Libre

atix

Fundación

24

2018

10 Años Aportando al Conocimiento

2018 - Bolvia

http://revista.atixlibre.org
Twitter: @atixlibre
Facebook: facebook.com/Atix.Libre

DIRECCION GENERAL
Esteban Saavedra Lopez

DIAGRAMACION
Jenny Saavedra Lopez
Esteban Saavedra Lopez

REVISION
Jenny Saavedra Lopez

CONTACTO
info@atixlibre.org
http://revista.atixlibre.org

AtixLibre

El conocimiento es el mejor legado que la humanidad ha transmitido desde sus inicios, ya que este ha representado la mejor forma de supervivencia de nuestra especie. Por eso es tan importante que las personas con mayor experiencia, las instituciones públicas o privadas sean las llamadas a abrir espacios de oportunidad, de capacitación y de dejar una escuela llena de conocimientos y experiencias a nuestras nuevas generaciones.

Si bien muchos sabemos de la importancia de este legado, son muy pocas las personas o instituciones que realizan este tipo de actividades, desde **AtixLibre** hacemos un llamado para unirse en esta cruzada y sobre todo a la noble actividad de capacitar, formar y brindar oportunidad a los niños y a los más jóvenes.

Si bien hoy en día se habla mucho de tecnologías libres, también deberíamos hablar de aperturar espacios libres, sin restricciones, donde desde los niños y los más jóvenes puedan demostrar sus capacidades y compartir sus conocimientos; no solo deberíamos conformarnos con decir que el conocimiento no es nada si no se comparte; sino deberíamos trabajar en facilitar espacios y promover actividades de intercambio y transmisión de conocimiento.

En este número quiero dar la bienvenida a Stephanie, la más joven de nuestras autoras, que con tan solo 16 años tiene un enorme potencial tecnológico y sobre todo muchas ganas de compartir sus conocimientos.

Sean bienvenidos a nuestra edición número 24.

Esteban Saavedra L.

Presidente Fundación AtixLibre

Contenido

Número 24 - Julio 2018

1

Flask
Una Broma Seria

Actualmente en el desarrollo moderno de aplicaciones web se utilizan distintos Frameworks que son herramientas que nos dan un esquema de trabajo y una serie de utilidades y funciones que nos facilita y nos abstrae de la construcción de aplicaciones web dinámicas.

Flask es un framework minimalista escrito en Python, concebido para facilitar el desarrollo de Aplicaciones Web bajo el patrón MVC, de forma rápida y con un mínimo número de líneas de código.

Introducción

A veces terminamos como desarrolladores frente a una tarea que se puede resolver con un Framework como Django, pero cuya solución resulta demasiado compleja, debido a las restricciones o la misma estructura del Framework o la tarea es tan simple que el uso de un Framework no se justifica.

A veces menos es más y la solución más simple con un micro Framework como Flask es lo adecuado.

En este artículo se muestra a través de geistesblitze [1], una pequeña aplicación Web para recolectar, de forma rápida, ideas. El código fuente de la aplicación está disponible bajo una licencia libre en GitHub. Las partes más importantes de la aplicación se muestran en este artículo.

De DENIED a Flask 1.0

Armin Ronacher [2] bajo un seudónimo con testimonios falsos y un vídeo de cinco minutos, lanzó hace un poco más de ocho años, inicialmente como broma, el micro framework DENIED. Usando un truco para empacar todo lo necesario (el kit de funciones WSGI Werkzeug [3] y el motor de plantillas Jinja2 [4]) en un único archivo deny.py.

Las reacciones fueron positivas y así empezó el desarrollo del micro framework Flask [5], que hace poco, el 26 de abril, publicó su versión 1.0. Algunas de sus ideas eran no sólo muy interesantes sino que han resultado ser muy útiles.

Flask es simple y se limita a lo que toda aplicación Web necesita: Un sistema de enrutamiento y un sistema de platillas, y renuncia a características que no todas las aplicaciones Web necesita como por ejemplo

una Base de Datos, ya sea relacional o sin esquem o un Framework Javascript para el desarrollo del Frontend.

Flask y Python 3

Flask recomienda usar la versión más reciente de Python 3, pero soporta Python 3.4 en adelante Python 2.7 y PyPy.

En este artículo se utiliza la versión 3.6.5 de Python, pero la aplicación funciona también con la versión 2.7.15.

geistesblitze

La aplicación permite el registro de nuevos usuarios y la recolección rápida y fácil de nuevas ideas. Muestra el uso de algunas características de Flask, como las plantillas (render_template), redirección (redirect), mensajes (flash) y la referencia de vistas basada en las funciones de Python (url_for).

La aplicación hace uso además de extensiones comunes, que demuestran la filosofía de Flask, como:

- Flask-Bootstrap [6]
- Flask-Login [7]
- Flask-SQLAlchemy [8], y
- Flask-WTF [9].

Estructura

Los únicos valores predeterminados que existen en Flask (aunque pueden ser cambiados con parámetros al crear el objeto de la aplicación) son los directorios static y templates, que deben estar en el mismo directorio donde existe la aplicación.

La aplicación puede estar en su totalidad en un solo archivo o puede estar dividida en varios archivos dentro de un paquete, como en este caso:

```
geistesblitze
├── forms.py
├── __init__.py
├── models.py
├── static
│   └── favicon.ico
├── templates
│   ├── 403.html
│   ├── 404.html
│   ├── add_idea.html
│   ├── base.html
│   ├── idea.html
│   ├── ideas.html
│   ├── index.html
│   ├── login.html
│   └── register.html
└── views.py
```

No todos los archivos se muestran en este artículo, pero todos están incluidos en en el proyecto en GitHub.

Instalación

En Python, a partir de la versión 3.4, la biblioteca estándar incluye el módulo venv, que usamos para crear un entorno virtual para nuestra aplicación y sus dependencias:

```
$ git clone https://github.com/nnrcschmdt/geistesblitze
$ cd geistesblitze
$ pyvenv venv
$ source venv/bin/activate
(venv) $ pip install -r requirements.txt
```

Aplicación

La aplicación consiste de formularios, modelos y vistas que están en los archivos forms.py, models.py y views.py respectivamente dentro del paquete geistesblitze.

El archivo **`__init__.py`** sirve para marcar el sub directorio geistesblitze como un paquete y para inicializar la aplicación creando los objetos app, bootstrap, e inicializando objeto db, la base de datos.

```
from os.path import abspath, dirname, join
from flask import Flask
from flask_bootstrap import Bootstrap
from flask_login import LoginManager
from geistesblitze.models import db, User

basedir = abspath(dirname(__file__))

app = Flask(__name__)
app.config['SECRET_KEY'] = 'hard to guess string'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:////%s' % join(basedir,
                                                               'geistesblitze.sqlite')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

bootstrap = Bootstrap(app)

db.init_app(app)
Además del inicializar el objeto login_manager, para manejar las sesiones.
login_manager = LoginManager(app)
login_manager.session_protection = 'strong'
login_manager.login_view = 'login'

import geistesblitze.views

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

Finalmente se crea el comando create_all, para ser ejecutado desde la consola y crear las tablas necesarias.

```
@app.cli.command()
def create_all():
    """Create all the tables"""
    db.create_all()
```

Para poder ejecutar el comando tenemos que hacer visible nuestra aplicación:

```
(venv) $ export FLASK_APP=geistesblitze
(venv) $ flask create_all
```

Modelos (models.py)

La base de datos consiste de dos tablas: una de usuarios y otra de ideas.

```
from flask_login import UserMixin
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash

db = SQLAlchemy()
```

La tabla de usuarios solo guarda y luego verifica el hash de la contraseña. No se almacenan las contraseñas en texto plano, simplemente se usan las funciones generate_password_hash() para generar un hash seguro y check_password_hash() para verificarlo.

Estas funciones son parte deel kit de funciones WSGI Werkzeug, uno de los componentes clave de Flask.

```
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), unique=True)
    password_hash = db.Column(db.String(128), unique=False)
    ideas = db.relationship('Idea', backref='user', lazy='dynamic')

    @property
    def password(self):
        raise AttributeError('password is not a readable attribute')

    @password.setter
    def password(self, password):
        self.password_hash = generate_password_hash(password)

    def verify_password(self, password):
        return check_password_hash(self.password_hash, password)

    def __repr__(self):
        return '<User %r>' % self.username
```

Cada idea contiene una referencia al usuario que creó la idea, para que luego cada usuario solo pueda ver sus propia ideas.

```
class Idea(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(128), unique=False)
    description = db.Column(db.Text(), nullable=True)

    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))

    def __repr__(self):
        return '<Idea %r>' % self.name
```

Formularios (forms.py)

La aplicación contiene tres formularios creados con WTForms [10]. No sólo se pueden definir formularios sino también implementar su validación.

```
from flask_wtf import FlaskForm
from wtforms import (PasswordField, StringField, SubmitField, TextAreaField,
ValidationError)
from wtforms.validators import EqualTo, DataRequired
from geistesblitze.models import User
```

El formulario de registro verifica que las dos contraseñas especificadas coincidan y también si el nombre de usuario no exista en la base de datos.

Del mismo modo se podría verificar por ejemplo una longitud mínima de los nombres de usuario o el formato de la contraseña al momento del registro.

```
class RegisterForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password',
                             validators=[DataRequired(),
                                         EqualTo('password2',
                                                 message='Passwords must match.')])
    password2 = PasswordField('Confirm Password', validators=[DataRequired()])
    submit = SubmitField('Register')

    def validate_username(self, field):
        if User.query.filter_by(username=field.data).first():
            raise ValidationError('Username already in use.')
```

El formulario de inicio de sesión verifica que se hayan especificado el nombre de usuario y la contraseña.

```
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Log In')
```

El formulario para agregar una nueva idea verifica que tanto el nombre como la descripción de la idea se hayan especificado.

```
class AddIdeaForm(FlaskForm):
    name = StringField('Name', validators=[DataRequired()])
    description = TextAreaField('Description', validators=[DataRequired()])
    submit = SubmitField('Save')
```

Vistas (views.py)

```
from flask import render_template, redirect, flash, url_for
from flask_login import login_required, login_user, logout_user, current_user

from geistesblitze import app
from geistesblitze.forms import AddIdeaForm, LoginForm, RegisterForm
from geistesblitze.models import db, User, Idea
```

Al empezar se mostrará una página con el formulario de registro y el formulario para iniciar una sesión.

```
@app.route('/')
def index():
    return render_template('index.html',
                           register_form=RegisterForm(),
                           login_form=LoginForm())
```

Al registrar un nuevo usuario, el formulario de registro se muestra durante una solicitud GET, y cuando se hace una solicitud POST se intenta validar el formulario, verificando la existencia o no del usuario, en caso de no existir este será creado y almacenado para luego emitir un mensaje adecuado y el usuario se reenvía a la página de inicio de sesión.

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    form = RegisterForm()

    if form.validate_on_submit():
        user = User(username=form.username.data, password=form.password.data)
        db.session.add(user)
        db.session.commit()
        flash('Your account has been created')
        return redirect(url_for('login'))
    return render_template('register.html', form=form)
```

Al iniciar sesión, el formulario de inicio de sesión se muestra en una solicitud GET y en una solicitud POST se intenta validar el usuario y la contraseña especificadas. Si son correctas se inicia la sesión del usuario (login_user()) y el usuario es reenviado a la lista de sus propias ideas, o se muestra el mensaje de error apropiado cuando el usuario y la contraseña no coinciden.

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user is not None and user.verify_password(form.password.data):
            login_user(user, True)
            return redirect(url_for('get_ideas'))
        flash('Invalid username of password')
    return render_template('login.html', form=form)
```

Al cerrar la sesión, el usuario es desconectado (logout_user()), se emite un mensaje adecuado y el usuario es reenviado a la página de inicio de sesión.

```
@app.route('/logout')
def logout():
    logout_user()
    flash('You have been logged out')
    return redirect(url_for('login'))
```

Al visualizar una idea, se muestran los detalles de la idea y una lista de ideas. Si la idea dada no existe o la idea no fue agregada por el usuario que inició sesión, se mostrará una página de error correspondiente.

```
@app.route('/ideas/<int:id>')
@login_required
def get_idea(id):
    idea = Idea.query.filter_by(id=id).first()
    ideas = Idea.query.filter_by(user=current_user).all()

    if idea is None:
        return render_template('404.html'), 404
    if idea.user != current_user:
        return render_template('403.html'), 403
    return render_template('idea.html', idea=idea, ideas=ideas)
```

Las ideas del usuario actualmente conectado aparecen en la lista de ideas.

```
@app.route('/ideas')
@login_required
def get_ideas():
    ideas = Idea.query.filter_by(user=current_user).all()
    return render_template('ideas.html', ideas=ideas)
```

Al agregar una nueva idea, el formulario para una nueva idea se muestra en una solicitud GET, y cuando se hace una solicitud POST se intenta validar el formulario (que el nombre y la descripción de la idea se han especificado), la idea se crea y se guarda un mensaje apropiado, se emite un mensaje y el usuario es reenviado a la lista de ideas.

```
@app.route('/add_idea', methods=['GET', 'POST'])
@login_required
def add_idea():
    form = AddIdeaForm()

    if form.validate_on_submit():
        idea = Idea(name=form.name.data,
        description=form.description.data)
        idea.user = current_user
        db.session.add(idea)
        db.session.commit()
        flash('Your idea has been saved')
        return redirect(url_for('get_ideas'))
    return render_template('add_idea.html', form=form)
```

Plantillas

Otro componente central de Flask es el motor de plantillas Jinja2. Inspirado en las plantillas de Django, Jinja2 tiene una serie de características que lo convierten en un sistema muy potente, flexible y rápido. En particular, las plantillas no solo pueden heredarse y los bloques sobrescribirse, sino que también puede ampliar estos bloques. En combinación con bootstrap, esto resulta un marco muy flexible para implementar aplicaciones.

En la plantilla base se muestra la navegación, dependiendo de si el usuario ha iniciado sesión o no, el enlace a la lista de ideas o los enlaces para iniciar sesión y registrarse.

Los mensajes se muestran en el área de contenido, si corresponde.

```
{% extends "bootstrap/base.html" %}

{% block title %}Geistesblitze{% endblock %}

{% block head %}
{{ super() }}
<link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
{% endblock %}

{% block navbar %}
<div class="navbar" role="navigation">
    <div class="container">
        <div class="navbar-header">
            <a class="navbar-brand" href="{{ url_for('get_ideas') }}">Geistesblitze</a>
        </div>
        <div class="navbar-collapse collapse">
            {% if current_user.is_authenticated %}
            <ul class="nav navbar-nav navbor-left">
                <li><a href="{{ url_for('add_idea') }}">Add Idea</a></li>
            </ul>
            {% endif %}
            <ul class="nav navbar-nav navbar-right">
                {% if current_user.is_authenticated %}
                <li><a href="{{ url_for('logout') }}">Log Out</a></li>
                {% else %}
                <li><a href="{{ url_for('login') }}">Log In</a></li>
                <li><a href="{{ url_for('register') }}">Register</a></li>
                {% endif %}
            </ul>
        </div>
    </div>
</div>
{% endblock %}
```

```
{% block content %}
<div class="container">
    {% for message in get_flashed_messages() %}
    <div class="alert alert-warning">
        <button type="button" class="close" data-dismiss="alert">&amp;times;</button>
        {{ message }}
    </div>
    {% endfor %}

    {% block page_content %}
    {% endblock %}
</div>
{% endblock %}
```

La extensión Flask-Bootstrap contiene macros que hacen que sea mucho más fácil representar los formularios. Para mostrar el formulario para la inscripción para iniciar la sesión o agregar una nueva idea, solo necesita escribir wtf.quick_form(form). Por ejemplo para el formulario de registro:

```
{% extends "base.html" %}
{% import "bootstrap/wtf.html" as wtf %}
{% block title %}Register{% endblock %}
{% block page_content %}
  <div class="page-header">
    <h1>Register</h1>
  </div>
  <div class="col-md-4">
    {{ wtf.quick_form(form) }}
  </div>
{% endblock %}
```

Ahora podemos iniciar la aplicación, dentro del entorno virtual, sólo tenemos que ejecutar

```
(venv) $ flask run
```

La aplicación es lo más simple para empezar. Carece de funciones como borrar una idea, y un Frontend más moderno. En una próxima entrega implementaré un Frontend un poco más moderno con jQuery [11]. Dejo a las lectoras y los lectores como reto, crear mientras tanto, otro con algo más de moda como Angular o React, y hacer un pull-request al repositorio del proyecto.

Referencias

[1] https://github.com/nnrcschmdt/geistesblitze

[2] http://lucumr.pocoo.org/about/

[3] http://werkzeug.pocoo.org/

[4] http://jinja.pocoo.org/

[5] http://flask.pocoo.org/

[6] https://pypi.org/project/Flask-Bootstrap/

[7] https://pypi.org/project/Flask-Login/

[8] https://pypi.org/project/Flask-SQLAlchemy/

[9] https://pypi.org/project/Flask-WTF/
[10] https://pypi.org/project/WTForms/
[11] https://jquery.com/

Ernesto Rico Smith
Usuario GNU/Linux desde 1994
e.rico.schmidt@gmail.com
BOLIVIA

2 Vagrant
Automatizando Entornos Virtuales

Infraestructura

En un contexto tecnológico donde el común denominador es el término virtualización, han surgido numerosas herramientas que permiten de una u otra forma gestionar este tipo de entornos, dándonos la facilidad de poder preparar, configurar, desplegar y compartir entornos de trabajo ya sea para desarrollos, demostraciones o entornos de producción.

Estas herramientas permiten a las unidades de desarrollo, QA e infraestructura optimizar su trabajo y reducir considerablemente el tiempo y la carga de trabajo en la ejecución de estos entornos.

Introducción

En la última década, los centros de datos y las arquitecturas de servidores se han revolucionado con la práctica de la virtualización; que básicamente se refiere a la capacidad de alojar recursos computacionales que alguna vez sólo dependieron del hardware disponible, y ahora son desplegados a manera de contenedores de software especializado.

La virtualización nos otorga la capacidad de usar entornos virtuales flexibles, convertidos en recursos computacionales compartidos, permitiendo a desarrolladores y administradores de sistemas ampliar sus capacidades de despliegue o ejecución ya sea para entornos de desarrollo, test o producción, sin necesidad de modificar el sistema operativo principal o ejecutando las mismas en servidores físicos completamente separados.

A medida que los proyectos se vuelven más complicados, también es más fácil olvidar las configuraciones o adecuaciones realizadas, más aún cuando se tienen numerosos equipos y proyectos de desarrollo. Para estos casos un entorno de desarrollo virtualizado puede ser la solución, ya que en lugar de tener un servidor dedicado con su propio entorno de software (aplicaciones, base de datos, etc), se puede disponer de varios entornos virtualizados, cada uno configurado de forma independiente y sin afectar al normal funcionamiento de los demás entornos.

Uso de contextos virtualizados

- Crear una máquina virtual con un sistema operativo distinto de la máquina física.
- Implementar entornos de desarrollo con lo mínimo necesario de los entornos de producción.
- Capacidad de modificar las propiedades de la máquina virtual (RAM, CPUs, etc)
- Compartir carpetas entre la máquina física y la máquina virtual.

- Instalar y configurar software específico que no afecte a los demás entornos.
- Hacer uso de herramientas de aprovisionamiento para un despliegue ágil.
- Homogenizar entornos de desarrollo o test en contextos locales y remotos

¿Que es Vagrant?

VAGRANT

- Vagrant es una poderosa herramienta, que permite administrar y gestionar entornos virtalizados
- Vagrant is una herramienta que simplifica el flujo de trabajo y reduce la carga de trabajo necesaria para correr y operar máquinas virtuales.
- Vagrant es una herramienta para crear y configurar máquinas virtuales portables y reproducibles, de manera automática.

Características

- Herramienta desarrollada en Ruby
- Es multiplataforma
- Facilita la creación de entornos virtualizados
- Útil para tareas de desarrollo
- Ofrece una interfaz de línea de comandos super sencilla para interactuar.
- Soporta la mayor parte de las soluciones de virtualización: VirtualBox, VMWare, Docker e Hyper-V
- Soporta las soluciones más populares de herramientas de configuración: Ansible, Chef, Puppet y Salt
- Facilita procedimientos para distribuir y compartir entornos virtualizados de propósito general o específico.
- Dispone de repositorios de BOXs preconfigurados
- Utilizada para el entrenamiento e instrucción

Objetivo

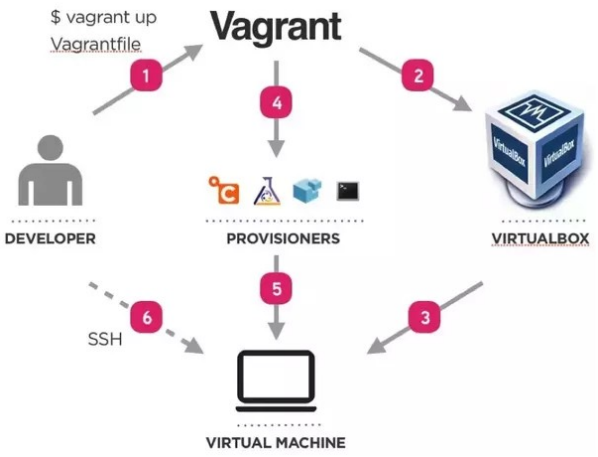
Preparar, configurar, compartir y desplegar entornos de trabajo dentro un proyecto de desarrollo o prueba.

¿Por qué usar Vagrant?

- Vagrant provee una interface de línea de comandos y un lenguaje de configuración que permite fácilmente definir y controlar máquinas virtuales ya sea de forma independiente o la interacción entre las mismas.
- Vagrant permite automatizar procesos de creación y gestión de máquinas virtuales.
- Vagrant dispone de un lenguaje de scripting que permite compartir las configuraciones de cada proyecto, sin necesidad de almacenar las máquinas virtuales completas, ahorrando de

esta forma espacio en el disco.

- Vagrant permite almacenar las configuraciones realizadas en repositorios públicos permitiendo que las mismas se mejoren de manera colaborativa
- Vagrant ayuda a simplificar la infraestructura interna de la empresa, consumiendo menos recursos del datacenter.
- Vagrant permite estandarizar y agilizar la manera de gestionar entornos de desarrollo y demostración.



Conceptos importantes

- **Boxes:** Una box es un paquete que contiene un sistema operativo para un provider específico (Virtual Box normalmente).

  Las boxs pueden ser descargadas de los repositorios públicos o pueden ser creadas a partir de entornos ya existentes.
- **Host:** La máquina/SO Host es la máquina desde la que iniciamos Vagrant.
- **Guest:** La máquina/SO Guest es la máquina virtual que arrancados desde el Host.
- **Provider:** Vendría a ser el target de nuestra instalación. La máquina donde vamos a volcar toda nuestra configuración. En el 99% de los casos será una máquina virtual con VirtualBox aunque vagrant está preparado para utilizar otros providers
- **Provisioners:** Son los métodos que utilizamos para gestionar la máquina virtual desde su arranque, ya sea instalar automáticamente software y/o, modificar configuraciones.

  El método más sencillo es utilizar un script shell como provisioner. Otros sistemas de provisioning que podemos utilizar son Ansible, Chef o Puppet

Requerimiento

Para el correcto funcionamiento de todas las virtudes que posee Vagrant es necesario tener instalado:

- **Virtual Box**: que será el provider de las máquinas virtuales.

- **Vagrant**: el gestor de las máquinas virtuales.

- **Git**: como versionador de configuraciones.

Instalación

La instalación de Vagrant puede ser realizada por medio de los gestores de paquetes de cada distribución Linux o por medio de los paquetes específicos para cada sistema operativo, los cuales pueden obtenerse en **www.vagrantup.com/downloads.html**

Verificación de la versión instalada

```
vagrant –version

vagrant -v
```

Ayuda en línea de comandos

```
Vagrant

vagrant -h

vagrant –help
```

Uso de Vagrant

La gestión de máquinas virtuales considera las siguientes acciones:

- Búsqueda de una Box
- Inicializar una Box
- Añadir una Box
- Ejecutar una Box
- Gestionar una Box
- Provisión de Box

Búsqueda de una Box

Lo primero es disponer de la imagen del box (sistema operativo y aplicaciones) que se ejecutara en la máquina virtual. Existen varios sitios donde existen imágenes de box disponibles para su descarga, algunos ejemplos son:

- `https://vagrantcloud.com/discover`
- `https://app.vagrantup.com/boxes/search`
- `http://www.vagrantbox.es/`

Descarga de imágenes de box específicas:

- `http://files.vagrantup.com/lucid32.box`
- `http://files.vagrantup.com/lucid64.box`
- `http://files.vagrantup.com/precise32.box`
- `http://files.vagrantup.com/precise64.box`

Inicializar una box

La inicialización nos permitirá realizar el manejo de las Boxs y de los proyectos, para esto se deben seguir los siguientes pasos:

Crear directorio de trabajo

```
mkdir mientorno
cd mientorno
```

Inicializar un proyecto

Permite crear un archivo de configuración base para la Box (Vagrantfile).

Inicialización genérica

```
vagrant init miBox
```

Inicialización específica

Esta opción a parte de inicializar permite realizar la descarga automática de la box especificada como parámetro

```
vagrant init ubuntu/trusty64
```

```
$ vagrant init
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

Añadir la Box sin inicializar un entorno

De forma local

```
vagrant box add miBox /ruta/boxes/mientorno.box
```

De forma remota

```
vagrant box add http://servidor/boxes/mientorno.box
```

Haciendo uso del nombre público

```
vagrant box add ubuntu/trusty64
```

Ubicación de las box añadidas

```
~/.vagrant.d/boxes
```

Ejecutar el entorno

Representa ejecutar la máquina virtual con la box seleccionada, esta ejecución puede realizarse de dos formas:

Mediante el provider por defecto

```
vagrant up
```

Mediante un provider específico

```
vagrant up --provider=virtualbox
```

Gestión de máquinas virtuales

Conexión vía ssh a la máquina virtual

```
vagrant ssh
```

Suspender la máquina virtual

```
vagrant suspend
```

Volver a ejecutar la máquina virtual

```
vagrant resume
```

Parar la maquina virtual

```
vagrant halt
```

Eliminar la máquina virtual

```
vagrant destroy
```

Estado de la máquina

```
vagrant status
```

Estado global de las máquinas virtuales

```
vagrant global-status
```

Archivo Vangrantfile

Cada proyecto (entorno) dispone de una archivo que gobierna el comportamiento y las propiedades de un entorno o máquina virtual.

Características de Vangrantfile

- Escrito en Ruby
- Pensado para ser compartido por los desarrolladores o los usuarios del entorno
- Permite crear entornos idénticos
- Puede estar dentro de un sistema de control de versiones.
- Permite estandarizar propiedades de los entornos

Estructura básica de Vangrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
Vagrant.configure("2") do |config|
  # Opciones de configuración
end
```

Opciones de configuración

Vagrant por medio de su Vagrantfile nos permite realizar ciertas configuraciones o personalizar la máquina virtual, entre algunas de las configuraciones disponibles se encuentran:

- Configurar características de hardware
- Seleccionar el sistema operativo
- Nombre del host
- Redirección de puertos
- Configuración de red
- Configurar carpetas compartidas
- Configurar características de hardware

Configuración de la máquina virtual

```
config.vm.provider "virtualbox" do |hw|
  hw.memory = 2048
  hw.cpus = 2
end
```

Sistema operativo o entorno

Permite configurar que sistema operativo y que versión tendrá la máquina virtual

```
config.vm.box = "ubuntu/trusty64"

config.vm.box = "ubuntu/trusty32"

config.vm.box = "centos/7"

config.vm.box = "http://boxes.gajdaw.pl/sinatra/sinatra-v1.0.0.box"
```

Nombre del host

Permite configurar el nombre del host de la máquina virtual, se sugiere que sea un nombre descriptivo.

```
config.vm.hostname = "webserver.atixlibre.org"

config.vm.hostname = "ns1.atixlibre.org"

config.vm.hostname = "db01.atixlibre.org"

config.vm.hostname = "web01.atixlibre.org"
```

Dirección de la box

Permite indicar la ubicación de la imagen de la box base, en la cual se basa la máquina virtual

Desde la URL

```
config.vm.box_url = "http://dominio.com/imagenes/miboxdemo.box"
```

Desde un archivo local

```
config.vm.box_url = "file:///home/boxs_de_vagrant/CentOS-7-x86_64-Vagrant.box"
```

Redireccionamiento de puertos

Permite indicar la relación entre los puertos de la máquina virtual y los de la máquina física.

```
config.vm.network "forwarded_port", guest: 80, host: 8080
config.vm.forward_port 80, 8080
config.vm.forward_port 3306, 3306
```

En el caso de existir alguna colisión de puertos, se puede especificar que exista una autocorrección de los mismos

```
config.vm.forwarded_port 80, 8080, auto_correct: true
```

Esta autocorrección considera por defecto el rango de puertos 2200 al 2250, pero existe la posibilidad de ser personalizado

```
config.vm.usable_port_range = (2200..2250)
```

También existe la posibilidad de poder especificar el tipo de protocolo a ser utilizado

```
config.vm.forwarded_port 80, 8080, protocol: "udp"
```

Configuración de la red

Permite configurar la conexión de red que dispondrá la máquina virtual

IP estática

```
config.vm.network "private_network", ip: "192.168.10.100"
```

IP dinámica

```
config.vm.network "private_network", type: "dhcp"
```

Tipo de Interfaz

```
config.vm.network : bridged
```

Enlace a la interfaz física

```
config.vm.network :public_network,:bridge=>"eth0"
```

Carpetas compartidas

Esta característica permite compartir directorios de archivos entre la máquina física y la máquina virtual, permitiendo compartir archivos y carpetas bajo ciertos parámetros

Incluyendo carpeta origen y carpeta destino

```
config.vm.synced_folder "home/atixlibre/sites", "/var/www/html"
```

incluyendo el propietario y el grupo

```
config.vm.synced_folder "home/atixlibre/sites", "/var/www", owner: "www-data", group:
"www-data"
```

Incluyendo varias carpetas a sincronizar

```
config.vm.synced_folder "home/atixlibre/sitios", "/var/www", owner: "www-data", group:
"www-data"

config.vm.synced_folder "home/atixlibre/documentos", "/home/vagrant/documentos"
```

Otorgando permisos a carpetas y ficheros

```
config.vm.synced_folder "home/atixlibre/public", "/home/vagrant/public", :mount_options
=> ["dmode=777", "fmode=666"]
```

Aprovisionamiento

El aprovisionamiento de una máquina virtual es una tarea fundamental, ya que en esta radica los aspectos que caracterizarán a la box.

El aprovisionamiento es básicamente la ejecución de una serie de acciones para adecuar el funcionamiento de la Box; esta serie de acciones pueden ser escritas en el Vagrantfile en modo “online” o bien en un fichero externo o incluso en un enlace externo.

Aprovisionamiento online

```
config.vm.provision :shell, :inline => "sudo apt-get update"
config.vm.provision "shell", inline: "apt-get install -y nginx"
```

Aprovisionamiento externo

```
config.vm.provision "shell", path: "aprovision.sh"

config.vm.provision :shell, :path => "aprovision.sh"
```

Aprovisionamiento externo mediante URL

```
config.vm.provision "shell", path: "https://scripts.atixlibre.org/aprovision.sh"
```

aprovision.sh

```
#!/usr/bin/env bash
apt-get update
apt-get install -y apache2
```

Entorno de demostración

Como ejemplo demostrativo, realizaremos la configuración de una máquina virtual tomando en cuenta los siguientes aspectos de contexto:

Hardware de la máquina virtual

- Memoria: 1024MB
- Número de CPUs: 2
- Interfaz de red: 192.168.10.100

Software de la máquina virtual

- Sistema operativo: Centos 7

Particularidades

- Actualizar repositorios y paquetes del sistema operativo
- Instalar el servidor de aplicaciones Apache
- Publicar una página de Bienvenida, cuyo código fuente se encuentra en la carpeta **`/home/atixlibre/`** de la máquina física.

Archivo Vangrantfile completo

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
Vagrant.configure("2") do |config|
# Para el caso de una distribucion Centos 7
  config.vm.box = "centos/7"
# Para el caso de una distribucion Debian Stretch
  config.vm.box = "debian/stretch64"
  config.vm.hostname = "webserver.atixlibre.org"
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.network "private_network", ip: "192.168.10.100"
  config.vm.synced_folder "/home/atixlibre/html", "/atixlibre" owner: "vagrant", group:
"www-data", mount_options: ["dmode=775,fmode=664"]
  config.vm.provision "shell", path: "aprovision.sh"
  config.vm.provider "virtualbox" do |hw|
    hw.memory = 1024
    hw.cpus = 2
  end
end
```

Para una distribución Debian

```
aprovision.sh
#!/bin/bash
# Update server
apt-get update
apt-get upgrade -y
# Instalar Apache
apt-get install apache2 -y
#Reiniciar apache
/etc/init/apache2 restart
```

Para una distribución Centos

```
aprovision.sh
#!/bin/bash
# Update server
yum update
yum upgrade -y
# Instalar Apache
yum install httpd -y
# Reiniciar Apache
service httpd restart
```

Agradecimiento

Un profundo y sincero agradecimiento al Ing. Esteban Saavedra, por haberme impulsado y colaborado en la elaboración de este artículo, siendo el coautor del mismo, por su actitud altruista hacia la formación/capacitación y por su incansable labor en impulsar el uso de tecnologías libres en todos los ámbitos.

Referencias

[1] http://www.vagrantup.com

3

Arduino
Aprendiendo Róbotica

Actualmente, el mundo atraviesa por una gran ola de cambios tecnológicos donde el uso de las tecnologías libres se está difundiendo más y más y llegando a limites nunca antes vistos, razón por la cual las personas deben adaptarse y aprender día a día de la realidad que están viviendo.

Una tecnología que ha copado la atención de grandes y pequeños en todos los ámbitos es la utilización de hardware libre, que permite crear entornos automatizados, prototipos, robótica educativa entre otros.

Introducción

Cada día resulta más sencillo fabricar ciertos dispositivos electrónicos en nuestras casas, implementar laboratorios de prueba, automatizar ciertos procesos o dar rienda suelta a nuestra imaginación con inventos que nos ayuden en nuestra vida cotidiana; para lo cual hoy en día existen nuevas herramientas de fácil acceso que nos ayudan en el manejo y aplicación de estas nuevas tecnologías.

Todo esto es posible gracias al trabajo de los investigadores que pertenecen a la comunidad del hardware y software libre, ya que estos comparten sus conocimientos con el mundo sin fines de lucro, para que más y más personas puedan aprender y beneficiarse de la ola tecnológica que estamos atravesando actualmente, y es ahí donde la comunidad de hardware libre y Arduino se hacen presentes, brindándonos plataformas, componentes y diversos dispositivos electrónicos para lograr este cometido.

¿Qué es Arduino?

- Arduino es una comunidad internacional que diseña y manufactura placas de desarrollo de hardware para construir dispositivos que nos permitan controlar objetos del mundo real.
- Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinarios.

- Arduino es plataforma de prototipos de código abierto, donde la principal característica es su sencillez y facilidad de uso.
- Arduino nos proporciona un software altamente potente, y nos brinda la posibilidad de hacer uso de diferentes entornos de desarrollo, que permiten implementar el lenguaje de

programación de Arduino (c++) y el bootloader que es ejecutado en la placa.

¿Qué es el hardware Libre?

- Se puede llamar hardware libre a aquellos dispositivos de hardware cuyas especificaciones y diagramas esquemáticos son de acceso público y libre.
- El uso de hardware libre se asemeja al uso del software libre por sus características de libre uso, estudio, modificación y redistribución.
- El objetivo de la comunidad de hardware libre es crear diseños de aparatos informáticos de forma abierta, de manera que todas las personas puedan acceder y aprender de las mismas.
- Esto viene difundiéndose de forma similar a lo que pasaba en la década de los años 70, cuando los primeros apasionados a los ordenadores construían sus propios equipos y componentes electrónicos y enseñaban al mundo sus avances y aplicaciones en los diferentes sectores del mundo real o de la industria.

¿Qué son los microcontroladores?

- Un microcontrolador es un circuito integrado digital que puede ser usado para muy diversos propósitos. Está compuesto por una unidad central de proceso (CPU), memorias (ROM y RAM), líneas de entrada y salida.
- Un microcontrolador tiene los mismos bloques de funcionamiento básicos, que el de una computadora, lo que nos permite tratarlo como un pequeño dispositivo electrónico, prácticamente funciona como una mini PC.
- Su función es automatizar procesos y procesar información; los microcontroladores se aplican en toda clase de inventos y productos donde se requiere seguir un proceso automático, capaz de ejecutar las órdenes grabadas en su memoria, está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica.

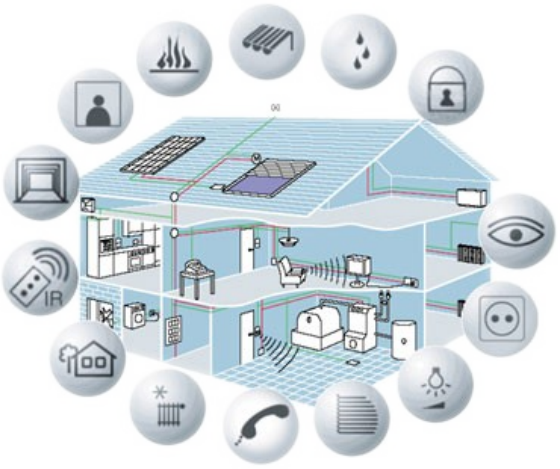
¿Qué puedo hacer con Arduino?

- Con las placas Arduino podemos crear y construir todo tipo de dispositivos electrónicos, desde grandes máquinas hasta pequeños juegos para niños.

- Arduino puede actuar como un sistema autónomo que realice una o varias tareas específicas para las cuales haya sido programado y que las mismas se encuentren almacenadas en su memoria.

- Arduino es una plataforma para programar un microcontrolador y por lo tanto puede hacer todo lo que hace un equipo de cómputo, todo dependerá del alcance de nuestra imaginación.

- Un Arduino puede hacer las tareas de un autómata y programar una aplicación o un SCADA - Supervisory Control And Data Acquisition (Supervisión,Control y Adquisición de Datos) para interactuar con el mundo exterior.



- Con Arduino también podemos crear contextos inteligentes (IoT - Internet of Things), es decir, que los dispositivos que nos rodean y tengan capacidad de computo se puedan conectar a internet, permitiéndonos mandar correos, publicar algún contenido en redes sociales, publicar datos en tiempo real en internet o en un servidor privado, compartir su estado, gestionar operaciones remotamente, entre muchas otras funcionalidades.

- Arduino simplifica el proceso de trabajo con microcontroladores, permitiendo:
  - Desarrollar en un entorno de programación simple y claro.
  - Desarrollo de proyectos de código abierto.
  - Desarrollo de entornos interactivos extensibles y escalables.

Características de Arduino

- Multiplataforma
- Asequible
- Entorno de programación simple y directo
- Sencillo
- Flexible
- Escalable
- Software ampliable
- Funcionalidades ampliables

Características de hardware de una placa Arduino

Existen varias placas Arduino que son utilizadas actualmente en el mundo, esto como una herramienta para poder programar o diseñar en su generalidad. Las características más importantes entre estas placas son las siguientes:

- Microcontrolador: ATmega328
- Voltaje Operativo: 5v
- Voltaje de Entrada (Recomendado): 7 – 12 v
- Pines de Entradas/Salidas Digital: 14 (De las cuales 6 son salidas PWM)
- Pines de Entradas Análogas: 6
- Memoria Flash: 32 KB (ATmega328) de los cuales 0,5 KB es usado por Bootloader.
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Velocidad del Reloj: 16 MHZ.

Entornos de aplicación de Arduino

Los entornos de aplicación que permite Arduino son diversos, entre los cuales se encuentran:

- La enseñanza
- Desarrollo de prototipos
- Desarrollo de entornos interactivos
- Desarrollo de proyectos de automatización y domótica
- Desarrollo de proyectos robóticos.
- Gestión de drones
- Internet de las cosas - IoT

¿Cómo funciona Arduino?

Actualmente se ha vuelto algo común hablar de Arduino y el uso que se le da; esto ya sea en el trabajo o en la escuela, oímos hablar de Arduino de muchas formas, pero no todos saben cómo funciona exactamente.

Arduino es un sistema, y no solo una placa. Por esto, el funcionamiento concreto dependerá del proyecto que se desarrolle, básicamente comprende:

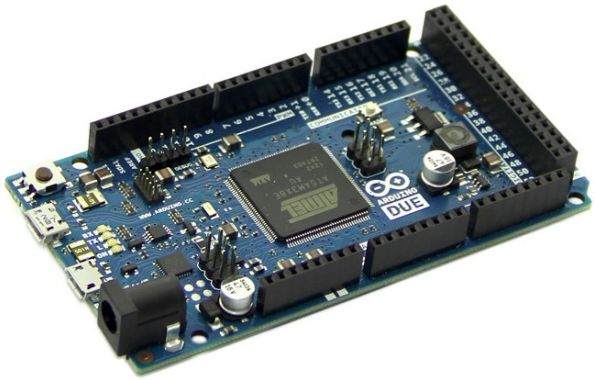
- **Interfaz de entrada**: que puede estar directamente unida a los periféricos , o conectarse a ellos por medio de puertos. El objetivo de esta interfaz es llevar la información al microcontrolador.
- **Interfaz de salida**: que lleva la información procesada por el microcontrolador a los periféricos encargados de hacer el uso final de esos datos.

¿Cuáles son las principales placas de Arduino?

- **Arduino UNO**: Es la placa estándar y la más conocida y documentada. Salió a la luz en septiembre de 2010 sustituyendo su predecesor Duemilanove con varias mejoras de hardware que consisten básicamente en el uso de un USB HID propio en lugar de utilizar un conversor FTDI para la conexión USB. Es 100% compatible con los modelos Duemilanove y Diecimila. Viene con un Atmega328p con 32Kbytes de ROM para el almacenamiento de programas.

- **Arduino MEGA**: Es con mucha diferencia el más potente de las placas con microcontrolador ATmega2560 y el que más pines entrada/salida (I/O) tiene, apto para trabajos algo más complejos aunque tengamos que sacrificar un poco el espacio físico. Cuenta con mayor capacidad de memoria y más pines que el resto de los modelos.

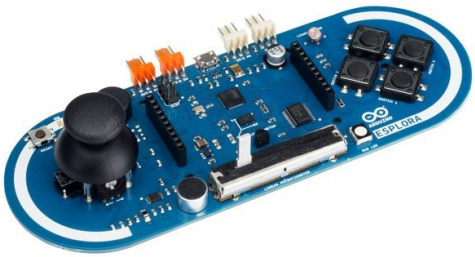
- **Arduino DUE**: Arduino con la mayor capacidad de procesamiento, basado en un microcontrolador AT91SAM3X8E de 32 bit. Este Arduino está alimentado a 3.3V y dado que gran parte de los shields, sensores, actuadores para Arduino y compatible son a 5V lo limita, pero cada vez se ven más elementos donde se puede elegir el voltaje entre 3.3 y 5V.



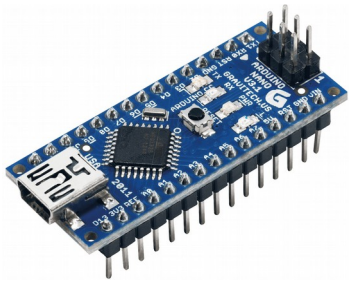
- **Arduino LEONARDO**: La diferencia de este Arduino con el resto es que trae un único MCU ATmega32u4 que tiene integrado la comunicación USB, lo que elimina la necesidad de un segundo procesador. Esto tiene otras implicaciones en el compartimento del Arduino al conectarlo al ordenador, lo que no lo hace apto para iniciarse con él.

- **Arduino ESPLORA**: El Arduino Esplora es una placa electrónica derivada del Arduino Leonardo. Es una placa que de inicio viene integrada con una serie de elementos que facilitan su rápida integración en diversos proyectos, ya que contiene: un sensor de luz, sensor de temperatura, un acelerómetro, un joystick, cuatro botones, un potenciómetro, un led RGB, un zumbador.

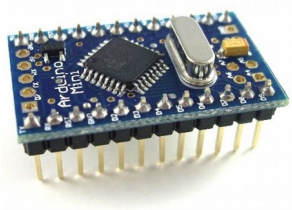
  Es una placa ideal para aquellas personas que deseen probar Arduino sin tener que ocuparse de principio en la electrónica de los elementos que la componen.



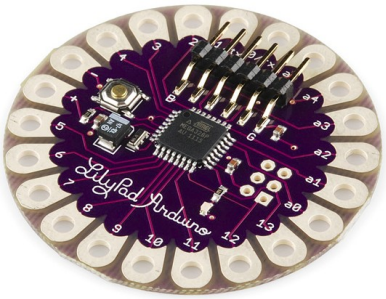
- **Arduino MICRO**: También basado en el ATmega32u4 pero mucho más pequeño, frecuentemente utilizado para proyectos más compactos o de IoT.



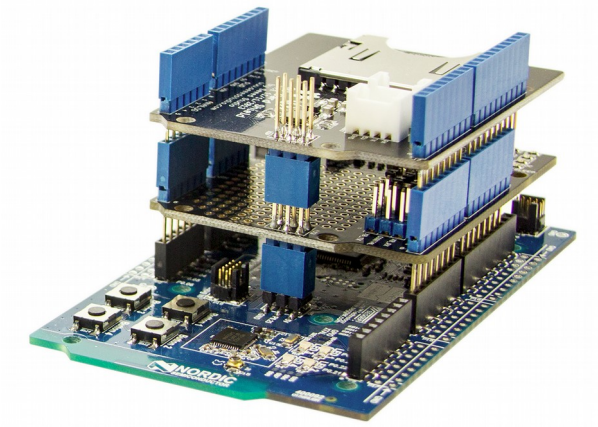
- **Arduino MINI**: Versión miniaturizada de la placa Arduino UNO basado en el ATMega328. Mide tan sólo 30x18mm y permite ahorrar espacio en los proyectos que lo requieran. Las funcionalidades son las misma que Arduino UNO.



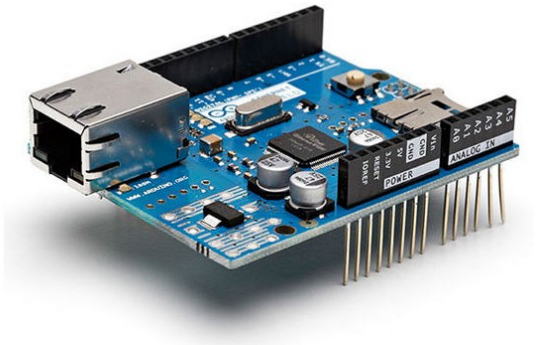
- **Arduino LILYPAD**: Diseñado para dispositivos de computación vestible “wearables” y e-textiles.



- **Arduino YUN**: El Arduino Yun es un Arduino que es diferente a lo que son el resto de Arduino porque además de llevar un microcontrolador, incorpora un Microprocesador MIPS con un Sistema Operativo Linux embebido. La ventaja que aporta Arduino Yun y sus derivados es que el microcontrolador y el microprocesador están conectado mediante un puerto serie y además Arduino nos ofrece una serie de herramientas/librerías que facilita la interconexión entre ellos.



¿Qué son los Shields de Arduino?

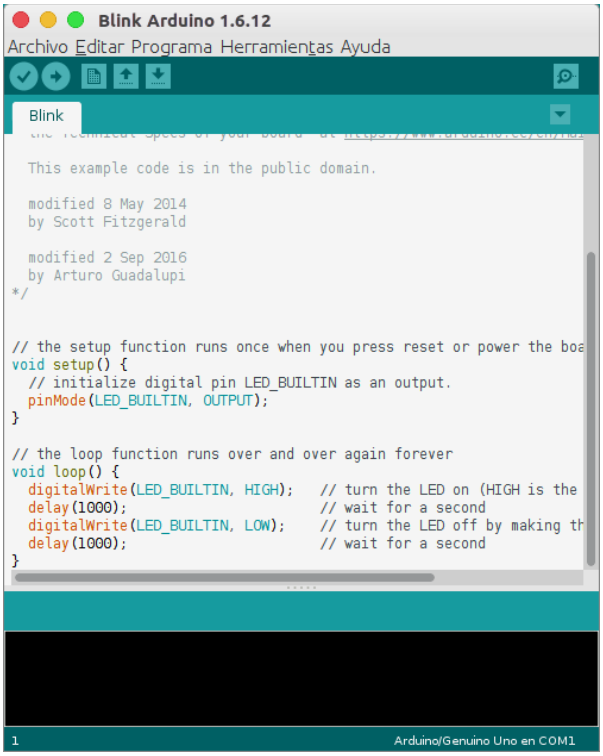


- La plataforma Arduino ha incorporado en su plataforma el uso de placas adicionales con funcionalidades específicas denominadas Shields.
- Las Shields permiten expandir las funcionalidades de la placa base de Arduino, manteniendo la compatibilidad y permitiendo la escalabilidad de entradas y salidas (digitales/analógicas).
- Las shields son placas de circuitos modulares que se montan unas encima de otras para dar funcionalidad extra a un Arduino.
- Las shields se pueden comunicar con el Arduino, bien por algunos de los pines digitales o analógicos o bien por algún bus como el SPI, I2C o puerto serie, así como usar algunos pines como interrupción. Estas shields se alimentan generalmente a través del Arduino mediante los pines de 5V y GND.

Entorno de desarrollo

Actualmente existen varias formas y entornos para poder desarrollar proyectos en Arduino, entre las que se encuentran:

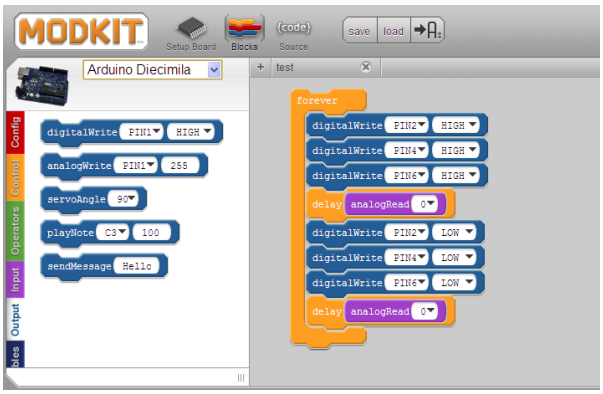
- IDE Arduino



- Scracth for arduino (S4A)

- ArduBlock

- ModKit



Adicionalmente están: Eclipse, STINO, Xcode con Arduino, Atmel Studio, Codebender, entre muchos otros.

Referencias

[1] http://ww.arduino.cc

4 RabbitMQ

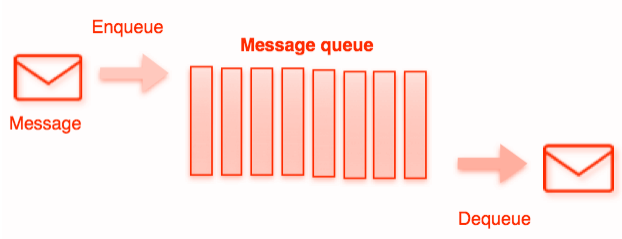
Gestión del Broker de Mensajería

Dentro el desarrollo de un proyecto de software en muchas ocasiones hay que integrarse (acoplarse) con otros actores, componentes, o sistemas internos y externos, siendo necesario enviar o recibir información de ellos. En el mayor de los casos estas comunicaciones tienen que estar permanentemente disponibles, ser rápidas, seguras, asíncronas y fiables entre otros requisitos.

Una de las mejores soluciones de acoplamiento de aplicaciones heterogéneas es el manejo de colas de mensajes.

Introducción

Las colas de mensajes (MQ) solucionan estas necesidades, actuando de intermediario entre emisores y destinatarios, o en un contexto más definido, productores y consumidores de mensajes. Se pueden usar para reducir las cargas y los tiempos de entrega por parte de los servidores de aplicaciones web, ya que las tareas, que normalmente tardarían bastante tiempo en procesarse, se pueden delegar a un tercero cuyo único trabajo es realizarlas.



El uso de colas de mensajes también es bueno cuando se desea distribuir un mensaje a múltiples destinatarios para consumo o para realizar balanceo de carga entre trabajadores.

Beneficios

- **Garantía de entrega y orden**: Los mensajes se consumen, en el mismo orden que llegan, y son consumidos una única vez
- **Redundancia**: Los mensajes en las colas persisten hasta que son procesados por completo
- **Desacoplamiento**: Siendo capas intermedias de comunicación entre procesos, aportan la flexibilidad en la definición de arquitectura de cada uno de ellos de manera separada, siempre que se mantenga una interfaz común
- **Escalabilidad**: Con más unidades de procesamiento, existe la posibilidad de crear más colas y de esta forma balancear su respectiva carga
- **Enrutamiento flexible**: Basado en diferentes reglas de enrutamiento o selección de colas.
- **Clusterización**: Capacidad de crecimiento horizontal
- **Federación**: Provee un acceso común, a un conjunto de colas, dando la posibilidad de poderlas gestionar
- **Alta disponibilidad y Tolerancia a**

fallos: Capacidad de poder disponer redundancia ante posibles fallos

¿Qué es RabbitMQ?

- RabbitMQ es un software de encolado de mensajes llamado broker de mensajería o gestor de colas.
- RabbitMQ es un software de negociación de mensajes y entra dentro de la categoría de middleware de mensajería.
- Es un software donde se pueden definir colas, las aplicaciones se pueden conectar a las mismas y transferir/leer mensajes en ellas.
- RabbitMQ es un sistema gestor de colas de mensajes de código abierto que implementa el estándar **AMQP** (Advanced Message Queuing Protocol).

Que es AMQP

- AMQP - Advanced Message Queuing Protocol
- Es un protocolo pensado para la interoperabilidad
- Es un protocolo completamente abierto
- Es un protocolo binario

Modelo de AMQP

Este modelo posee los siguientes elementos:

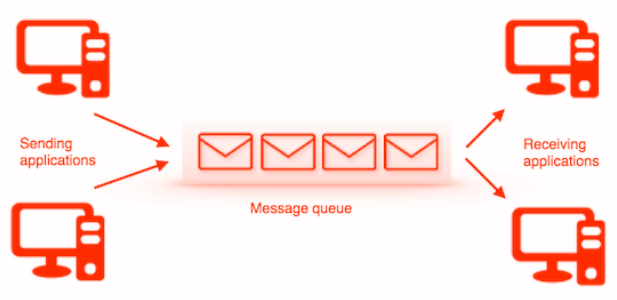
- Intercambiadores (Exchanges)
- Cola de mensajes (messages queues)
- Enlaces (Bindings)
- Reglas de Enrutamiento (Routing Rules)

Características de RabbitMQ

- **Soporte a múltiples protocolos de mensajería**: Directamente o mediante plugins programados, tales como AMQP 0-9-1, AMQP 1-0, STOMP, MQTT o HTTP.
- **Clientes**: Existe una amplia variedad de clientes y conectores desarrollados para permitir la comunicación en una gran variedad de lenguajes tales como Java/JVM, MacOSXx, Amazon Ec2, C/C++, Ruby, Python, Perl, Erlang.
- I**nterfaz de Usuario**: Existe una UI muy completa que permite (entre otras muchas posibilidades), administrar los brokers generados (crear colas, exchanges, usuarios etc.) o incluso consultar el tráfico de mensajes en el sistema en tiempo real.
- Permite agrupar diferentes nodos en un cluster que actúe como único broker lógico, admitiendo incluso políticas de alta disponibilidad y mirroring.
- Permite establecer distintos brokers en una misma máquina, así como generar diferentes virtual hosts en los que ubicarlos.
- Los brokers generados disponen de

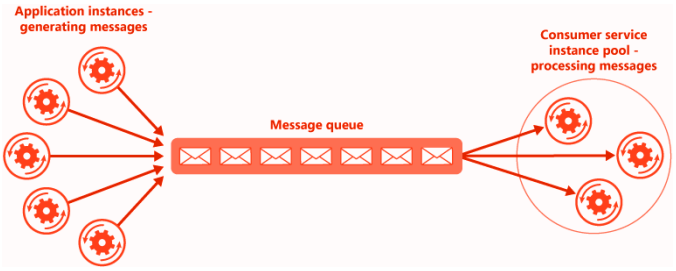
un sistema de logs que informan al administrador de las acciones que se están llevando a cabo sobre el mismo, facilitando la depuración de errores.

¿Cómo funciona?



- Hay aplicaciones clientes, llamadas productores, que crean mensajes y los entregan al broker (cola de mensajes), otras aplicaciones, llamadas consumidores, se conectan a la cola y se suscriben para poder consumir los mensajes y procesarlos.

Características de funcionamiento



- Un mensaje puede incluir cualquier tipo de información.
- Un software puede ser un productor, consumidor, o ambos simultáneamente.
- Los mensajes colocados en la cola se almacenan hasta que el consumidor los recupera o los consume.
- Los mensajes no se publican directamente en una cola, en lugar de eso, el productor envía mensajes a un exchange. Los exchanges son agentes de enrutamiento de mensajes.
- Un exchange es responsable del enrutamiento de los mensajes a las diferentes colas: acepta mensajes del productor y los dirige a colas de mensajes con ayuda de atributos de cabeceras, bindings y routing keys.
- Un binding es un “enlace” que se configura para vincular una cola a un exchange
- Un routing key es un atributo del mensaje, este es utilizado por el exchange para decidir cómo enrutar el mensaje.
- Los exchanges, los enlaces y las colas pueden configurarse con parámetros tales como durable, temporary, y auto delete en el momento de su creación.
  - Los exchanges declarados como durable sobrevivirán a los reinicios del servidor y durarán hasta que se eliminen explícitamente.
  - Aquellos de tipo temporary existen hasta que el Broker se cierre.
  - Por último, los exchanges configurados como auto delete se eliminan una vez que el último objeto vinculado se ha liberado del exchange.

Comunicación con el Broker

Para establecer una comunicación con el broker, es necesario disponer de un usuario que esté registrado en el mismo. RabbitMQ permite establecer limitaciones en los usuarios que genera, bien mediante tags (administrator, policymaker, management o monitoring) o mediante cada virtual host al que este se encuentre asociado.

Conceptos Útiles

- **Producer (Productor)**: Programa que escribe mensajes en un intercambiador (Exchange)
- **Consumer (Consumidor)**: Programa

que escucha mensajes en una cola (Queue)

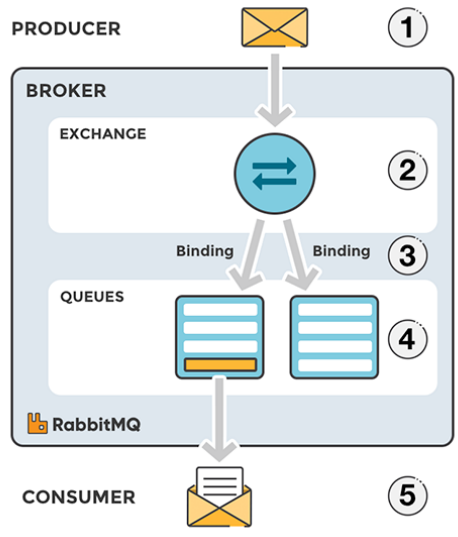
- **Exchange**: Es el punto de entrada de un mensaje.
- **Queue**: Es el punto de lectura de un mensaje.
- **Bindings**: Son reglas que indican cómo llegar de un Exchange a las Queue asociadas.
- **Routing key:** Filtro asociado a un Binging que permite seleccionar sólo algunos mensajes para dicho binding.
- **Virtual host o vhost:** Un entorno aislado, con sus propios grupos de usuarios, exchanges, queues, ...

Tipos de Exchange

RabbitMQ tiene 4 tipos de exchange predefinidos:

- Direct Exchange
- Topic Exchange
- Famout Exhange
- Headers Exchange

Funcionamiento estándar



El funcionamiento básico o estándar que posee RabbitMQ es el siguiente:

1. El productor publica un mensaje para el exchange.
2. El exchange recibe el mensaje y es responsable del enrutamiento del mensaje.
3. Se debe establecer un enlace entre la cola y el exchange. En este caso, tenemos enlaces a dos colas diferentes del intercambio. El intercambio enruta el mensaje a las colas.
4. Los mensajes permanecen en la cola hasta que sean manejados por un consumidor
5. El consumidor maneja el mensaje

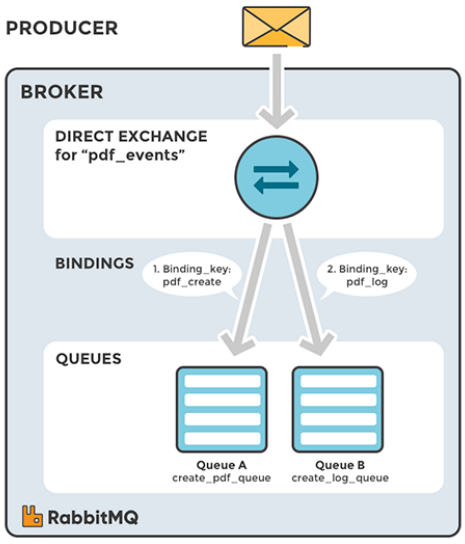
Default Exchange

El default exchange es un intercambio directo pre-declarado sin nombre, generalmente está referido por una cadena vacía "".

Cuando usa el intercambio predeterminado, su mensaje será entregado a la cola con un nombre igual a la clave de enrutamiento del mensaje.

Cada cola se enlaza automáticamente al intercambio predeterminado con una clave de enrutamiento que es igual que el nombre de la cola.

Direct Exchange



Un intercambio directo entrega mensajes a colas basadas en una clave de enrutamiento de mensajes.

La clave de enrutamiento es un atributo del mensaje agregado al encabezado (header) del mensaje por el productor.

La clave de enrutamiento se puede ver como una "dirección" que el exchange está utilizando para decidir cómo enrutar el mensaje. Un mensaje va a la (s) cola (s) cuya clave de enlace coincide exactamente con la clave de enrutamiento del mensaje.

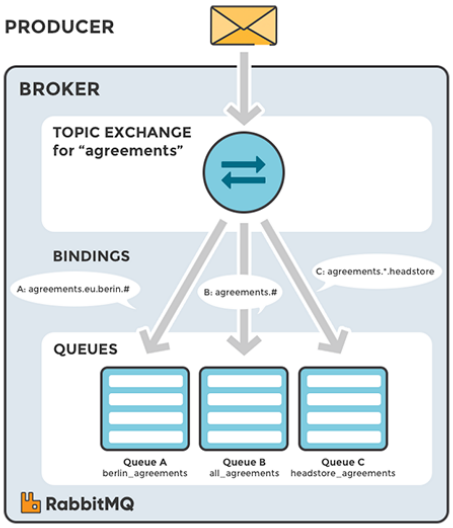
Ejemplo:

Se envía un mensaje con la clave de enrutamiento pdf_log al exchange pdf_events.

Los mensajes se enrutan a pdf_log_queue porque la clave el rounting key (pdf_log) coincide con el binding key (pdf_log).

Si la clave de enrutamiento de mensajes no coincide con ninguna clave de enlace, el mensaje será descartado.

Topic Exchange



Los intercambios temáticos o por tópicos enrutan mensajes a colas basadas en coincidencias de comodines entre la clave de enrutamiento y algo llamado el patrón de enrutamiento.

Los mensajes se enrutan a una o varias colas en función de una coincidencia entre una clave de enrutamiento de mensaje y este patrón.

La clave de enrutamiento debe ser una lista de palabras, delimitada por un punto (.).

Para el ejemplo de la gráfica: agreements.us y agreements.eu.stockholm, identifican los acuerdos que se establecen para una empresa con oficinas en muchos lugares diferentes.

Los patrones de enrutamiento pueden contener un asterisco ("*") para hacer coincidir una palabra en una posición específica de la clave de enrutamiento (por ejemplo, un patrón de enrutamiento de "agreements. *. *. b. *" Solo coincidirá con claves de enrutamiento donde la primera palabra es "agreements" y la cuarta palabra es "b").

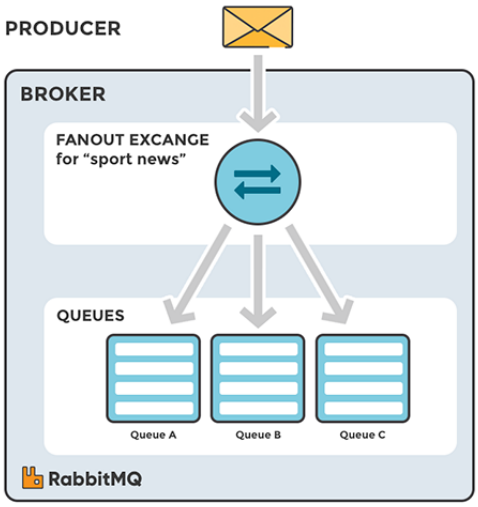
El símbolo "#" indica coincidencia en cero o más palabras (por ejemplo, un patrón de enrutamiento de "agreements.eu.berlin.#" Coincide con cualquier clave de enrutamiento que comience por "agreements.eu.berlin").

Ejemplo

Se envía un mensaje con la clave de enrutamiento agreements.eu.berlin a los acuerdos de intercambio.

Los mensajes se enrutan a la cola berlin_agreements porque el patrón de enrutamiento de "agreements.eu.berlin.#" coincide, asi también se enruta a la cola all_agreements porque la clave de enrutamiento (agreements.eu.berlin) coincide con el patrón de enrutamiento (agreements. #).

Fanout Exchange



El exchange Fanot copia y enruta un mensaje recibido a todas las colas que están vinculadas a él, independientemente de las claves de enrutamiento o la coincidencia de patrones como con los intercambios directos y de temas. Las llaves proporcionadas simplemente serán ignoradas.

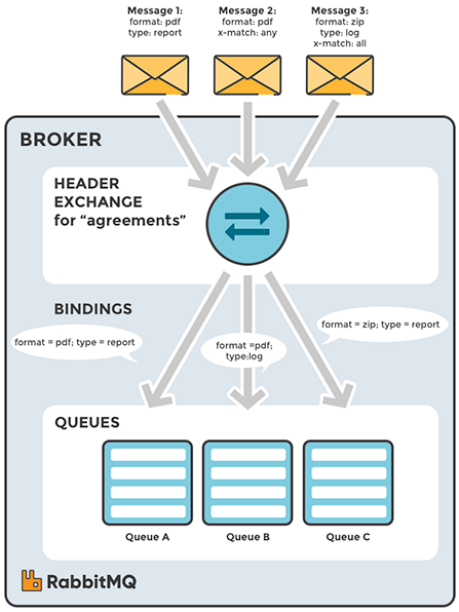
Los intercambios de Fanout pueden ser útiles cuando el mismo mensaje debe enviarse a una o más colas con consumidores que pueden procesar el mismo mensaje de diferentes maneras.

La gráfica muestra un ejemplo en el que un mensaje recibido por el intercambio se copia y enruta a las tres colas que están vinculadas al intercambio.

Ejemplo

Se envía un mensaje al intercambio sport_news. El mensaje se enruta a todas las colas (Cola A, Cola B, Cola C) porque todas las colas están vinculadas al intercambio. Siempre que se ignoren las claves de enrutamiento.

Header Exchange



El header exchange intercambia la ruta en función de los argumentos que contienen encabezados y valores opcionales.

Son muy similares a los Topic Exchange, pero se enruta en función de los valores de encabezado en lugar de las claves de enrutamiento.

Un mensaje se considera coincidente si el valor del encabezado es igual al valor especificado al vincularse.

La propiedad "x-match" puede tener dos valores diferentes: "any" o "all", donde "all" es el valor predeterminado. Un valor de "all" significa que todos los pares de encabezado (clave, valor) deben coincidir y un valor de "any" significa que al menos uno de los pares de encabezado debe coincidir. El argumento "any"es útil para dirigir mensajes que pueden contener un subconjunto de criterios conocidos (desordenados).

Instalación

```
apt install RabbitMQ-server
```

Gestión del servicio

Activar desde el arranque

```
systemctl enable RabbitMQ-server
```

Iniciar el servicio

```
systemctl start RabbitMQ-server
```

Detener el servicio

```
systemctl stop RabbitMQ-server
```

Crear un usuario admin

Por defecto RabbitMQ crea un usuario guest con contraseña guest, pero existe la posibilidad de crear mas usuarios con diferentes atribuciones.

- Crear usuario

```
    RabbitMQctl add_user admin password
```

- Asignar rol de administrador

```
    RabbitMQctl set_user_tags admin administrator
```

- Privilegios sobre un vhost

```
    RabbitMQctl set_permissions -p / admin ".*" ".*" ".*"
```

Interacción con Python

Para el siguiente ejemplo haremos uso de la PIKA una librería de python para el manejo de RabbitMQ

Instalación

```
pip install pika
```

- Conexión

```
conexion = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
```

- Apertura del canal

```
canal = conexion.channel()
```

- Cerrar conexión

```
conexion.close()
```

- Declaración de la cola

```
canal.queue_declare(queue='prueba')
```

- Envío de mensajes

Esta parte puede ser la parte complicada ya que puede establecer el tipo de exchange, la cola de destino, el cuerpo del mensaje. En el ejemplo establecemos que la cola destino por defecto esta establecida por routing_key.

```
canal.basic_publish(exchange='', routing_key='prueba', body='Hola Mundo')
```

productor.py

```
import pika

# Establecer conexión
conexion = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
canal = conexion.channel()

# Declarar la cola
canal.queue_declare(queue='prueba')

# Publicar el mensaje
canal.basic_publish(exchange='', routing_key='prueba', body='Fundación AtixLibre')
print("Mensaje enviado.")

# Cerrar conexión
conexion.close()
```

Ejecución del programa productor

```
$ python productor.py
Mensaje enviado.
```

Verificación del estado de la cola

Debemos recordar que la entrega de mensajes es asíncrona, lo que representa que el mensaje permanecerá en la cola, hasta que algún programa o aplicación la consuma.

Estado de la cola de mensajes

```
$ RabbitMQctl list_queues
Listing queues …
prueba 1
```

Recepción de mensajes

- Suscripción

Antes de poder recibir los mensajes, es necesario realizar una suscripción, la cual nos permita recibir una llamada cada vez que haya un mensaje en cola, este proceso se llama callback.

```
def recepcion(canal, method, properties, body):
    print("Mensaje en cola: %s" % body)
```

- Enlazar con la recepción

```
canal.basic_consume(recepcion, queue='prueba', no_ack=True)
```

donde:

- **queue='prueba'**: Cola a la que se enlaza
- **no_ack=True**: Elimina el mensaje cuando sea servido

- Esperar otros mensajes

```
canal.start_consuming()
```

consumidor.py

```
import pika

# Establecer conexión
conexion = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
canal = conexion.channel()

# Declarar la cola
canal.queue_declare(queue='prueba')

# Definir la función callback
def recepcion(ch, method, properties, body):
    print("Mensaje en cola: %s" % body)

# Enganchar el callback
canal.basic_consume(recepcion, queue='prueba', no_ack=True)

# Poner en espera
print('Esperando mensajes...')
canal.start_consuming()
```

Ejecución del consumidor

```
$ python consumidor.py
Esperando mensajes...
Mensaje en cola: b'Fundación AtixLibre'
```

Estado de la cola

```
$ RabbitMQctl list_queues
Listing queues ...
prueba 0
```

Simulación de un Entorno de Demostración

Para una mejor comprensión realizaremos un conjunto de programas que nos permitan realizar la simulación de la gestión de una cola de mensajes para realizar diferentes operaciones (Registrar, Generar e Imprimir) sobre distintos tipos de documentos (Formulario de Solicitud, Expediente Cliente, Certificado de Inscripción), donde cada uno de estos puede tener cierto número de páginas.

Descripción de Programas

El entorno posee los siguientes programas con las diferentes funciones:

- **productor.py**: Programa orientado a enviar o publicar los diferentes trabajos a la cola de mensajes
- **consumidor.py**: Programa orientado a recuperar los diferentes trabajos que se encuentran en la cola de mensajes
- **operaciondocumento.py**: Programa orientado a simular de forma aleatoria el tipo de documento, el tipo de trabajo a realizar, el numero de paginas de cada tipo de documento y simular la ejecución de cada uno de estos de a cuerdo a las características ya descritas.

- **trabajo.py**: programa orientado a realizar la definición de la clase trabajo, con sus diferentes atributos y los métodos de importar y exportar que permiten serializar los datos del mismo para ser enviados como cuerpo del mensaje a la cola respectiva.

productor.py

```
import pika
from trabajo import Trabajo
from operaciondocumento import documento, operacion, paginas, ejecutar
from random import randint
import time

# Establecer conexión
conexion = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
canal = conexion.channel()

# Declarar la cola
canal.queue_declare(queue='gestion_documentos')

# Bucle infinito
while True:
    # Tiempo aleatorio antes de lanzar el siguiente trabajo
    time.sleep(randint(1,2))

    # Generar un trabajo
    t = Trabajo(operacion(), documento(), paginas())

    # Publicar el mensaje
    canal.basic_publish(exchange='', routing_key='gestion_documentos',
body=t.exportar().encode('utf-8'))
    print("Trabajo enviado a la cola: %s" % t.exportar())
```

consumidor.py

```
import pika
from trabajo import Trabajo
from operaciondocumento import ejecutar

# Establecer conexión
conexion = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
canal = conexion.channel()

# Declarar la cola
canal.queue_declare(queue='gestion_documentos')

# Definir la función callback
def procesar(canal, method, properties, body):
    datos = body.decode('utf-8')
    print("Se ha recibido un trabajo: %s" % datos)
    t = Trabajo.importar(datos)
    ejecutar(t)
    canal.basic_ack(delivery_tag=method.delivery_tag)

# Enganchar el callback
canal.basic_consume(procesar, queue='gestion_documentos')

# Poner en espera
print('Esperando trabajos...')
canal.start_consuming()
```

operaciondocumento.py

```
import time
from random import randint

def documento ():
    a = randint(1,3)
    if a == 1:
       td='Formulario de Solicitud'
    elif a == 2:
       td='Expediente Cliente'
    elif a == 3:
       td='Certtificado de Inscripcion'
    return td

def operacion ():
    b = randint(1,3)
    if b == 1:
       ops='Generar'
    elif b == 2:
       ops='Imprimir'
    elif b == 3:
       ops='Registrar'
    return ops

def paginas():
    return randint(1,10)

def ejecutar(trabajo):
    if trabajo.operacion == 'Generar':
        print('Generando %s ...' % trabajo.documento)
        time.sleep(2)
        print('Hecho!')
    elif trabajo.operacion == 'Imprimir':
        print('Imprimiendo %s ...' % trabajo.documento)
        time.sleep(2)
        print('Hecho!')
    elif trabajo.operacion == 'Registrar':
        print('Registrando %s ...' % trabajo.documento)
        time.sleep(2)
        print('Hecho!')
    else:
        raise NotImplementedError('Operación "%s" no soportada.' % trabajo.operacion)
```

trabajo.py

```
import json

class Trabajo:
    def __init__(self, operacion, documento, paginas=None):
        self.operacion = operacion
        self.documento = documento
        self.paginas = paginas

    def exportar(self):
        return json.dumps(self.__dict__)

    @classmethod
    def importar(cls, datos):
        dic = json.loads(datos)
        return cls(dic['operacion'], dic['documento'], dic['paginas'])
```

Ejecución del productor.py

```
$ python productor.py
Trabajo enviado a la cola: {"operacion": "Generar", "documento": "Formulario de
Solicitud", "paginas": 6}
Trabajo enviado a la cola: {"operacion": "Imprimir", "documento": "Formulario de
Solicitud", "paginas": 2}
Trabajo enviado a la cola: {"operacion": "Imprimir", "documento": "Certtificado de
Inscripcion", "paginas": 8}
Trabajo enviado a la cola: {"operacion": "Generar", "documento": "Formulario de
Solicitud", "paginas": 9}
Trabajo enviado a la cola: {"operacion": "Imprimir", "documento": "Formulario de
Solicitud", "paginas": 9}
Trabajo enviado a la cola: {"operacion": "Registrar", "documento": "Certtificado de
Inscripcion", "paginas": 8}
Trabajo enviado a la cola: {"operacion": "Generar", "documento": "Expediente Cliente",
"paginas": 3}
Trabajo enviado a la cola: {"operacion": "Registrar", "documento": "Formulario de
Solicitud", "paginas": 7}
Trabajo enviado a la cola: {"operacion": "Imprimir", "documento": "Expediente Cliente",
"paginas": 7}
Trabajo enviado a la cola: {"operacion": "Generar", "documento": "Expediente Cliente",
"paginas": 7}
Trabajo enviado a la cola: {"operacion": "Registrar", "documento": "Formulario de
Solicitud", "paginas": 9}
Trabajo enviado a la cola: {"operacion": "Registrar", "documento": "Expediente
Cliente", "paginas": 7}
```

Ejecución del consumidor.py

```
$ python consumidor.py
Esperando trabajos...
Se ha recibido un trabajo: {"operacion": "Generar", "documento": "Formulario de
Solicitud", "paginas": 6}
Generando Formulario de Solicitud ...
Hecho!
Se ha recibido un trabajo: {"operacion": "Imprimir", "documento": "Formulario de
Solicitud", "paginas": 2}
Imprimiendo Formulario de Solicitud ...
Hecho!
Se ha recibido un trabajo: {"operacion": "Imprimir", "documento": "Certtificado de
Inscripcion", "paginas": 8}
Imprimiendo Certtificado de Inscripcion ...
Hecho!
Se ha recibido un trabajo: {"operacion": "Generar", "documento": "Formulario de
Solicitud", "paginas": 9}
Generando Formulario de Solicitud ...
Hecho!
Se ha recibido un trabajo: {"operacion": "Imprimir", "documento": "Formulario de
Solicitud", "paginas": 9}
Imprimiendo Formulario de Solicitud ...
Hecho!
Se ha recibido un trabajo: {"operacion": "Registrar", "documento": "Certtificado de
Inscripcion", "paginas": 8}
Registrando Certtificado de Inscripcion ...
Hecho!
Se ha recibido un trabajo: {"operacion": "Generar", "documento": "Expediente Cliente",
"paginas": 3}
Generando Expediente Cliente ...
Hecho!
Se ha recibido un trabajo: {"operacion": "Generar", "documento": "Certtificado de
Inscripcion", "paginas": 5}
Generando Certtificado de Inscripcion ...
Hecho!
```

Administración

RabbitMQ, posee la posibilidad de realizar la administración del mismo desde consola o desde su administrador web, para este último se debe habilitar el plugin **`RabbitMQ_management,`** el cual permite acceder a una web de administración que también tiene su propio api rest. Los usuarios de esta herramienta administrativa o del api, pueden tener distintos roles. Por defecto trae un usuario guest con el mismo password. Con este usuario inicial, se pueden crear usuarios administrativos con cada uno de los roles permitidos para acceder a la administración.

Habilitar interfaz web

RabbitMQ tiene la posibilidad de poder hacer uso de un conjunto de plugins, con el objetivo de dotarle de posibilidades adicionales; un plugin bastante interesante es el que permite habilitar una interfaz web para su administración.

```
RabbitMQ-plugins enable RabbitMQ_management
The following plugins have been enabled:
  mochiweb
  webmachine
  RabbitMQ_web_dispatch
  amqp_client
  RabbitMQ_management_agent
  RabbitMQ_management

Applying plugin configuration to rabbit@zeus... started 6 plugins
```

Dirección de acceso

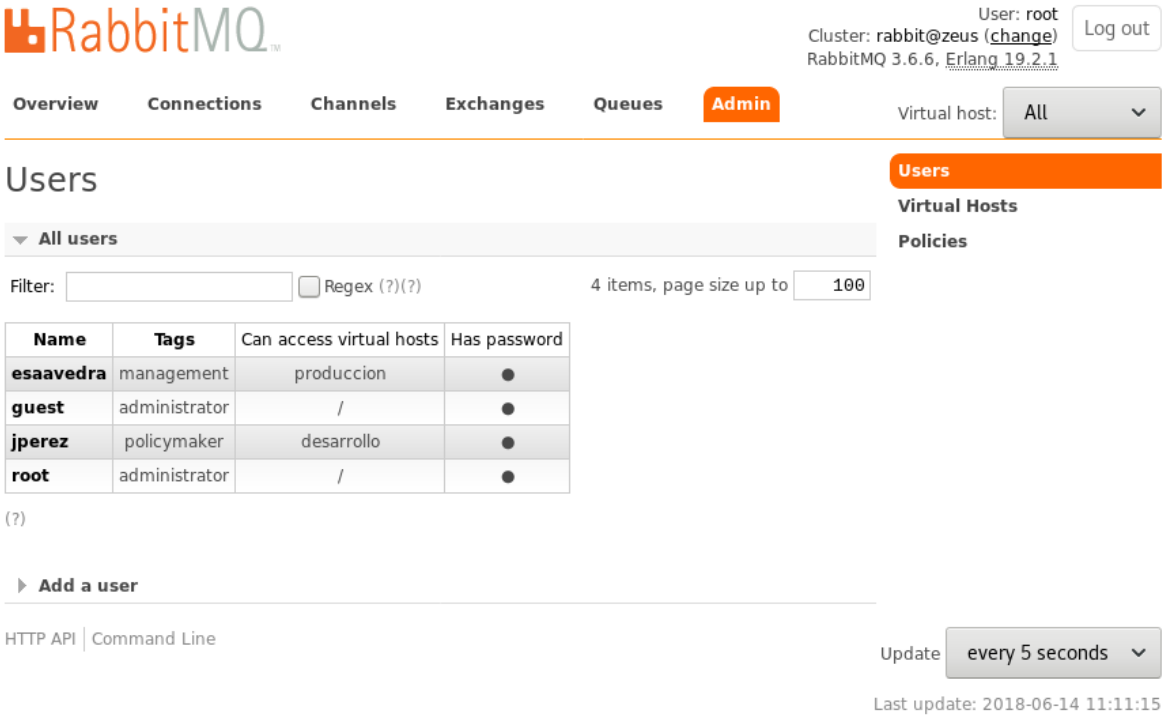
```
http://localhost:15672/
```

Descripción de la Interfaz web de Administración

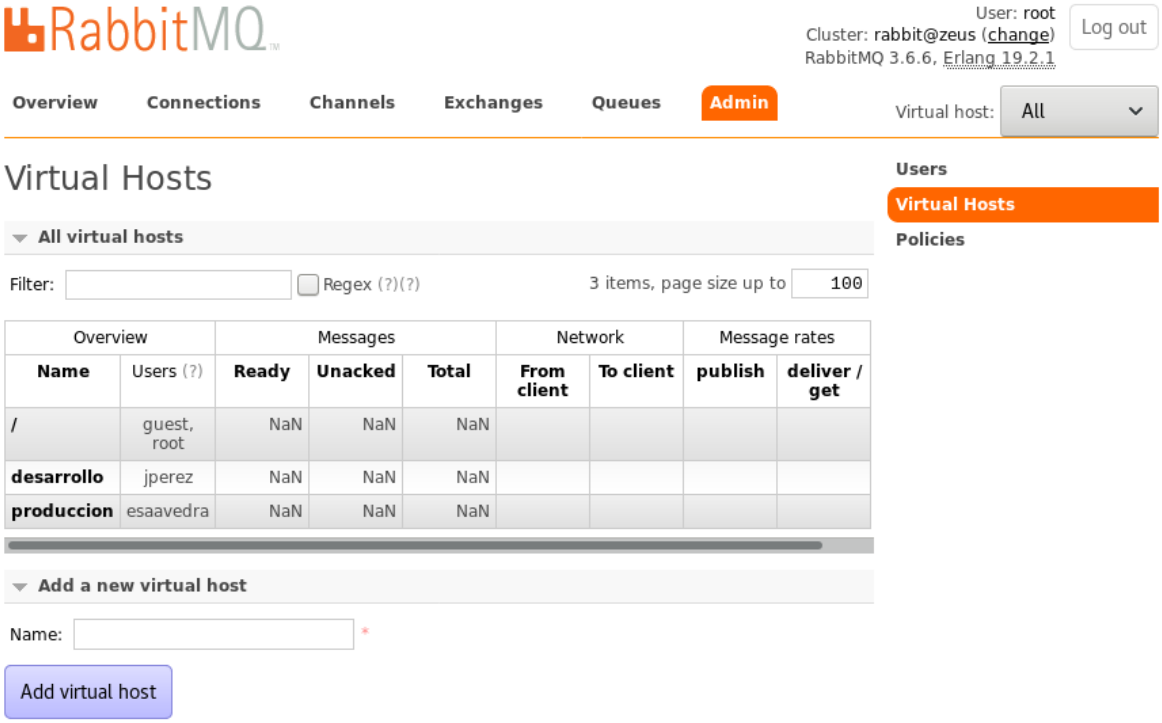
Página de Acceso

Pagina de Administración

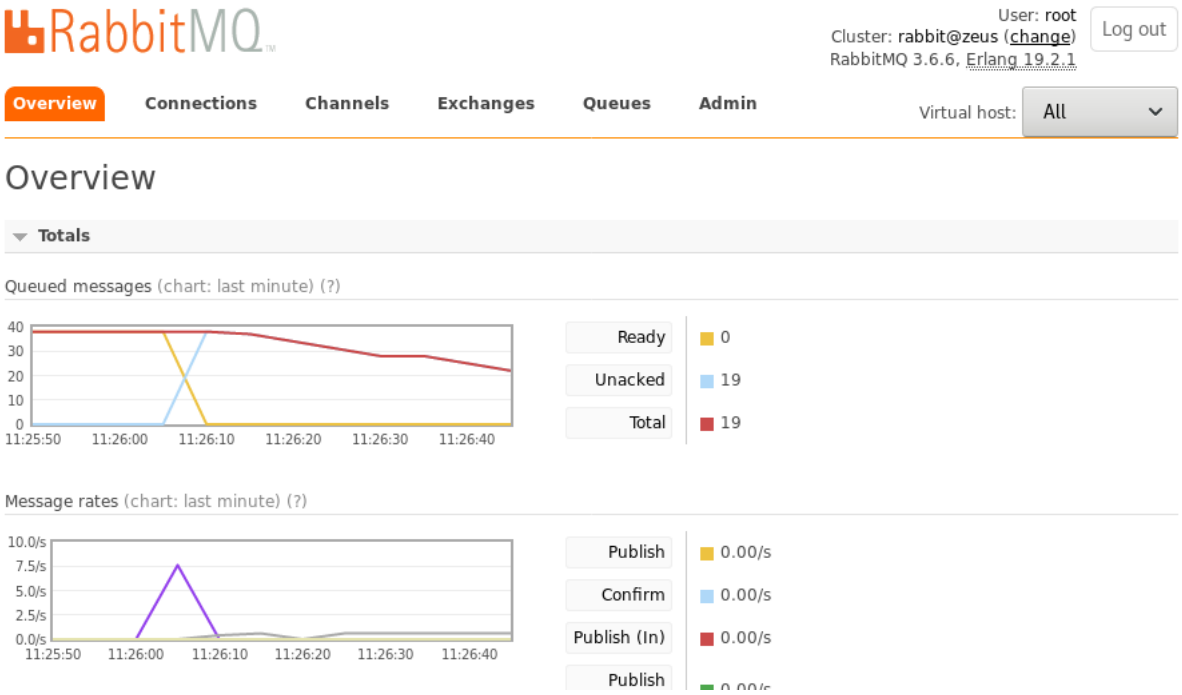
Permite la Gestión de usuarios, virtualhosts y políticas de acceso



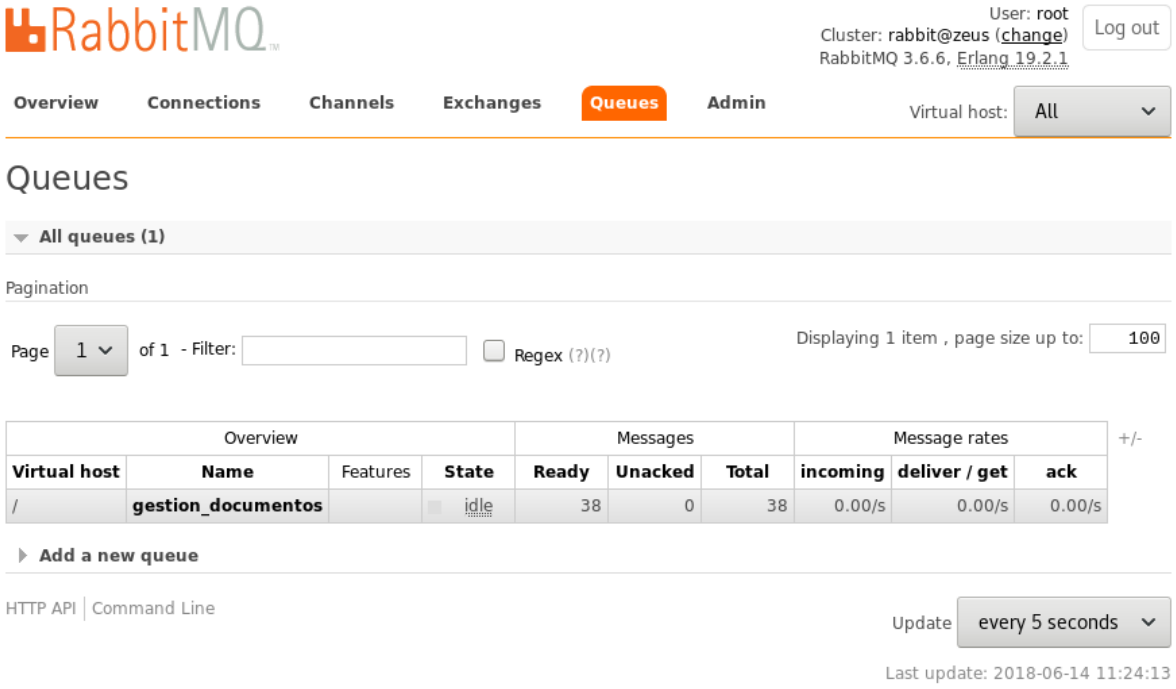
Gestión de Virtualhosts



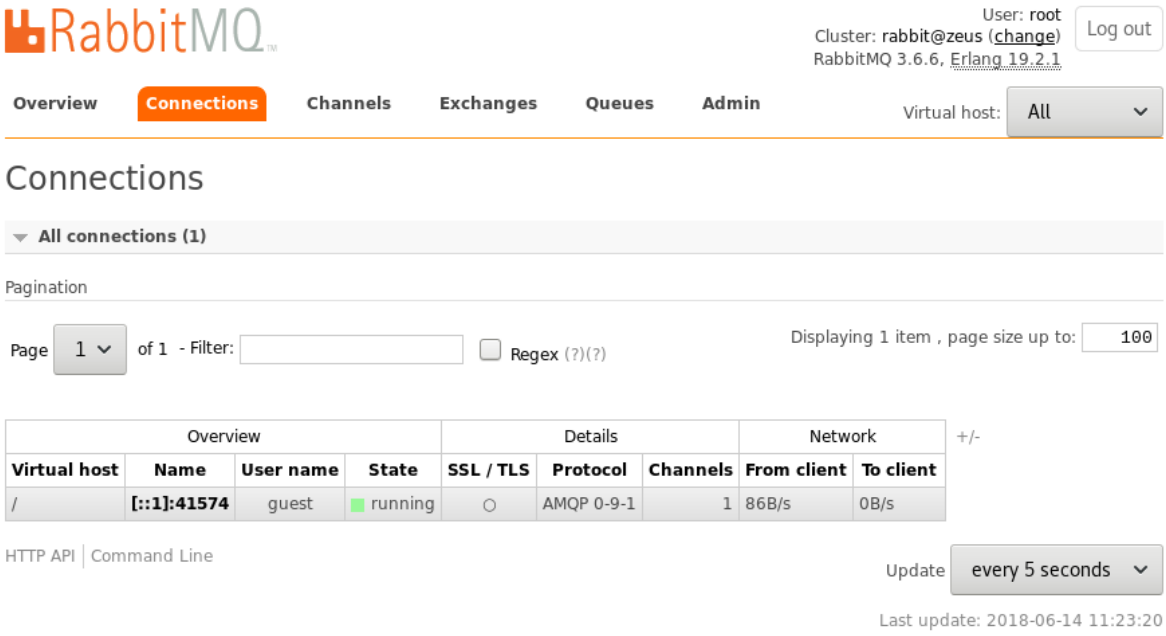
Dashboard principal, que permite mostrar de forma gráfica, lo que acontece con el broker en tiempo real.



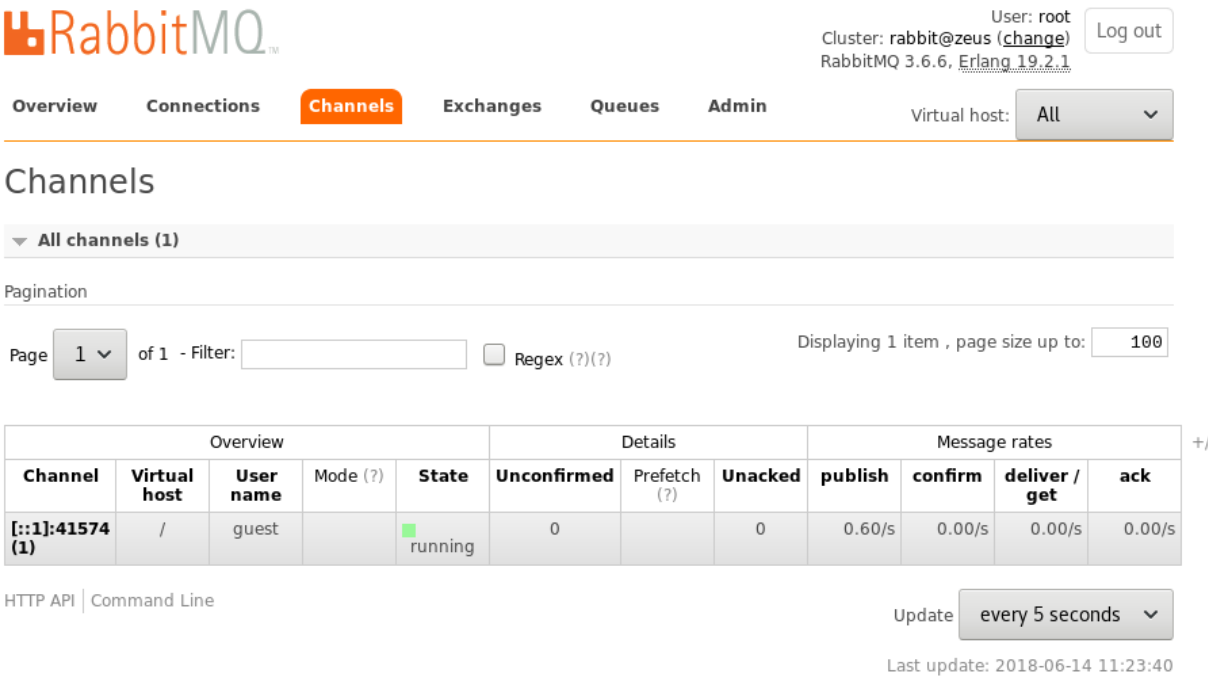
Detalle de colas en funcionamiento



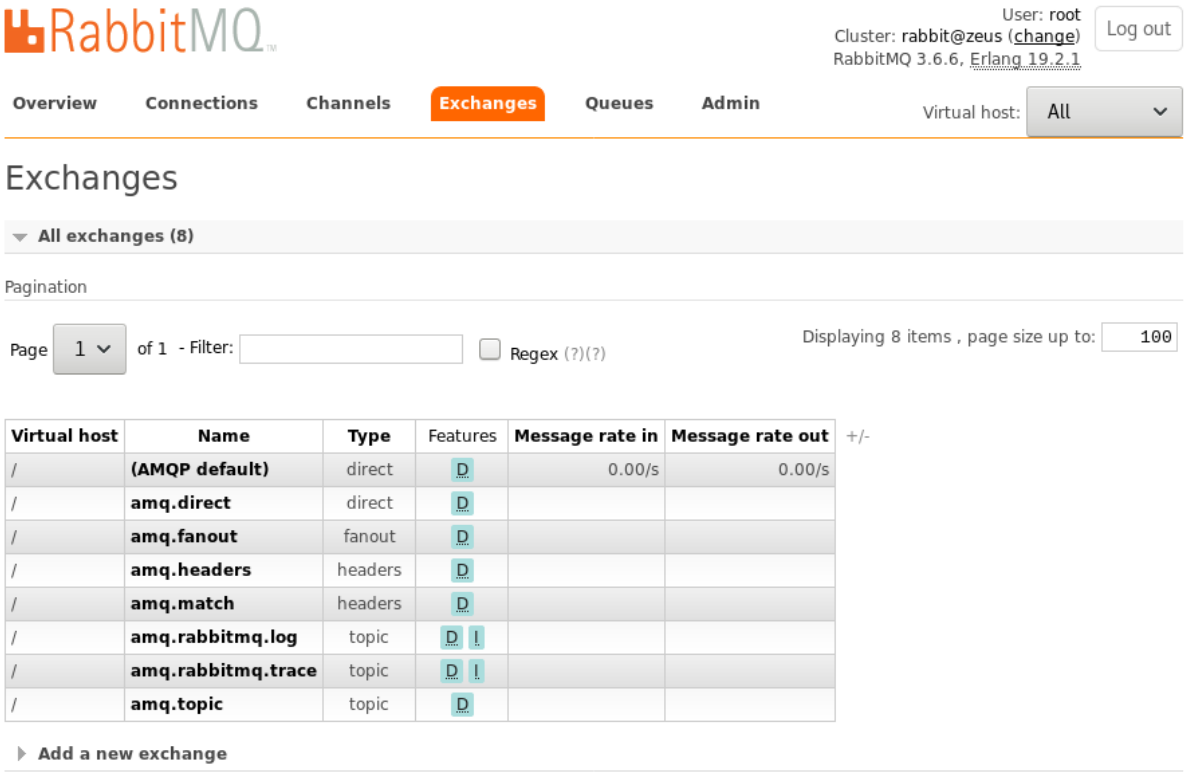
Detalle de las conexiones en uso



Detalle de canales que se hacen uso



Tipos de exhange disponibles



Conclusiones

Como se ha podido observar, son grandes y variadas las aplicaciones que se pueden dar a un gestor de colas de mensajes o broker como RabbitMQ, entre las que destacan:

- Uso de un broker de mensajes para la interoperabilidad o acoplamiento de aplicaciones, sin importar el lenguaje en que hayan sido desarrolladas cada una de éstas.
- En aplicaciones donde el tiempo de demora de ciertas operaciones no debería interferir en el normal flujo de la aplicación, teniendo en cuenta que estas operaciones pueden ser ejecutadas de forma asíncrona con la ayuda de broker de mensajes.

Referencias

[1] http://www.RabbitMQ.org

5 TCO Tools

Cálculo del Costo Total de Propiedad

En una empresa es muy relevante tener claro el costo total de los Sistemas Informáticos (TCO), que son la suma de los costos de inversión de hardware y software, los gastos de implementación y los gastos recurrentes, esto nos permite medir la variable costo a lo largo del tiempo para la óptima gestión de los Sistemas Informáticos o las Tecnologías de la Información y Comunicación (TIC).

Introducción

Gartner define el costo total de propiedad (TCO) en una evaluación integral de la tecnología de la información (TI) u otros costos a través de los límites de la empresa a lo largo del tiempo. Para TI, TCO incluye adquisición, administración y soporte de hardware y software, comunicaciones, gastos del usuario final y el costo de oportunidad del tiempo de inactividad, capacitación y otras pérdidas de productividad.

La herramienta TCO-Tool nos permite realizar el cálculo del TCO y tener diferentes reportes para el análisis del ciclo de vida de productos y/o servicios desde la perspectiva de costos.

Entre las características de TCO.Tool están:

- GNU Lesser General Public License.
- Modelado del TCO con 4 componentes:
  - Información Base de Hardware, Software, Comunicaciones, etc.
  - Control, parámetros del sistema que se utilizarán para clasificar y ampliar la información básica y para el cálculo de los costos.
  - Modelo de Costos, algoritmo para el cálculo del TCO en base a la Información Base y Control.
  - Reportes, informes en formato de texto o gráficos.
- Reportes:
  - General.
  - Dependencias de costos.
  - Costos por tipo de categoría.
  - Costos directos e indirectos.
  - Costos estimados.
  - Informes contables y financieros.
  - Plan de presupuesto.
  - Depreciaciones.

Instalación y ejecución

Para la versión TCO-Tool V1.5.1 es necesario como prerrequisito la instalación de Java SE 6, como versión mínima de java.

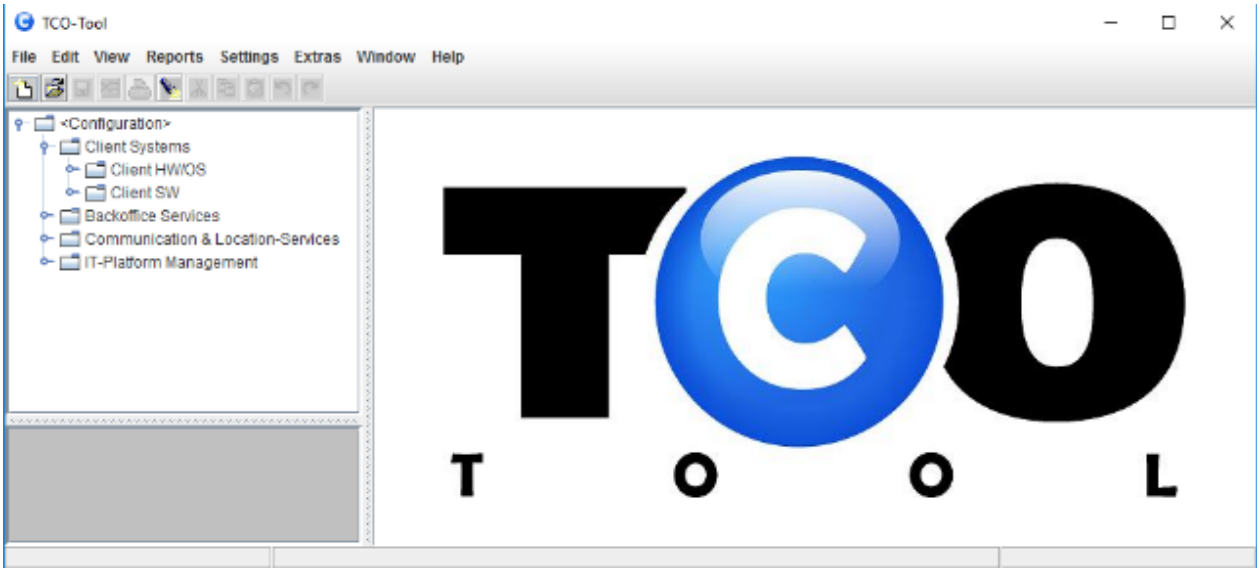
TCO-Tool se lo puede bajar de **`http://sourceforge.net/projects/tcotool`**.

Para la instalación simplemente se debe descomprimir el binario de java de TCO-Tool en una carpeta y para la ejecución del programa jar se debe ejecutar en la carpeta del Java SE 6 el siguiente comando:

```
#/usr/lib/jvm/java/bin/java -Xmx512m -jar /home/V1.5.1/TCO_Tool.jar
```

Donde, la carpeta de instalación del Java es **`/usr/lib/jvm/java/bin`** y la carpeta donde se descomprimió el jar de TCO-Tool es **`/home/V1.5.1`**.

La pantalla java que se abrirá al ejecutar el comando es:



Ejemplo

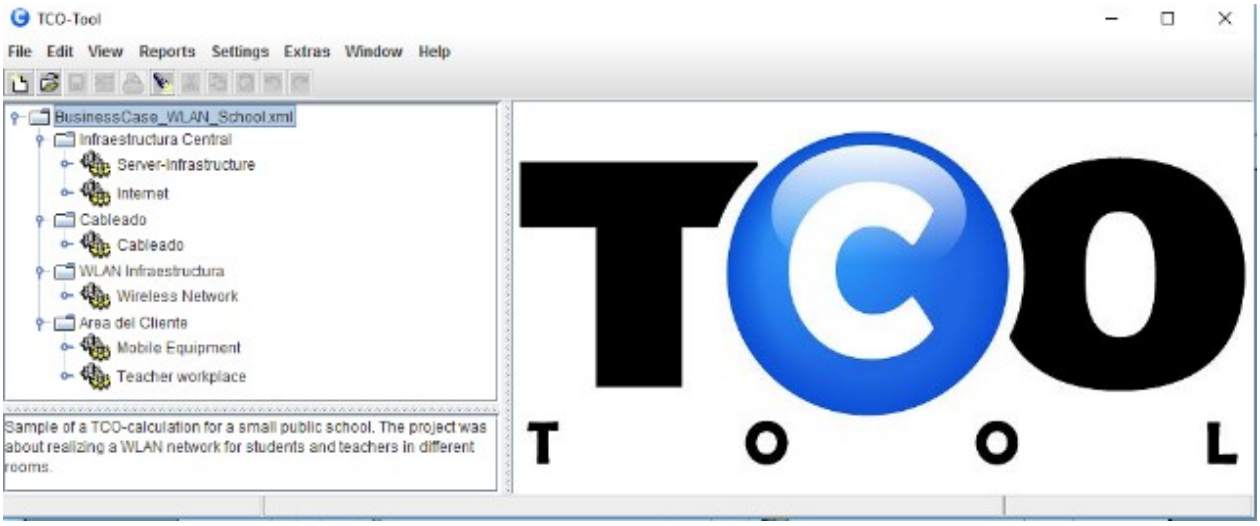
A continuación, explicamos en un ejemplo el costo total de una WLAN para una Escuela Pública con las siguientes características:

1. Edificio: La estructura física del edificio de la escuela pública es de 1 sótano subterráneo y 3 plantas superiores con diferentes salas para clases del tamaño de aproximadamente 20 estudiantes cada una.
2. Equipo WLAN específico: Cada piso está equipado con 6 a 8 puntos de acceso (AP) con un total de 20 AP. Las antenas de cada punto de acceso están específicamente aseguradas y escondidas en las paredes. Cada punto de acceso está conectado por Ethernet a un panel de conexión principal (Rack con conectores RJ-45) en el sótano (configuración de estrella). Cada sala contiene también una PC para el maestro.
3. Equipo servidor: El sótano contiene un enrutador (DHCP) y un firewall conectado a Internet.
4. Estaciones móviles: También hay un equipo móvil que consta de 2 dispositivos móviles

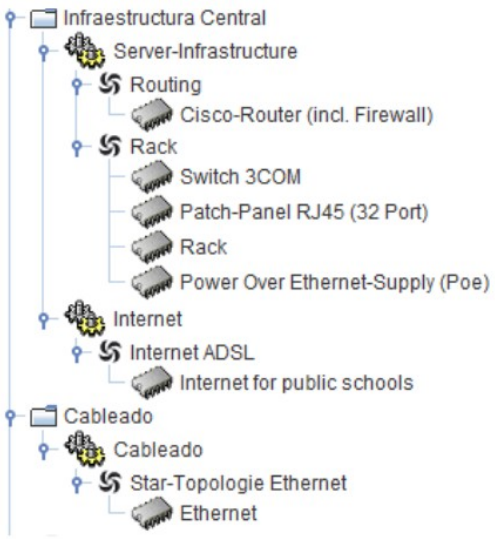
de acceso y 24 Notebooks para ser distribuidos entre los estudiantes.

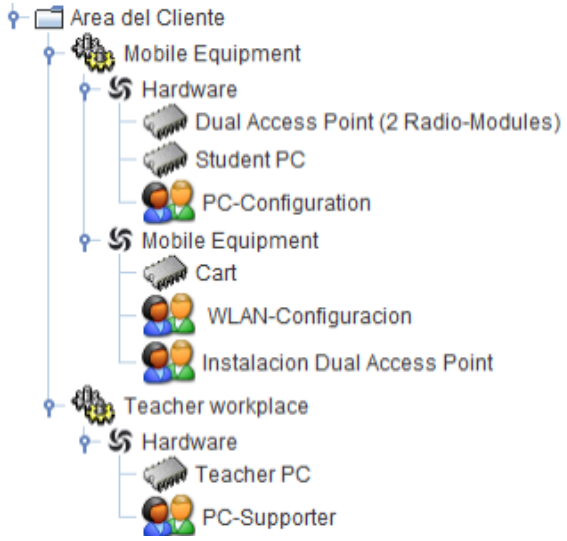
5. Configuración de software: 1 Subred mapea todas las PC de Cliente. Gestión de perfiles de estudiantes, autenticación al iniciar sesión (IAS) y descarga de perfil. Encriptación WPA a través de la conexión inalámbrica.
6. Costos adicionales: Enlace de Internet. Un sistema de enfriamiento en la sala de servidores. Un UPS para respaldo de la energía eléctrica.

En la herramienta se crearon los siguientes 4 grupos: Infraestructura Central, Cableado, WLAN Infraestructura y Área del Cliente.

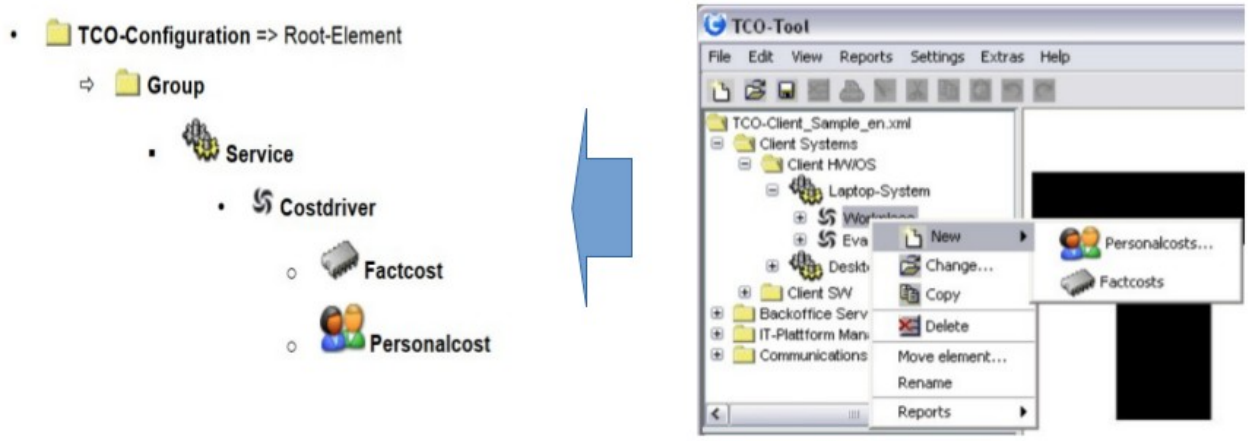


Dentro de cada uno de los 4 grupos se crearon los siguientes componentes:



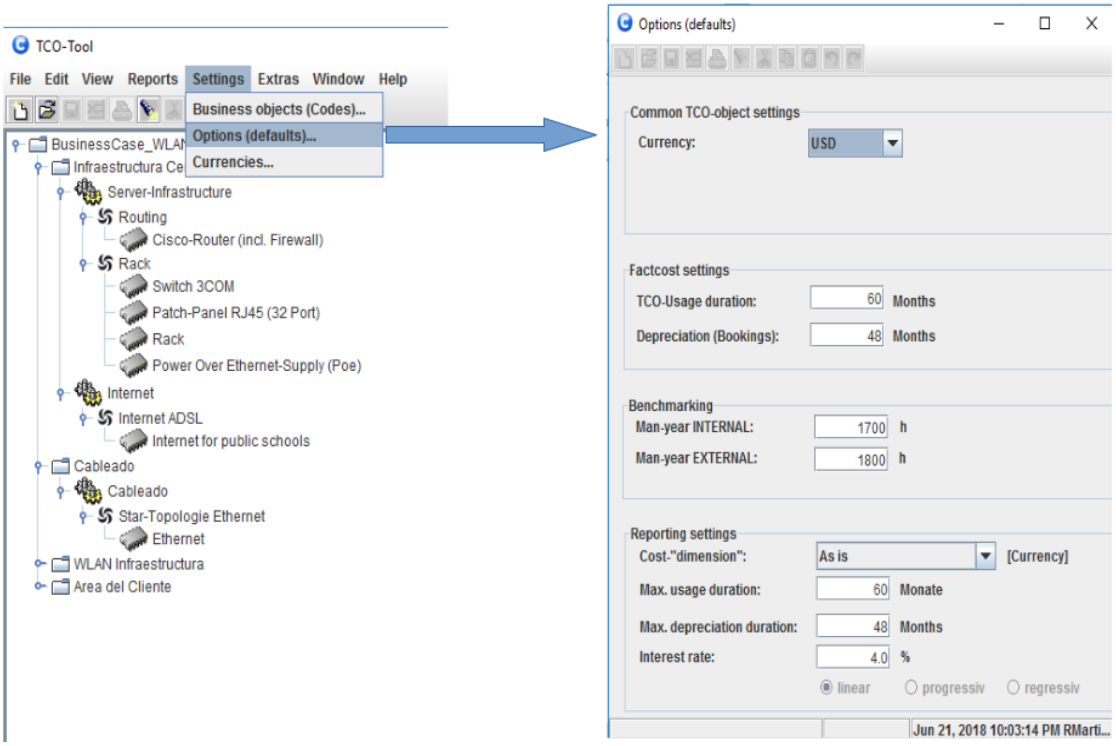


Los componentes de cada grupo se dividen en diferentes objetos los cuales se van añadiendo con click derecho en cada objeto:

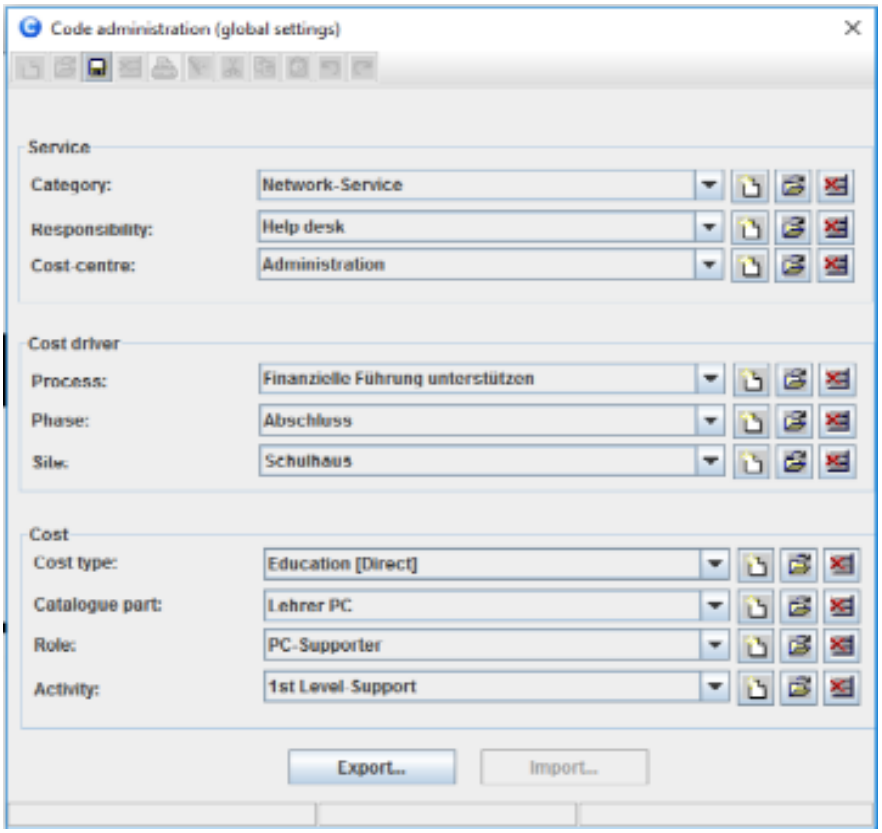


La configuración de las variables a utilizar se las realiza a partir del menú SETTINGS:

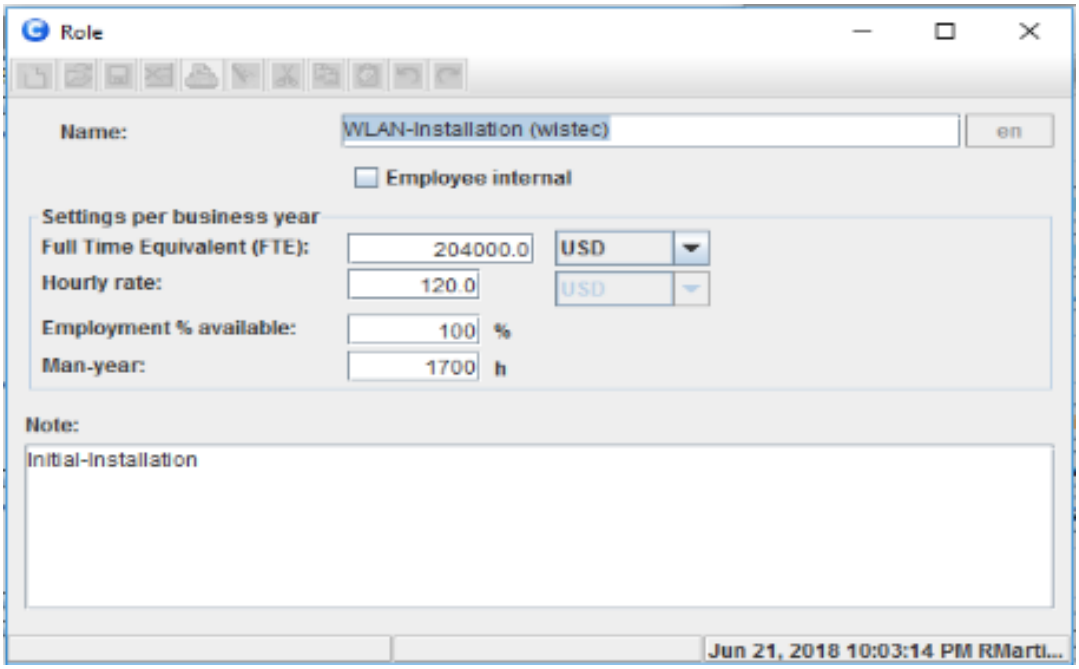
1. Primero debemos setear las variables de OPTIONS (DEFAULTS) donde se definen la moneda para los cálculos financieros, duración del tiempo para el cálculo del TCO, tiempo de la depreciación de las inversiones, costos de personal interno o externo y los valores para la generación de reportes.



2. Las variables de los objetos a utilizar dentro los grupos y sus componentes son definidos en el menú de BUSINESS OBJECTS, donde se definen los valores para cada tipo de objeto: Service, Cost Driver (Fact y Personal).



Por ejemplo, si deseamos setear los valores del Role de WLAN-Installation:

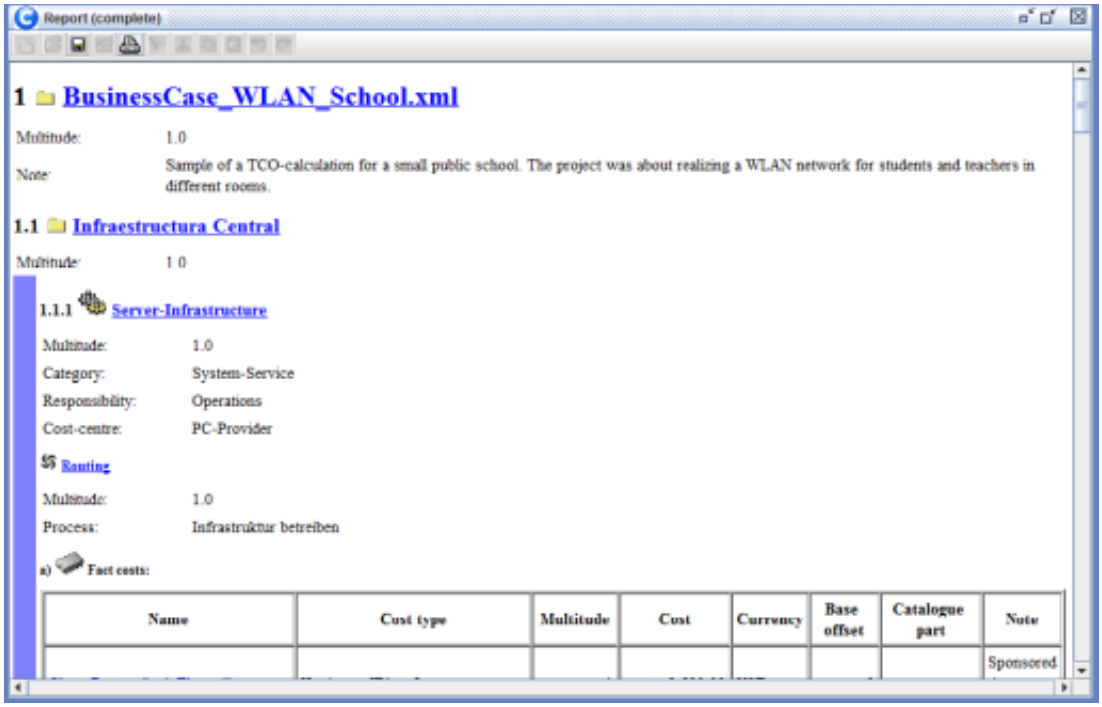


De esta misma forma se van llenando todos los datos en cuanto a tiempos y costos de los diferentes objetos que conforman el Modelo.

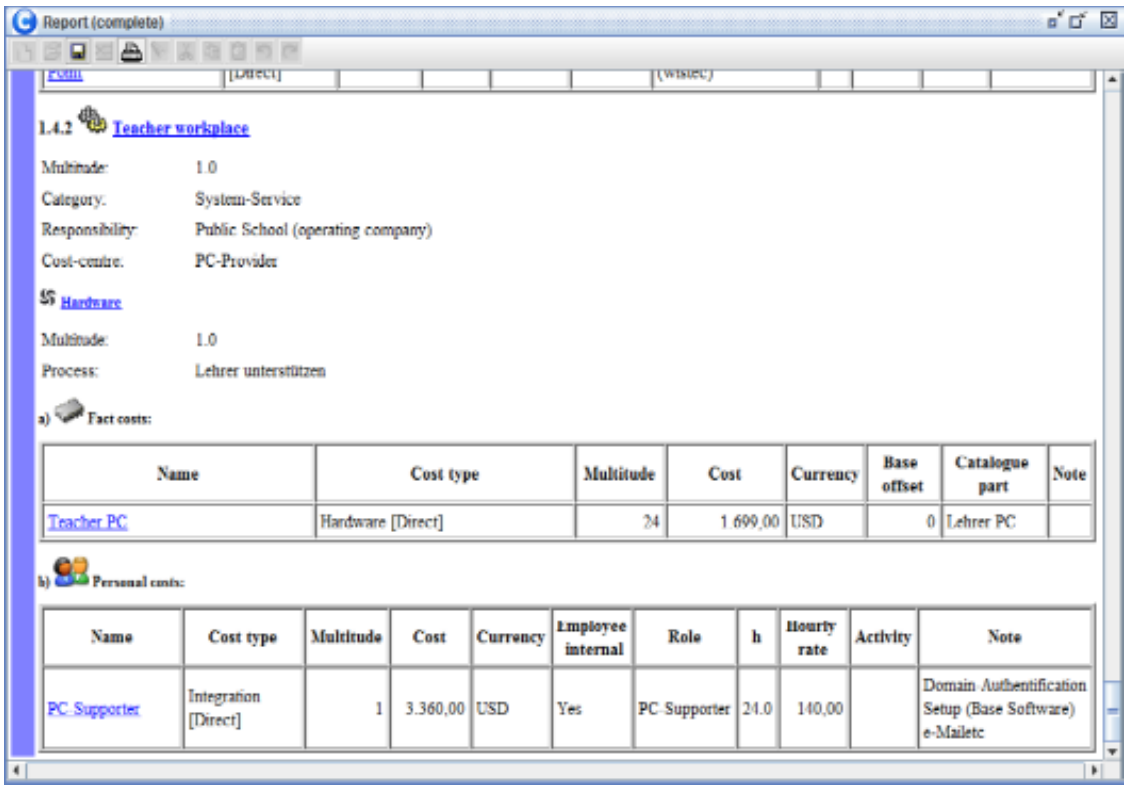
Reportes

Como parte de este ejemplo veremos los reportes más relevantes.

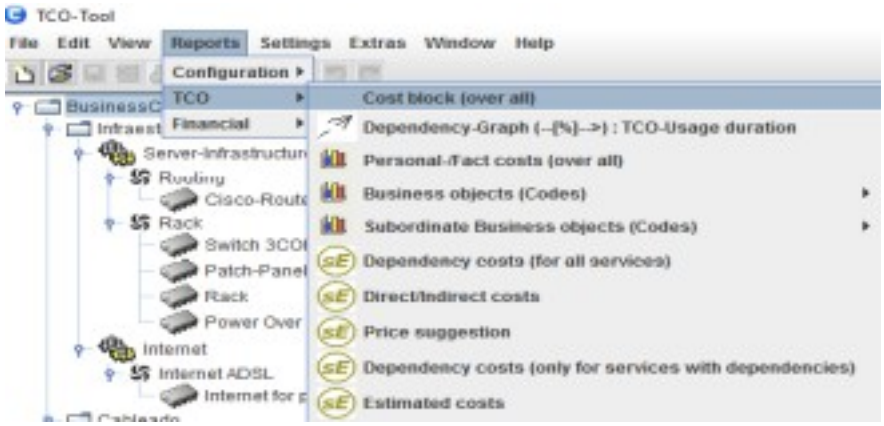
Reporte Completo



En este reporte se puede observar los costos a detalle de cada objeto de los 4 Grupos, como por ejemplo del Grupo TEACHER WORKPLACE:



Reporte TCO – Cost Block (oVer All)



Cost block (over all)

By <Cost-centre>:

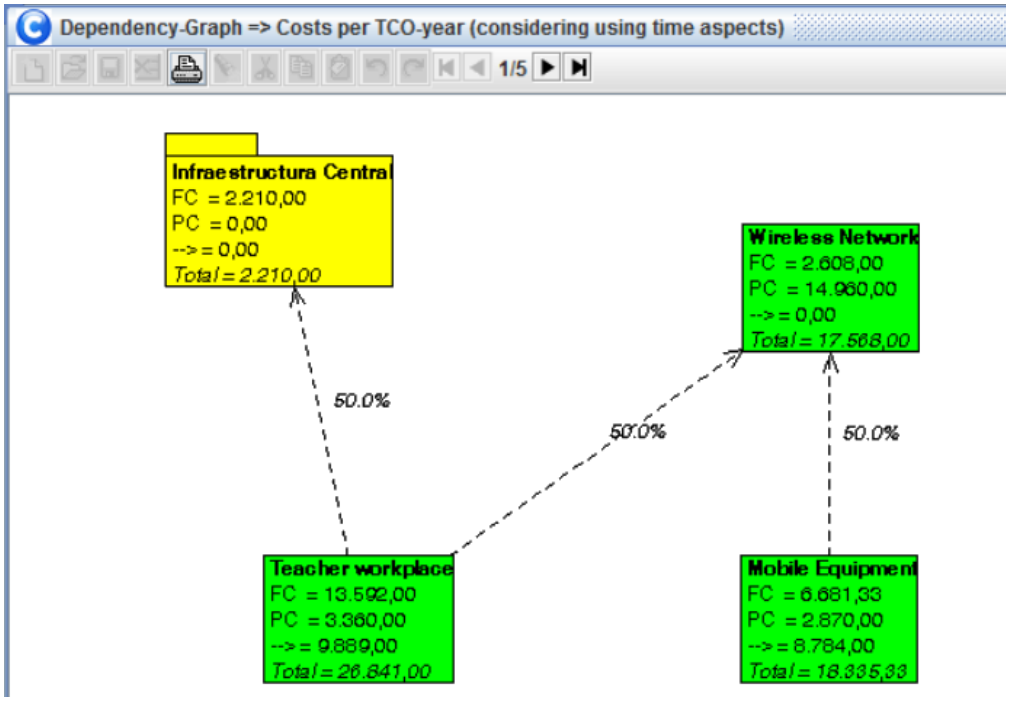
Cost-centre	TCO 1. Year	TCO 2. Year	TCO 3. Year	TCO 4. Year	TCO 5. Year	TCO costs over whole usage duration
Administration	0,00	0,00	0,00	0,00	0,00	**0,00**
Education	0,00	0,00	0,00	0,00	0,00	**0,00**
Infrastructure	0,00	0,00	0,00	0,00	0,00	**0,00**
Network-Installation	1.416,67	1.416,67	1.416,67	1.416,67	1.416,67	**7.083,33**
PC-Provider	28.713,33	22.483,33	22.483,33	22.483,33	22.483,33	**118.646,67**
Production	0,00	0,00	0,00	0,00	0,00	**0,00**
Swisscom Sponsoring	0,00	0,00	0,00	0,00	0,00	**0,00**
wistec	17.568,00	9.808,00	9.808,00	9.808,00	9.808,00	**56.800,00**
<Undefined>	0,00	0,00	0,00	0,00	0,00	**0,00**
Total	**47.698,00**	**33.708,00**	**33.708,00**	**33.708,00**	**33.708,00**	**182.550,00**

All costs in [USD]

En este reporte se puede observar el TCO por categorías a 5 años, las categorías son:

1. Por Tipo de Costo.
2. Por Responsabilidad.
3. Por Centro de Costo.
4. Por Proceso.
5. Por Sitio.

Gráfico de dependencia



En este grafico se puede observar la dependencia de los 4 grupos con los siguientes datos:

1. FC (Fact Cost). Costos de Hardware, Software, Comunicaciones, etc.
2. PC (Personal Cost). Costos de personal interno y/o externo.

En el manual de usuario de esta herramienta se tienen claramente explicados la forma de setear todas las variables y la explicación de todos los diferentes reportes.

Conclusiones

- Independientemente de si trata de reducir los costes corrientes o rentabilizar de forma más rápida la inversión, el cálculo del TCO nos facilitará la elección de la solución de TI que mejor se adapte a nuestras necesidades.
- Este tipo de herramienta nos facilita el calculo del TCO y nos permite tener los resultados rápidos cuando se realiza cualquier tipo de ajuste.
- Los reportes generados nos brindan información muy importante y detallada para la toma de decisiones en la gestión de TI.

Referencias

[1] http://www.tcotool.org

Renzo Martinez Pardo
Ingeniero de Sistemas Electrónicos
renzocmp@gmail.com
BOLIVIA

AtixLibre

Hacia un Futuro Innovador

Etico

Libre

Justo