

Technical Portfolio

Project Virtualization Team 7



Zoran van Rooijen

500682019

IN202

Inhoud

[Portfolio Research 2](#_Toc399481158)

[ROI & Company/project costs 2](#_Toc399481159)

[Office 2](#_Toc399481160)

[Costs 2](#_Toc399481161)

[ROI calculations 2](#_Toc399481162)

[What is REST 3](#_Toc399481163)

[What is Velocity 5](#_Toc399481164)

[What is Servlet 5](#_Toc399481165)

[Hypervisor/KVM/LibVirt 5](#_Toc399481166)

# Portfolio Research

# ROI & Company/project costs

## Office

To calculate the hour rates for ITopia employees, I did research to the costs we need to make for our daily operation.

First I started building the office using Ikea desks and lighting.  
<http://www.ikea.com/nl/nl/catalog/categories/departments/workspaces/>

After that we needed some comfortable chairs which wouldn’t hurt the backs of our employees.  
<http://www.hermanmiller.com/products/seating.html>

Us being a small company we had buy software to use for our business.  
<http://tweakers.net/zoeken?keyword=software>

Al of our company’s employees where in need for laptops.  
http://www.laptopshop.nl/?\_ga=1.80104479.926612589.1411547890

And of course a company needs office space.  
<http://www.fundainbusiness.nl/kantoor/almere/object-48911761-transistorstraat-71-d/>

## Costs

First of all we researched the server costs. For this we used the Dell server website:  
http://www.dell.com/nl/bedrijven/p/servers  
to custom build a server that would fit the requirements. The server specifications are added to the portfolio.

## ROI calculations

To understand how to make a good ROI calculation I have done a good amount of research. I had to research what the hour rate of ITopia experts would be, including taxes and insurance, and how to calculate gross margin and so on. Here are the websites which I used to research my calculations.

<http://www.investopedia.com/terms/r/returnoninvestment.asp>  
<http://www.entrepreneur.com/article/204854>  
http://www.marketingmo.com/campaigns-execution/how-to-calculate-roi/  
<http://en.wikipedia.org/wiki/Return_on_investment>  
<http://www.wikihow.com/Calculate-Payroll>  
<http://en.wikipedia.org/wiki/Depreciation>  
<http://en.wikipedia.org/wiki/Gross_margin>  
http://www.geekwire.com/2011/17-starting-business/

# What is REST

Representational state transfer (REST) is an abstraction of the architecture of the World Wide Web; more precisely, REST is an architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed hypermedia system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements.

The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine. REST has been applied to describe desired web architecture, to identify existing problems, to compare alternative solutions and to ensure that protocol extensions would not violate the core constraints that make the web successful. Fielding developed REST in collaboration with his colleagues during the same period he worked on HTTP 1.1 and Uniform Resource Identifiers (URI).

The REST architectural style is also applied to the development of web services as an alternative to other distributed-computing specifications such as SOAP. One can characterize web services as "RESTful" if they conform to the constraints described in the architectural constraints section. See the applied to web services section if you are only interested in the application of REST to web APIs.

The architectural properties of REST are realized by applying specific interaction constraints to components, connectors, and data elements. The formal REST constraints are:

**Architectural constraints**

**Client–server**

A uniform interface separates clients from servers. This separation of concerns means that, for example, clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable. Servers and clients may also be replaced and developed independently, as long as the interface between them is not altered.

**Stateless protocol**

The client–server communication is further constrained by no client context being stored on the server between requests. Each request from any client contains all the information necessary to service the request, and session state is held in the client. Important to note is that the session state can be transferred by the server to another service such as a database to maintain a persistent state for a period and allow authentication. The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that may be used the next time the client chooses to initiate a new state-transition.

**Cacheable**

As on the World Wide Web, clients can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients from reusing stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.

**Layered system**

A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load balancing and by providing shared caches. They may also enforce security policies.

**Code on demand**

**Client-side scripting**

Servers can temporarily extend or customize the functionality of a client by the transfer of executable code. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript. "Code on demand" is the only optional constraint of the REST architecture.

**Uniform interface**

The uniform interface constraint is fundamental to the design of any REST service. The uniform interface simplifies and decouples the architecture, which enables each part to evolve independently. The four constraints for this uniform interface are:

**Identification of resources**

Individual resources are identified in requests, for example using URIs in web-based REST systems. The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server may send data from its database as HTML, XML or JSON, none of which are the server's internal representation, and it is the same one resource regardless.

Manipulation of resources through these representations

When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource.

**Self-descriptive messages**

Each message includes enough information to describe how to process the message. For example, which parser to invoke may be specified by an Internet media type (previously known as a MIME type). Responses also explicitly indicate their cacheability.

Hypermedia as the engine of application state (HATEOAS)

Clients make state transitions only through actions that are dynamically identified within hypermedia by the server (e.g., by hyperlinks within hypertext). Except for simple fixed entry points to the application, a client does not assume that any particular action is available for any particular resources beyond those described in representations previously received from the server.

One can characterise applications conforming to the REST constraints described in this section as "RESTful". If a service violates any of the required constraints, it cannot be considered RESTful.

Complying with these constraints, and thus conforming to the REST architectural style, enables any kind of distributed hypermedia system to have desirable emergent properties, such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability.

# What is Velocity

Apache Velocity is a Java-based template engine that provides a template language to reference objects defined in Java code. It aims to ensure clean separation between the presentation tier and business tiers in a Web application (the model–view–controller design pattern).

Velocity is an open source software project hosted by the Apache Software Foundation.

# What is Servlet

The servlet is a Java programming language class used to extend the capabilities of a server. Although servlets can respond to any types of requests, they are commonly used to extend the applications hosted by web servers, so they can be thought of as Java applets that run on servers instead of in web browsers. These kinds of servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET.

# Hypervisor/KVM/LibVirt

A hypervisor or virtual machine monitor (VMM) is a piece of computer software, firmware or hardware that creates and runs virtual machines.

A computer on which a hypervisor is running one or more virtual machines is defined as a host machine. Each virtual machine is called a guest machine. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources.

KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module, kvm.ko, that provides the core virtualization infrastructure and a processor specific module, kvm-intel.ko or kvm-amd.ko. KVM also requires a modified QEMU although work is underway to get the required changes upstream.

Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc.

Libvirt allows management of different virtualization solutions such as KVM and Xen through a common (programming and user) interface.

**Sources:**

http://en.wikipedia.org/wiki/Hypervisor

http://www.linux-kvm.org/page/Main\_Page

https://wiki.debian.org/KVM

https://wiki.debian.org/libvirt