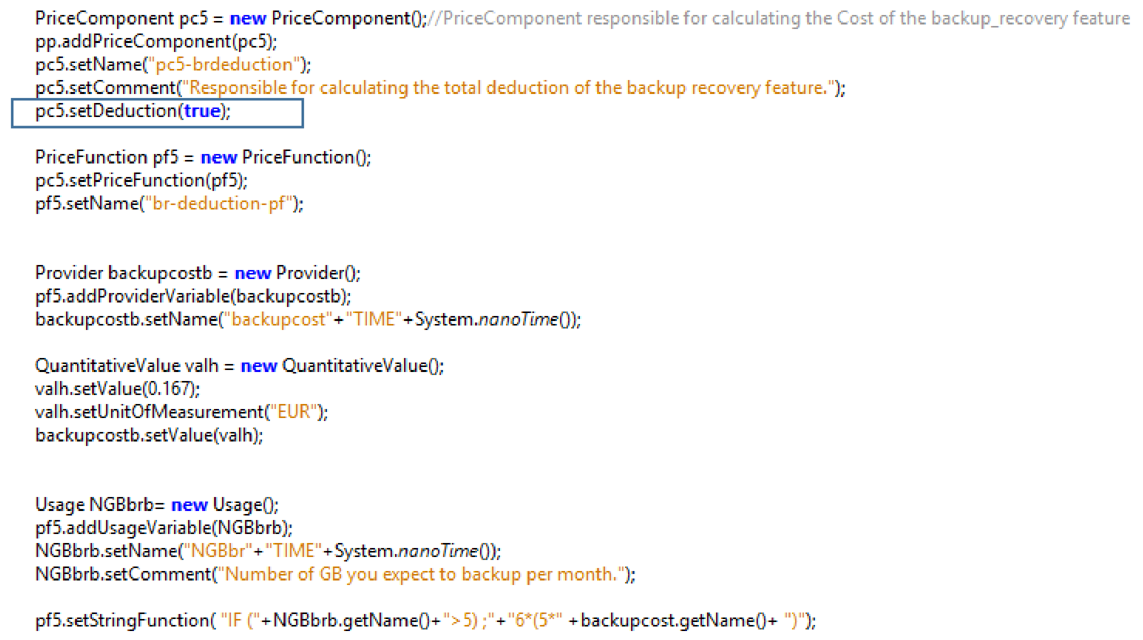
- Overall, there's 3 PriceComponents (one for each billable attribute) and 3 Deductions (2 to model the Free 5GB per month and the other to apply the discount based on the total cost of the selected service).
- The 3 billable PriceComponents are modeled using mathematical expressions like in the previous examples, e.g:



```
PriceComponent pc2 = new PriceComponent();//PriceComponent responsible for calculating the Cost of the Data Transfers per month
pp.addPriceComponent(pc2);
pc2.setName("pc1-datatransf_cost");
pc2.setComment("Responsible for calculating the cost of the data transfers.");

PriceFunction pf2 = new PriceFunction();
pc2.setPriceFunction(pf2);
pf2.setName("datatransf-cost-pf");

Usage NGBOut_z1 = new Usage();
pf2.addUsageVariable(NGBOut_z1);
NGBOut_z1.setName("ngboutz1"+"TIME"+System.nanoTime());
NGBOut_z1.setComment("Number of GB you expect to send out per month to the US West, US East, US North Central, US South Central, Europe West, Europe Nortl

Usage NGBOut_z2 = new Usage();
pf2.addUsageVariable(NGBOut_z2);
NGBOut_z2.setName("ngboutz2"+"TIME"+System.nanoTime());
NGBOut_z2.setComment("Number of GB you expect to send out per month to the Asia Pacific East, Asia Pacific Southeast, Japan East, Japan West.");

//Zone 1: US West, US East, US North Central, US South Central, Europe West, Europe North
//Zone 2: Asia Pacific East, Asia Pacific Southeast, Japan East, Japan West

Provider gboutcost_z1 = new Provider();
pf2.addProviderVariable(gboutcost_z1);
gboutcost_z1.setName("gboutcostz1"+"TIME"+System.nanoTime());

QuantitativeValue valb = new QuantitativeValue();
valb.setValue(0.08);
valb.setUnitOfMeasurement("USD");
gboutcost_z1.setValue(valb);

Provider gboutcost_z2 = new Provider();
pf2.addProviderVariable(gboutcost_z2);
gboutcost_z2.setName("gboutcostz2"+"TIME"+System.nanoTime());

QuantitativeValue valc = new QuantitativeValue();
valc.setValue(0.012);
valc.setUnitOfMeasurement("USD");
gboutcost_z2.setValue(valc);

pf2.setStringFunction("("+NGBOut_z1.getName() +"*"+gboutcost_z1.getName() + "+" + NGBOut_z2.getName() + "*" + gboutcost_z2.getName() + ")*6");
```

Microsoft Azure – PrePaid Credit Plan (5)

- The discounts are modeled using the PriceComponent class but, marked as a deduction, e.g:

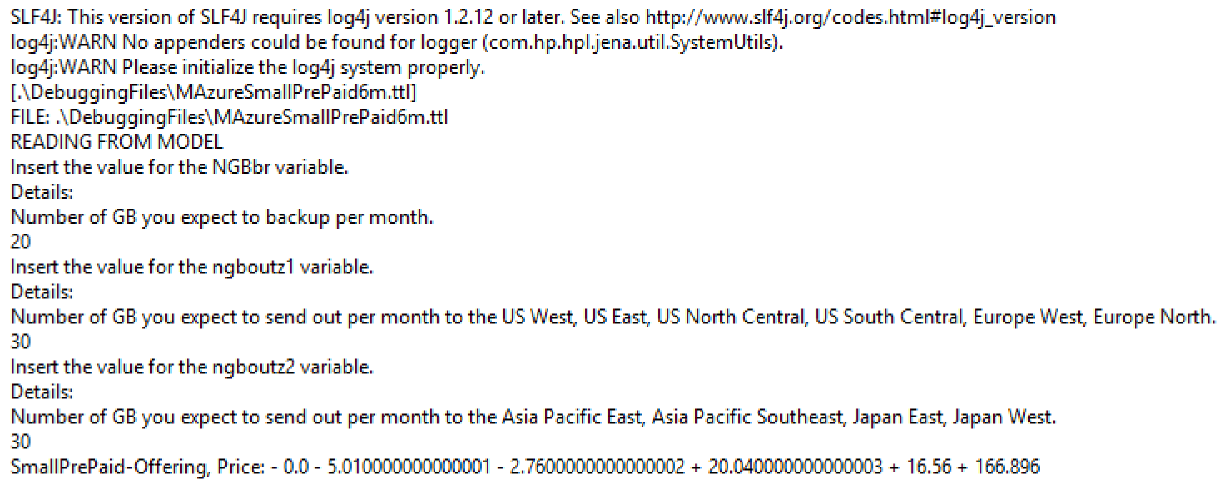


```
PriceComponent pc5 = new PriceComponent();//PriceComponent responsible for calculating the Cost of the backup_recovery feature
pp.addPriceComponent(pc5);
pc5.setName("pc5-brdeduction");
pc5.setComment("Responsible for calculating the total deduction of the backup recovery feature.");
pc5.setDeduction(true);

PriceFunction pf5 = new PriceFunction();
pc5.setPriceFunction(pf5);
pf5.setName("br-deduction-pf");


Provider backupcostb = new Provider();
pf5.addProviderVariable(backupcostb);
backupcostb.setName("backupcost"+"TIME"+System.nanoTime());

QuantitativeValue valh = new QuantitativeValue();
valh.setValue(0.167);
valh.setUnitOfMeasurement("EUR");
backupcostb.setValue(valh);


Usage NGBbrb= new Usage();
pf5.addUsageVariable(NGBbrb);
NGBbrb.setName("NGBbr"+"TIME"+System.nanoTime());
NGBbrb.setComment("Number of GB you expect to backup per month.");

pf5.setStringFunction( "IF ("+NGBbrb.getName()+">5) ;"+"6*(5*" +backupcost.getName()+ ")");
```

- What this mathematical expressions tells us is that if the total GB that the customer is expecting to back-up per month is higher than 5GB, the discount throughout the term will be the cost of 5GB multiplied by 6.
- The same approach is used to model the 5GB free per month for data transfer. To apply the discount presented on slide 36, I chose to define a mathematical expression that calculates the cost of every price component, adds the values and applies the corresponding discount.

Microsoft Azure – PrePaid Credit Plan (6)

- Once everything has been modeled, we invoke the writeToModel function to create our Semantic model.
- With the MicrosoftAzureReader class we can load and test the created model:
- As we can see, we have 3 deductions: 0.0, the global discount (the minimum amount to be eligible for the discount is 350euros), -2.76 and -5.01 correspond to the discounts of the 5GB free for back-up and data transfer per month. Adding the values, we get a total cost of 195,7Euros to rent the service for 6 months.
- Unfortunately, Microsoft's azure calculator doesn't calculate the cost for pre-paid subscriptions plans so we can't verify the price but, using the values they publish on their websites to manually perform the calculations, one can verify that the returned value is correct.



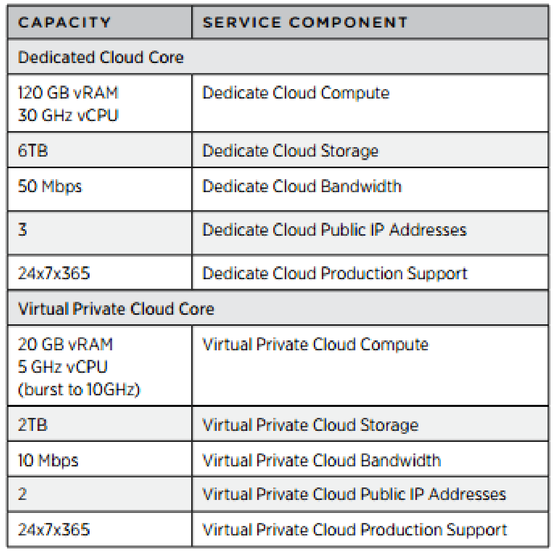
```
SLF4J: This version of SLF4J requires log4j version 1.2.12 or later. See also http://www.slf4j.org/codes.html#log4j_version
log4j:WARN No appenders could be found for logger (com.hp.hpl.jena.util.SystemUtils).
log4j:WARN Please initialize the log4j system properly.
[.\DebuggingFiles\MAzureSmallPrePaid6m.ttl]
FILE: .\DebuggingFiles\MAzureSmallPrePaid6m.ttl
READING FROM MODEL
Insert the value for the NGBbr variable.
Details:
Number of GB you expect to backup per month.
20
Insert the value for the ngboutz1 variable.
Details:
Number of GB you expect to send out per month to the US West, US East, US North Central, US South Central, Europe West, Europe North.
30
Insert the value for the ngboutz2 variable.
Details:
Number of GB you expect to send out per month to the Asia Pacific East, Asia Pacific Southeast, Japan East, Japan West.
30
SmallPrePaid-Offering, Price: - 0.0 - 5.010000000000001 - 2.7600000000000002 + 20.040000000000003 + 16.56 + 166.896
```

VMWare Hybrid vCloud – Recurring Resource Pooling

- As presented on Table.1, their Hybrid vCloud uses (but not only) a PB Bundled PrePaid Credit Plan pricing method where the user buys resources instead of Virtual Machines. Costs of consumption are then debit from the purchased amount.

- VMWare offers two types of offerings: Dedicated or Virtual Private Cloud:

- “To start your first vCloud Hybrid Service cloud you need to order a Core package. A vCloud Hybrid Service Core includes all the service components needed to provision your new cloud and is only used when you want to order a new service instance.”



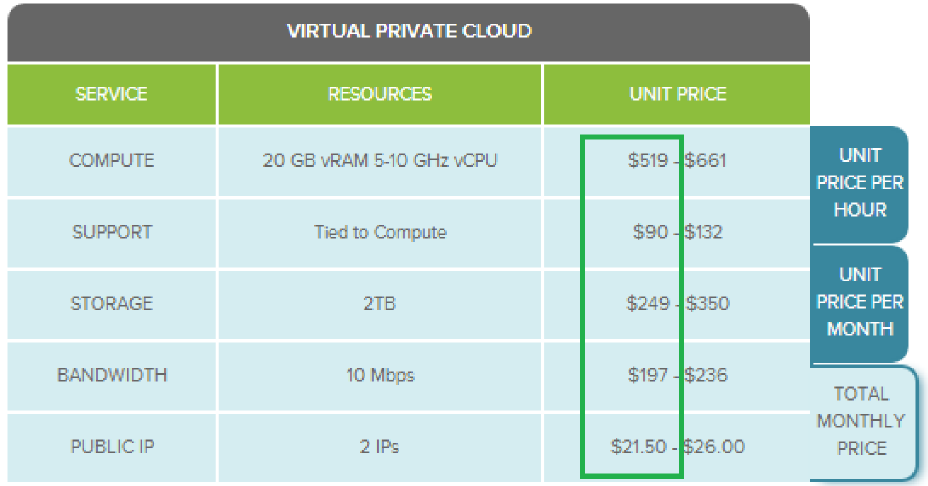
CAPACITY	SERVICE COMPONENT
Dedicated Cloud Core	
120 GB vRAM 30 GHz vCPU	Dedicate Cloud Compute
6TB	Dedicate Cloud Storage
50 Mbps	Dedicate Cloud Bandwidth
3	Dedicate Cloud Public IP Addresses
24x7x365	Dedicate Cloud Production Support
Virtual Private Cloud Core	
20 GB vRAM 5 GHz vCPU (burst to 10GHz)	Virtual Private Cloud Compute
2TB	Virtual Private Cloud Storage
10 Mbps	Virtual Private Cloud Bandwidth
2	Virtual Private Cloud Public IP Addresses
24x7x365	Virtual Private Cloud Production Support

http://www.vmware.com/files/pdf/vchs/VMware_vCloud_Hybrid_Service_Purchasing_and_Subscription.pdf

- This means that the minimum that the user can buy are the quantities specified on the image. Once he has decided on his core package, additional resources can be bought à la carte. E.g: (Virtual Private Cloud) After we’ve bought our core package and realize we need an extra 12TB of storage, we can add 6 units (each unit has 2TB) of Virtual Private Cloud Storage to our core package.

VMWare Hybrid vCloud – Recurring Resource Pooling (2)

- Currently, VMWare only offers two Subscriptions plans for this service: duration of 3 or 12 Months.
- Since VMWare uses a PB Bundled approach, CPU and Memory are charged as a single unit. Other resources are sold as single (or independent) units.
- Each of these units have a fixed fee per month:



VIRTUAL PRIVATE CLOUD

SERVICE	RESOURCES	UNIT PRICE
COMPUTE	20 GB vRAM 5-10 GHz vCPU	$519 - $661
SUPPORT	Tied to Compute	$90 - $132
STORAGE	2TB	$249 - $350
BANDWIDTH	10 Mbps	$197 - $236
PUBLIC IP	2 IPs	$21.50 - $26.00

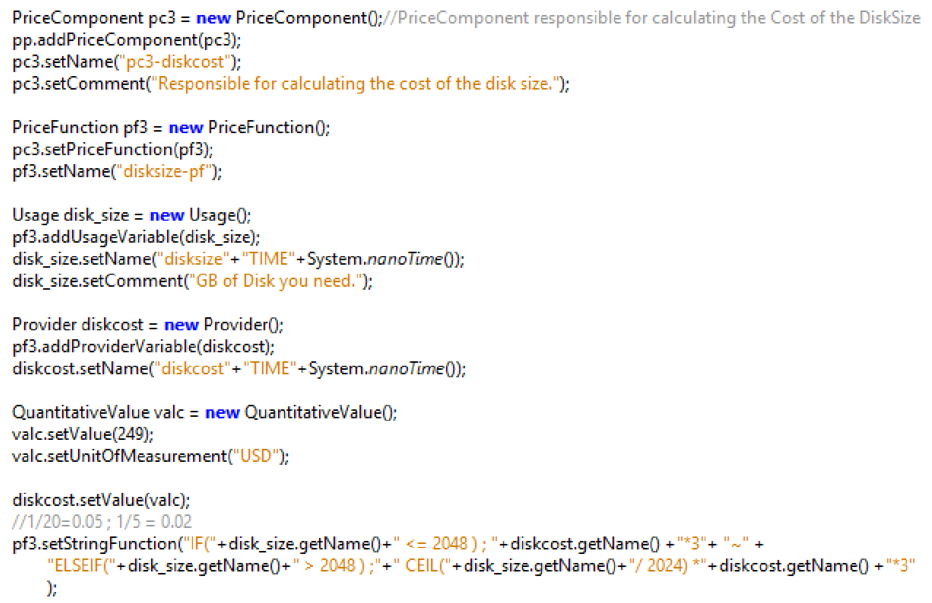
http://vcloud.vmware.com/about_services/pricing

VMWare Hybrid vCloud – Recurring Resource Pooling (2)

- There's a total of 5 billable attributes:
    - RAM+CPU
    - Support
    - Storage
    - Bandwidth
    - StaticIP
- Each of these components is sold by the unit, each unit has a fixed fee per month. E.g, in the case of Storage, each 2TB unit costs 249$ per month.
- Considering this, we can easily see that the customer will pay 249$*3 for the 3 Months subscriptions. But, in case the user needs 5,6,7 or 8 TB of Storage, his core package will need more Storage units aside from the one it already has (1 unit (2TB) by default). In order to calculate the number of Units that the customer needs, we simply have to divide the number of requested Storage by the quantity that each storage unit has. E.g, I want 7TB of storage for my core package; I'll need 7/2 = 3,5 storage units. While this is mathematically correct, units are sold as a whole, they can't be divided or made smaller. As such, in order to get my 7TB I would need **CEIL**(7/2) = 4units.
- As SPARQL "knows" this function, we only need to use it on the PriceComponent's mathematical expressions.

VMWare Hybrid vCloud – Recurring Resource Pooling (3)

- Storage PriceComponent:

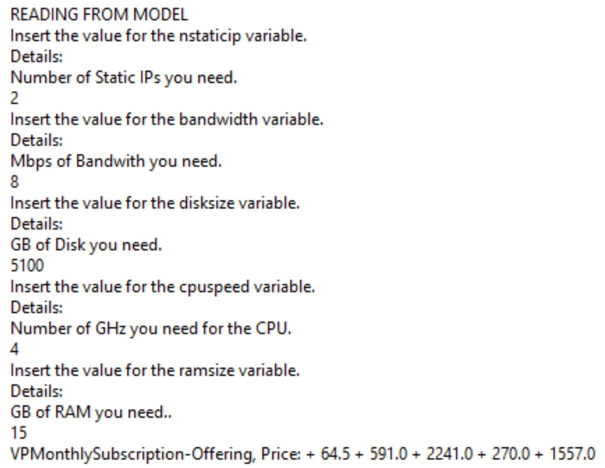


```
PriceComponent pc3 = new PriceComponent();//PriceComponent responsible for calculating the Cost of the DiskSize
pp.addPriceComponent(pc3);
pc3.setName("pc3-diskcost");
pc3.setComment("Responsible for calculating the cost of the disk size.");

PriceFunction pf3 = new PriceFunction();
pc3.setPriceFunction(pf3);
pf3.setName("disksize-pf");

Usage disk_size = new Usage();
pf3.addUsageVariable(disk_size);
disk_size.setName("disksize"+"TIME"+System.nanoTime());
disk_size.setComment("GB of Disk you need.");

Provider diskcost = new Provider();
pf3.addProviderVariable(diskcost);
diskcost.setName("diskcost"+"TIME"+System.nanoTime());

QuantitativeValue valc = new QuantitativeValue();
valc.setValue(249);
valc.setUnitOfMeasurement("USD");

diskcost.setValue(valc);
//1/20=0.05 ; 1/5 = 0.02
pf3.setStringFunction("IF("+disk_size.getName()+" <= 2048 ) ; "+diskcost.getName() +"*3"+ "~" +
    "ELSEIF("+disk_size.getName()+" > 2048 ) ;"+" CEIL("+disk_size.getName()+"/ 2024) *"+diskcost.getName() +"*3"
    );
```

- It's important to note the upper case on the "CEIL" function. Every word that is not a variable, like functions or strings, should be set in uppercase.
- In case the number of requested resources is higher than the default value for the unit, it's necessary to divide that number by the default value in order to know the number of necessary units.

VMWare Hybrid vCloud – Recurring Resource Pooling (4)

- Once everything has been modeled, we invoke the writeToModel() function to create our Semantic model.
- With the VMWareVPCloudReader class we can load and test the created model.



```
READING FROM MODEL
Insert the value for the nstaticip variable.
Details:
Number of Static IPs you need.
2
Insert the value for the bandwidth variable.
Details:
Mbps of Bandwith you need.
8
Insert the value for the disksize variable.
Details:
GB of Disk you need.
5100
Insert the value for the cpuspeed variable.
Details:
Number of GHz you need for the CPU.
4
Insert the value for the ramsize variable.
Details:
GB of RAM you need..
15
VPMonthlySubscription-Offering, Price: + 64.5 + 591.0 + 2241.0 + 270.0 + 1557.0
```

- As you can see, we have 5 pricing values which correspond to the cost of each of the billable attributes:
- 64.5/3 = 21.5USD -> StaticIPs cost
- 591/3 = 197USD -> Bandwidth cost
- 2241.0/3 = 747USD = 3*249USD = 3 units of Storage cost
- 270/3 = 90 -> Support cost
- 1557/3=519USD -> CPU+RAM cost

- Unfortunately, VMWare doesn't provide a calculator to verify the final pricing value obtained but it can be easily validated by performing the calculations manually and comparing both results.

Conclusion

- After successfully modeling each of the use cases pricing methods, we can see that both Pricing API and the LinkedUSDL pricing model are capable of tackling today's cloud market's pricing scene.
- Each billable attribute of the service can be modeled using PriceComponent objects and their properties/features.
- Discounts were modeled by either using the PriceComponent object and defining it has a "Deduction" or, by including it on the PriceFunction of the PriceComponents (e.g: ClougSigma's example).
- The current LinkedUSDL pricing model is very flexible. In fact, most obstacles derived from the API rather than the model itself.
- The PricingAPI is still under development however, this challenge helped us to greatly improve its and, at the same time, prove that both are viable tools to model the current cloud services and their pricing methods.
- With these examples, I hope that you can see how easy it becomes to model Services and their pricing methods using the LinkedUSDL Pricing API.