



Universidad Zaragoza

Diseño e implementación de un sistema dinámico de gestión de trabajos distribuidos en un entorno de máquinas virtuales

David Ceresuela

Proyecto fin de carrera – Ingeniería en Informática
Curso 2011/2012

Director: Javier Celaya

Introducción

- Computación en la nube
 - Acceder a una aplicación cuya lógica y datos están situados en una localización remota
 - Infraestructura distribuida
 - Máquinas virtuales
 - Usos: ejecución de trabajos, web...
 - Problema: **administración** de la infraestructura

Introducción

- Herramientas de gestión de configuración
 - Describir y llevar a un sistema informático a un cierto estado
 - Iteración y convergencia
 - Incrementar la productividad
 - Capaces de administrar nodos en entornos heterogéneos y complejos
 - **Incapaces** de administrar infraestructuras distribuidas como una entidad propia

Objetivo

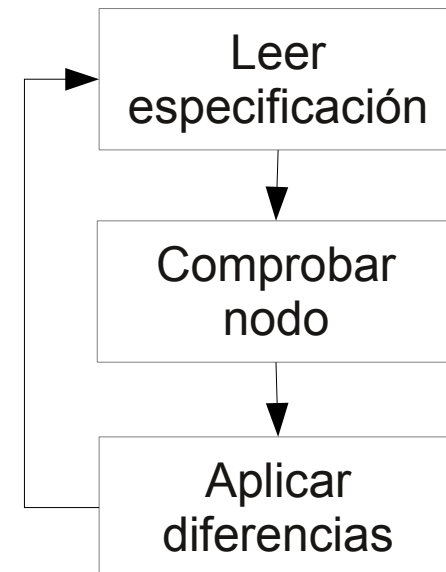
- Administración de infraestructuras distribuidas de ejecución de trabajos mediante herramientas de gestión de configuración

Ejemplos de infraestructuras distribuidas

- Ejecución de trabajos
 - AppScale
 - Implementación de código abierto del App Engine de Google
 - Alojamiento de aplicaciones web y ejecución de trabajos
 - Mayor **complejidad** por diversidad de nodos
 - TORQUE
 - Infraestructura clásica de ejecución de trabajos
 - Nodos de tipo maestro o computación
- Servicios web
 - Balanceador de carga, servidor web, base de datos

Gestión de la configuración en Puppet

- Especificación declarativa de los diferentes recursos a administrar (ficheros, usuarios, etc.)
- Ventajas
 - Programada en Ruby: abstracciones de mayor nivel que C y Java
 - Permite ser extendida
- Iteración y convergencia: comprobación sistemática del estado de los recursos



Modelado de recursos locales

- Locales a un nodo

```
class servidor_apache {  
  $apacheversion = "2.0.33"  
  package { "apache2":  
    ensure => $apacheversion,  
  }  
  service { "apache2":  
    ensure => running,  
    enable => true,  
  }  
}  
node /^www.+/ {  
  include servidor_apache  
}
```

- Son los únicos que existen en Puppet

Modelado de recursos distribuidos

- No existen en Puppet (ni en CFEngine, ni en Chef...)
- Características propias comunes:
 - Dependencia
 - Entre nodos, entre servicios de red...
 - Disponibilidad (fallos)
 - De máquina
 - De servicio
- Es necesaria una coordinación

Definición de un recurso en Puppet

- Tipo
 - Especificación declarativa en la que se definen los atributos del recurso
- Proveedor
 - Implementación necesaria para llevar dicho recurso al estado deseado

Definición de un recurso distribuido

- Tipo **recurso distribuido**
 - Atributos comunes: Nombre, Fichero de dominio, Conjunto de máquinas físicas
 - Añadir los atributos particulares de cada recurso
- Proveedor **recurso distribuido**
 - Puesta en marcha
 - Monitorización
 - Fallos de recursos: Puppet
 - Fallos del coordinador
 - Parada
 - Funciones específicas del recurso

Implementación

- Proveedor
 - Primera aproximación: proveedor común de recurso distribuido
 - Problema: Puppet no soporta herencia entre proveedores de distintos tipos
 - Segunda aproximación: proporcionar una clase Ruby con las funcionalidades comunes a todo tipo de proveedores

Implementación

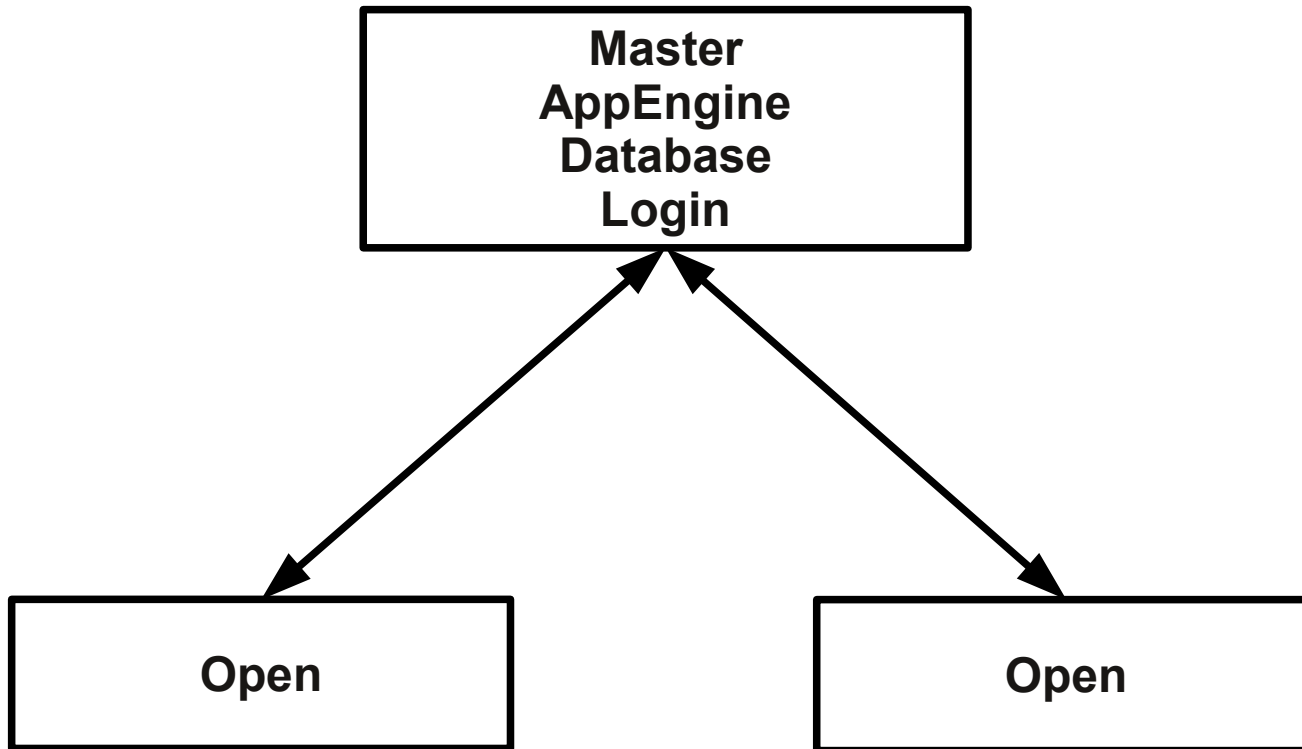
- **Clase Cloud**

- Clase Ruby
- Puesta en marcha
 - Inicio como líder
 - Inicio como no líder
 - Inicio como nodo no perteneciente a la infraestructura
- Monitorización
 - Monitorización como líder
- Parada
 - Apagado de máquinas virtuales
 - Borrado de ficheros de gestión interna

Implementación

- Recurso **appscale**
 - Tipo: Atributos comunes + atributos AppScale
 - Proveedor: Funciones AppScale
- Recurso **torque**
 - Tipo: Atributos comunes + atributos TORQUE
 - Proveedor: Funciones TORQUE
- Recurso **web**
 - Tipo: Atributos comunes + atributos web
 - Proveedor: Funciones web

Ejemplo: AppScale



Ejemplo: AppScale

- Manifiesto:

```
appscale {'myappscale':  
  master    => ["155.210.155.73", "/var/tmp/lucid-app1.img"],  
  appengine => ["155.210.155.73", "/var/tmp/lucid-app1.img"],  
  database  => ["155.210.155.73", "/var/tmp/lucid-app1.img"],  
  login     => ["155.210.155.73", "/var/tmp/lucid-app1.img"],  
  open      => ["/etc/.../modules/appscale/files/open-ips.txt",  
               "/etc/.../modules/appscale/files/open-imgs.txt"],  
  vm_domain => "/etc/puppet/modules/appscale/files/template.xml",  
  pool      => ["155.210.155.70"],  
  ensure    => running,  
}
```

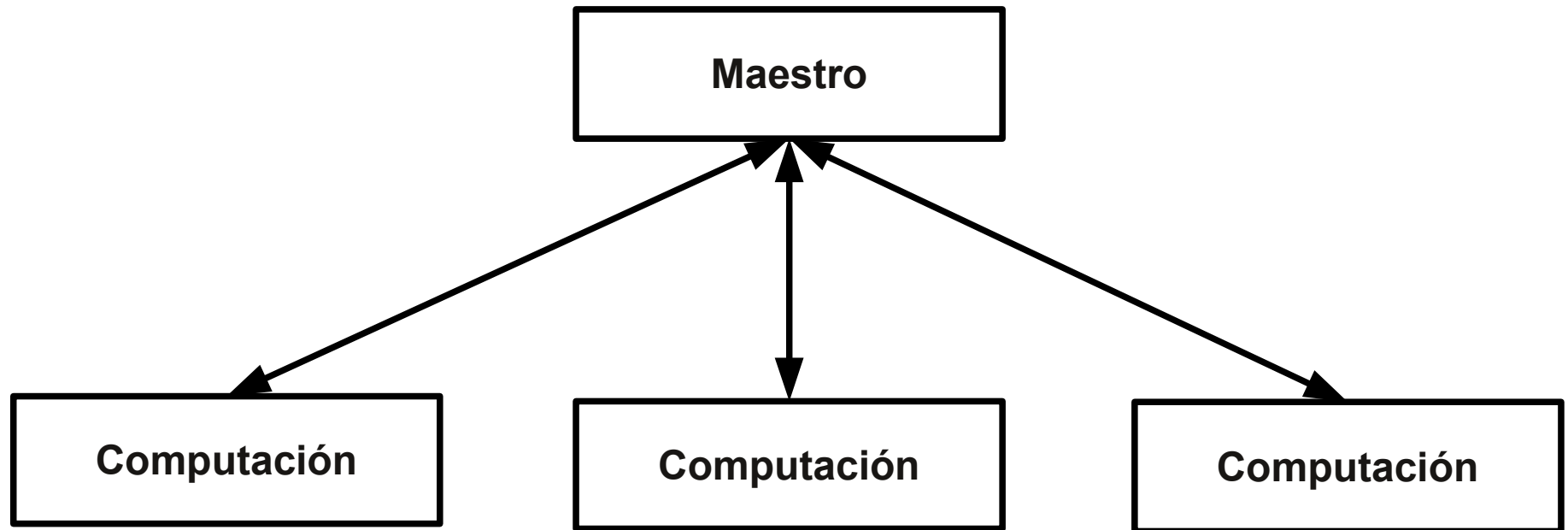
Ejemplo: AppScale

- Proveedor:

```
...
if part_of_cloud
  puts "#{MY_IP} is part of the cloud"

  # Check if you are the leader
  if cloud.leader?()
    cloud.leader_start("appscale", vm_ips, vm_ip_roles,
                      vm_img_roles, pm_up, method(:appscale_monitor))
  else
    cloud.common_start()
  end
else
  puts "#{MY_IP} is not part of the cloud"
  cloud.not_cloud_start("appscale", vm_ips, vm_ip_roles,
                       vm_img_roles, pm_up)
end
...
```


Ejemplo: TORQUE



Ejemplo: TORQUE

- Manifiesto:

```
torque {'mytorque':  
  head    => ["155.210.155.73", "/var/tmp/lucid-tor1.img"],  
  compute => ["/etc/.../modules/torque/files/compute-ips.txt",  
              "/etc/.../modules/torque/files/compute-imgs.txt"],  
  vm_domain => "/etc/puppet/modules/torque/files/template.xml",  
  pool      => ["155.210.155.70"],  
  ensure    => running,  
}
```

Ejemplo: TORQUE

- Proveedor:

```
def stop

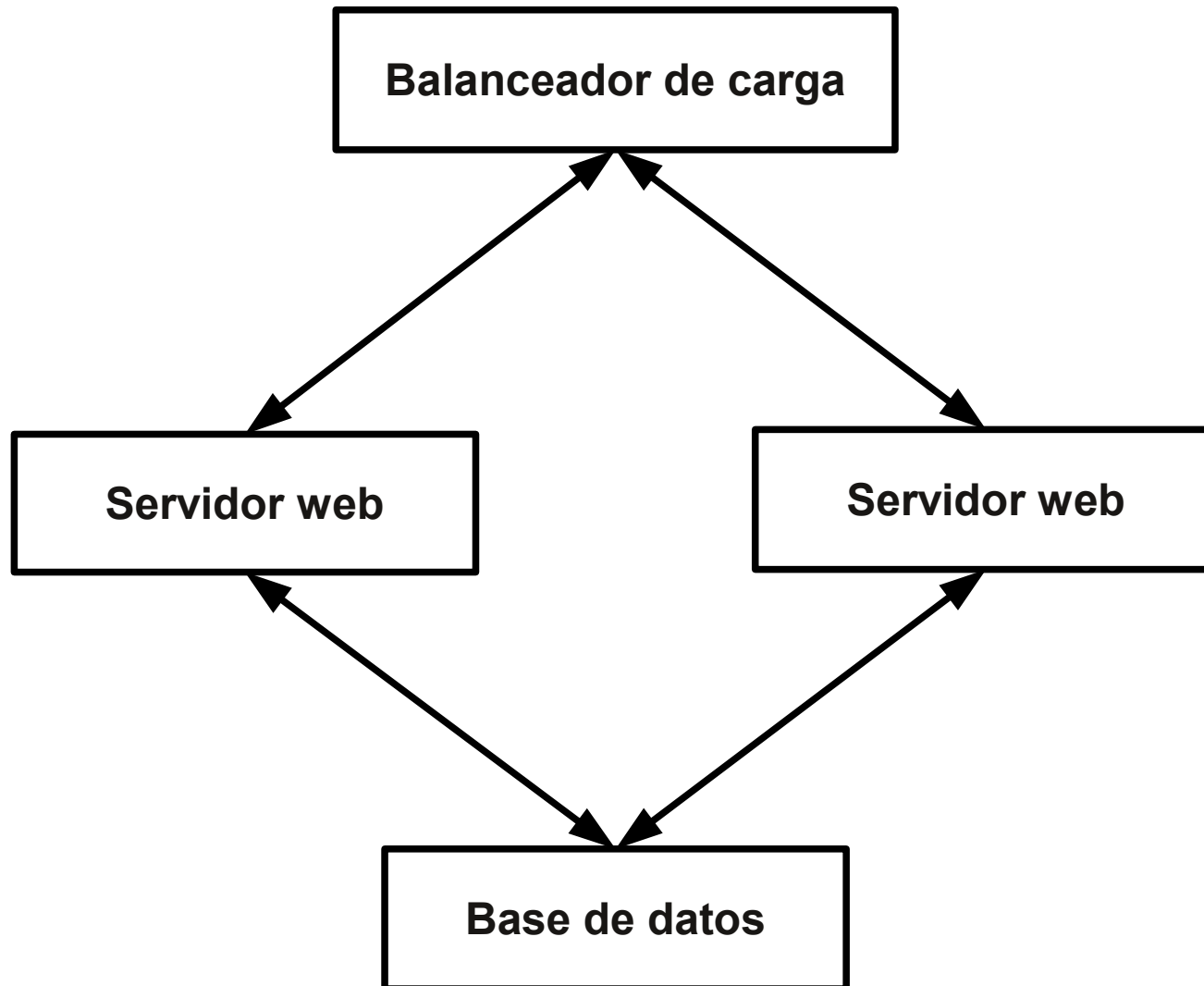
  cloud = Cloud.new(CloudInfrastructure.new(), CloudLeader.new(),
                    resource, method(:err))
  puts "Stopping cloud %s" % [resource[:name]]

  if cloud.leader?()
    ...

    if exists? && status == :running
      puts "It is a torque cloud"

      # Stop cloud infrastructure
      cloud.leader_stop("torque", method(:torque_cloud_stop))
    end
  else
    puts "#{MY_IP} is not the leader"      # Nothing to do
  end
end
```

Ejemplo: Infraestructura de servicios web de tres niveles



Ejemplo: Infraestructura de servicios web de tres niveles

- Manifiesto:

```
web {'myweb':  
  balancer => ["155.210.155.175", "/var/tmp/lucid-lb.img"],  
  server   => ["/etc/puppet/modules/web/files/server-ips.txt",  
               "/etc/puppet/modules/web/files/server-imgs.txt"],  
  database => ["155.210.155.177", "/var/tmp/lucid-db.img"],  
  vm_domain => "/etc/puppet/modules/web/files/template.xml",  
  pool      => ["155.210.155.70"],  
  ensure    => running,  
}
```

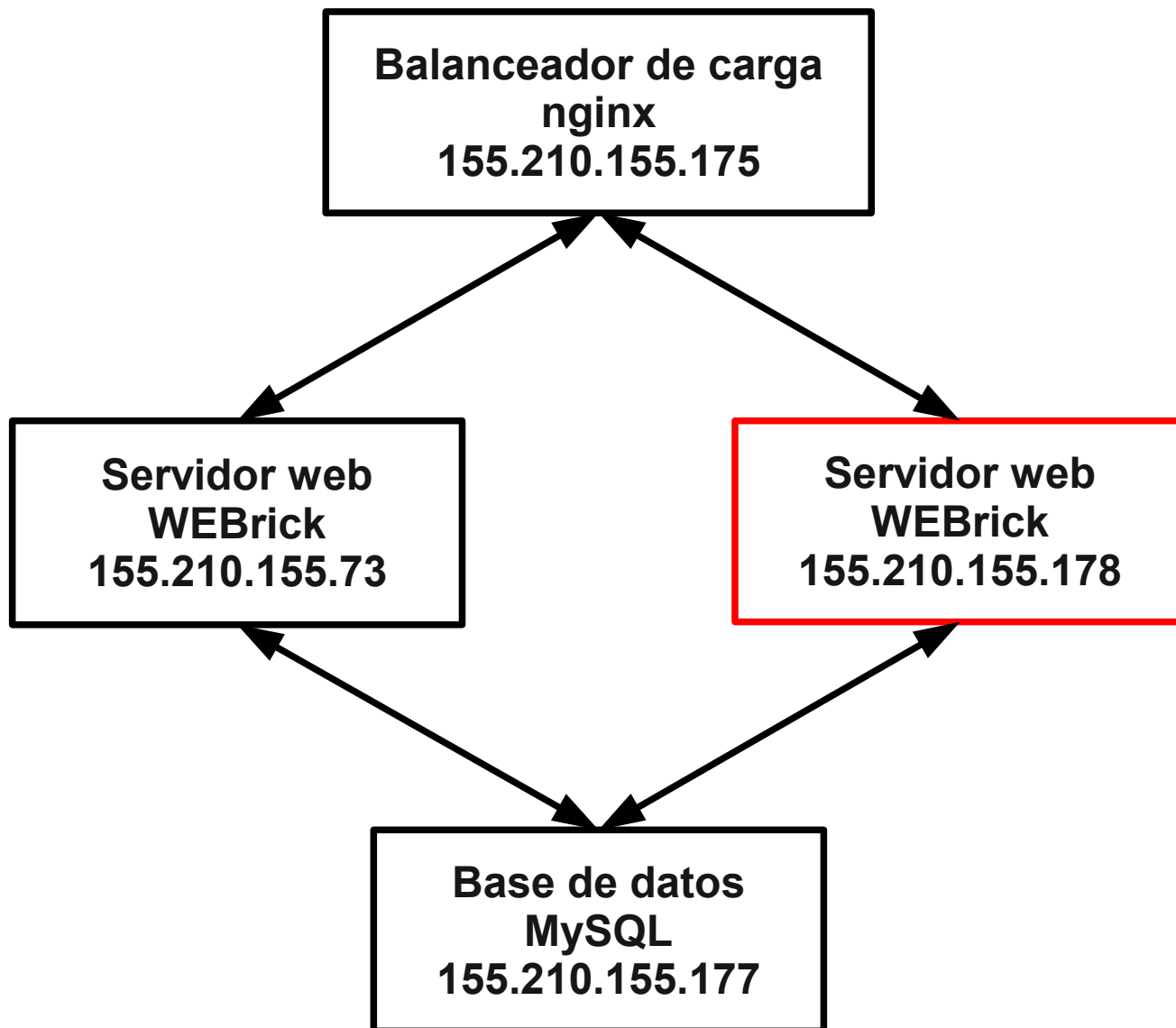
Ejemplo: Infraestructura de servicios web de tres niveles

- Tipo:

```
newproperty(:balancer,  
  :array_matching => :all) do  
  
  desc "The balancer node's information"  
end  
  
newproperty(:server,  
  :array_matching => :all) do  
  
  desc "The server nodes' information"  
end  
  
newproperty(:database,  
  :array_matching => :all) do  
  
  desc "The database node's information"  
end
```

```
newparam(:name) do  
  desc "The cloud name"  
  isnamevar  
end  
  
newparam(:vm_domain) do  
  desc "The XML file with ..."  
end  
  
newproperty(:pool,  
  :array_matching => :all) do  
  desc "The pool of ..."  
end
```

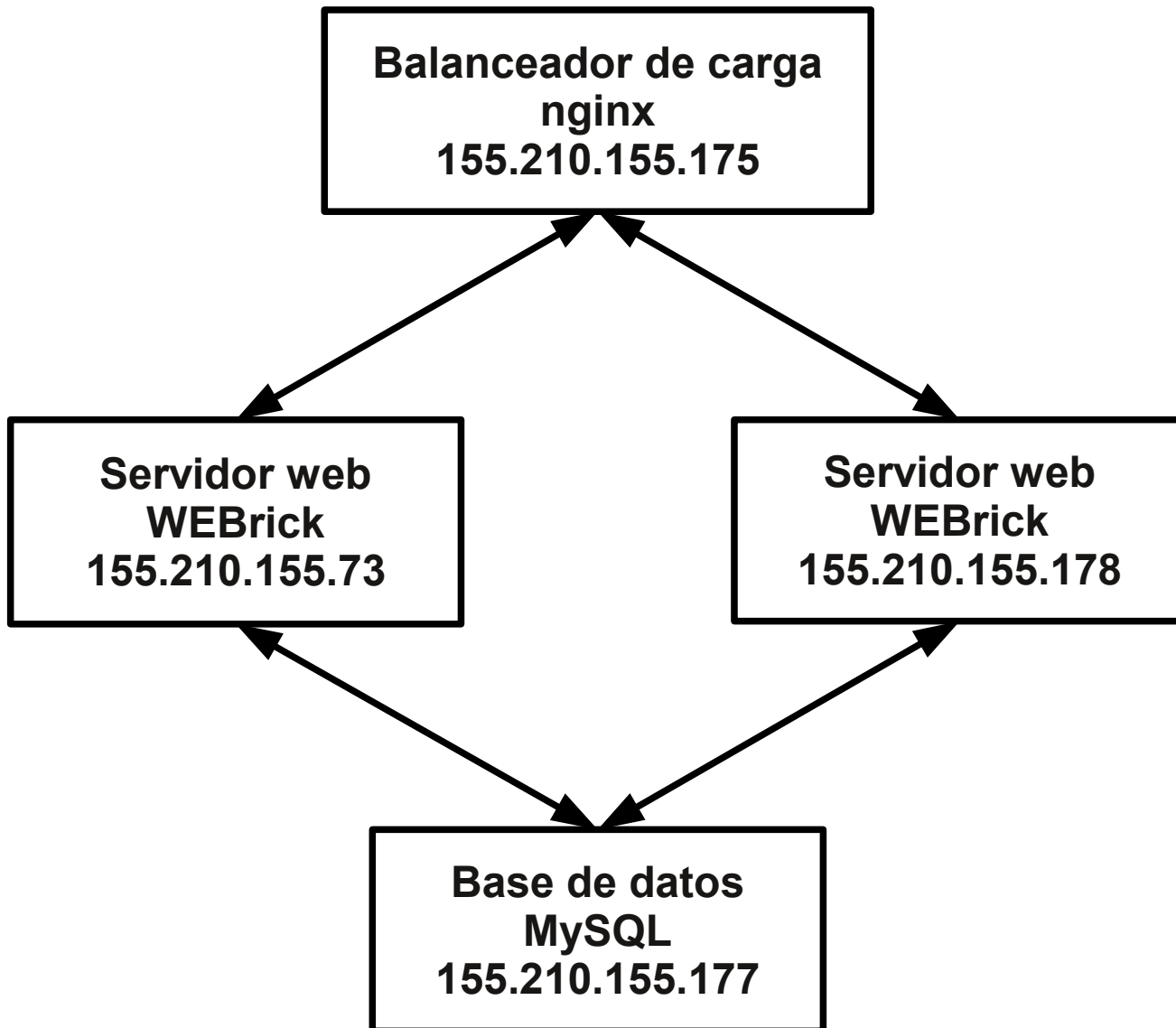
Demostración



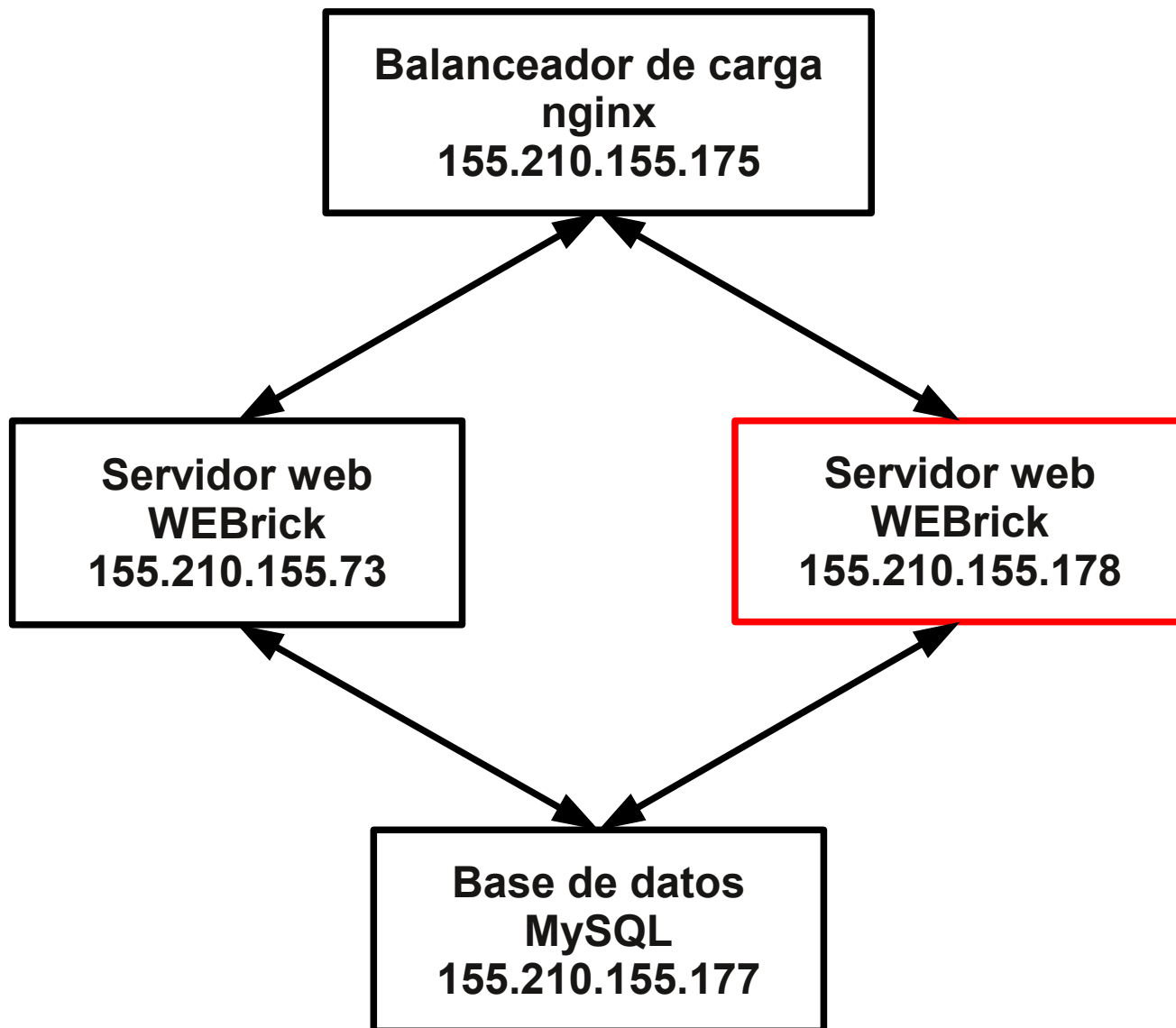
Demostración



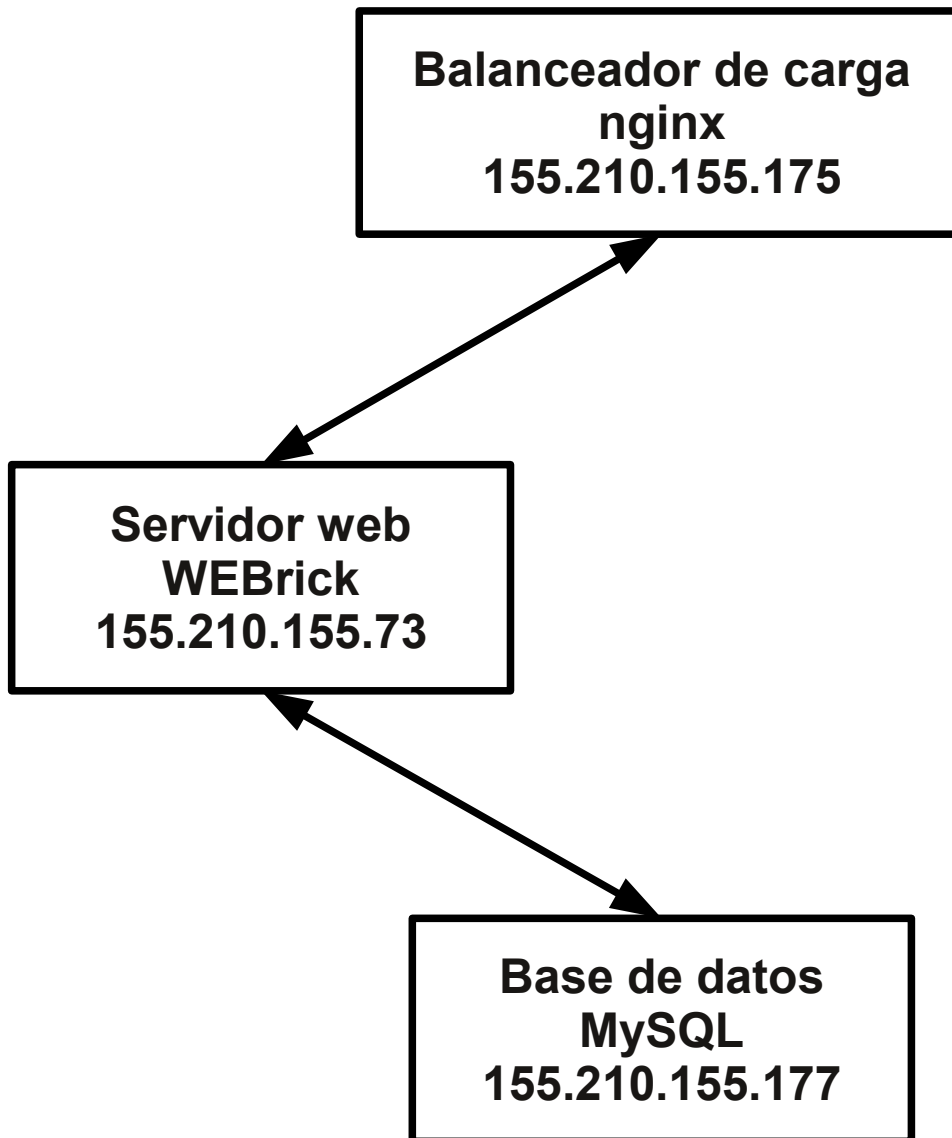
Demostración



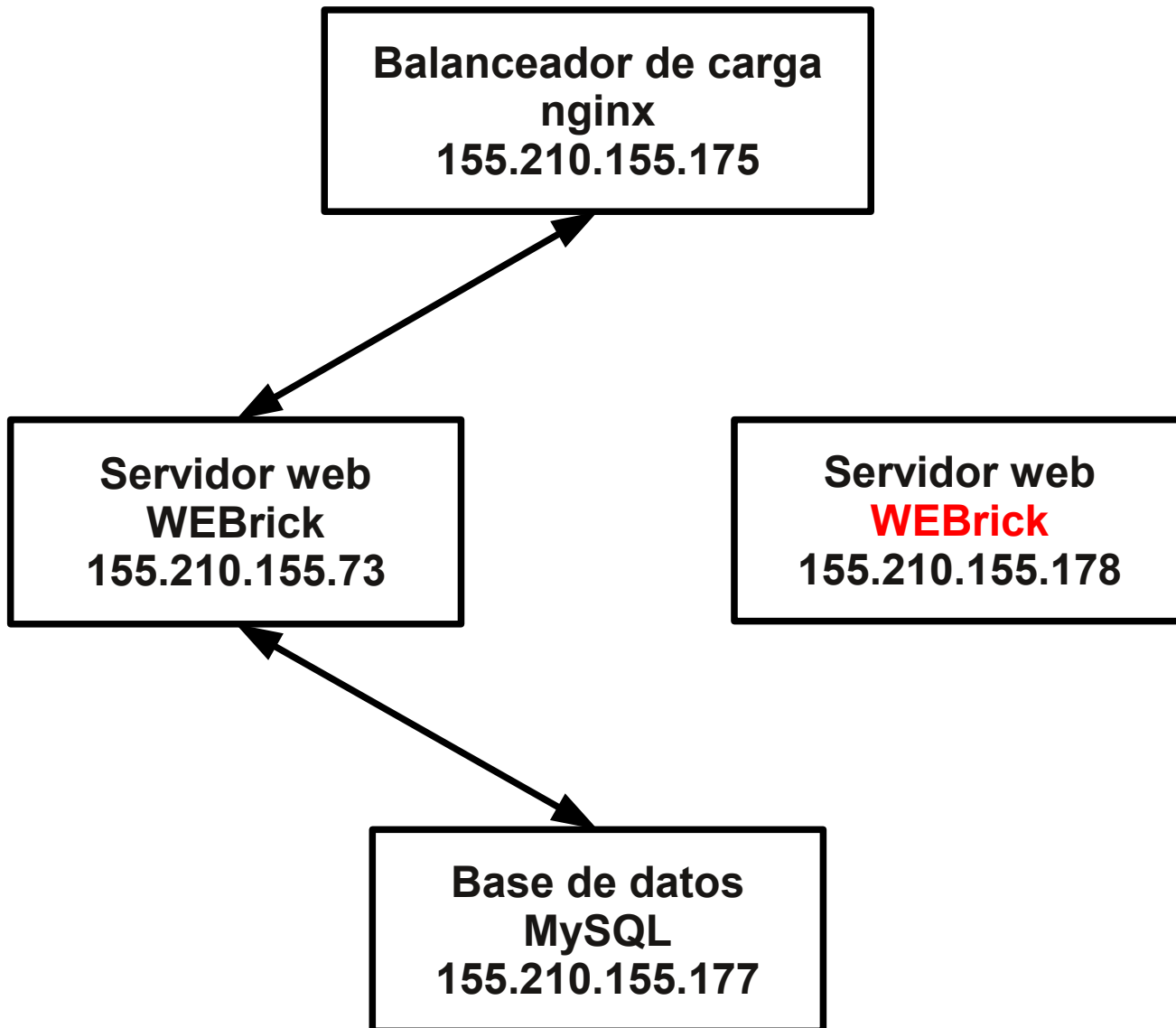
Demostración



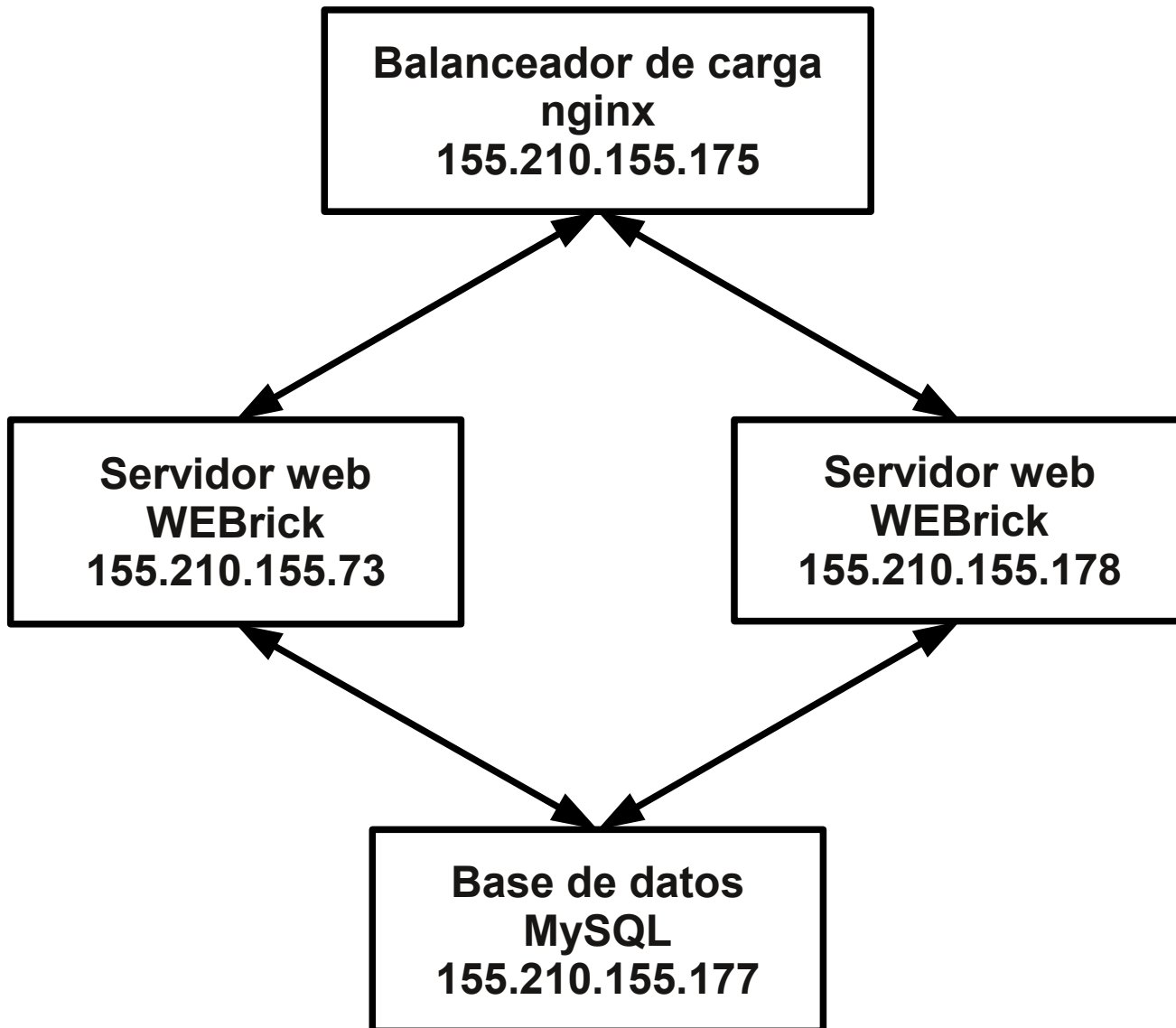
Demostración



Demostración



Demostración



Tecnologías

- MCollective: Envío de órdenes a varios nodos al mismo tiempo
- RabbitMQ: Patrón de mensajes editor / suscriptor
- god: Monitorización
- Web
 - nginx: Balanceador de carga
 - WEBrick: Servidor web
 - MySQL: Base de datos

Conclusiones

- Objetivo: Administración de infraestructuras distribuidas de ejecución de trabajos mediante herramientas de gestión de configuración
- Se ha creado la abstracción recurso distribuido
- Facilidad de puesta en marcha y administración de infraestructuras complejas
 - De ejecución de trabajos
 - De servicios web
- Se ha ido más allá de los objetivos iniciales



Universidad Zaragoza

Diseño e implementación de un sistema dinámico de gestión de trabajos distribuidos en un entorno de máquinas virtuales

David Ceresuela

Proyecto fin de carrera – Ingeniería en Informática
Curso 2011/2012

Director: Javier Celaya

Extras

- Manifiestos
- Clase Cloud
- Elección de líder
- Gestión del proyecto
- Servicios web
- AppScale
- IaaS, PaaS, SaaS

Manifestos

```
class fich1 {  
  file { 'testfile':  
    path      => '/tmp/testfile',  
    ensure    => present,  
    mode      => 0640,  
    content   => "I'm a test file.",  
  }  
}  
  
class fich2 {  
  ...  
}  
  
...  
  
node 'web.domain.com' {  
  include fich1  
}  
  
node 'db.domain.com' {  
  include fich2  
}
```

Manifestos

```
web {'myweb':  
  balancer => ["155.210.155.175", "/var/tmp/lucid-lb.img"],  
  server   => ["/etc/puppet/modules/web/files/server-ips.txt",  
               "/etc/puppet/modules/web/files/server-imgs.txt"],  
  database => ["155.210.155.177", "/var/tmp/lucid-db.img"],  
  vm_domain => "/etc/puppet/modules/web/files/template.xml",  
  pool      => ["155.210.155.70"],  
  ensure    => running,  
}
```

Clase Cloud

- Puesta en marcha
 - Inicio como líder
 - Inicio como no líder
 - Inicio como nodo no perteneciente a la infraestructura
- Monitorización
 - Monitorización como líder
- Parada
 - Apagado de máquinas virtuales
 - Borrado de ficheros de gestión interna

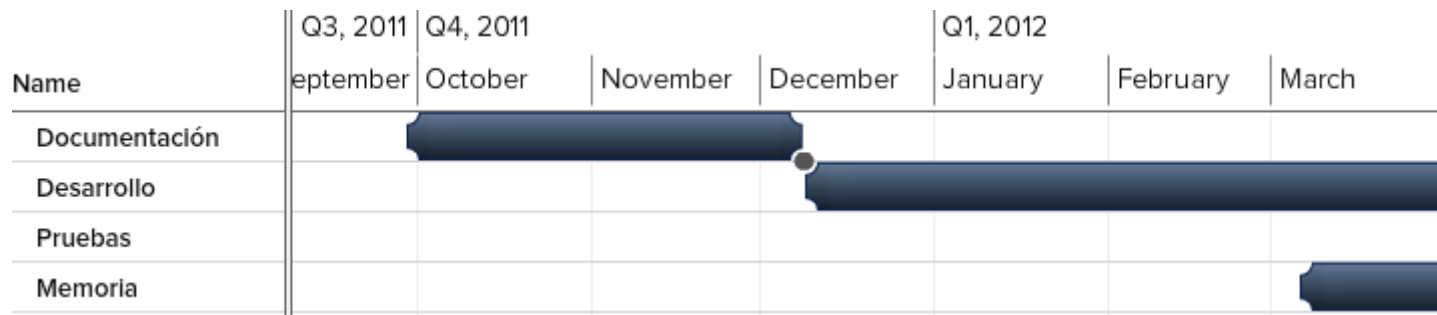
Clase Cloud

- Clase Cloud
- Módulo SSH
- Módulo Monitorización
- Clase Infraestructura (gestión infraestructura)
- Clase Elección de líder

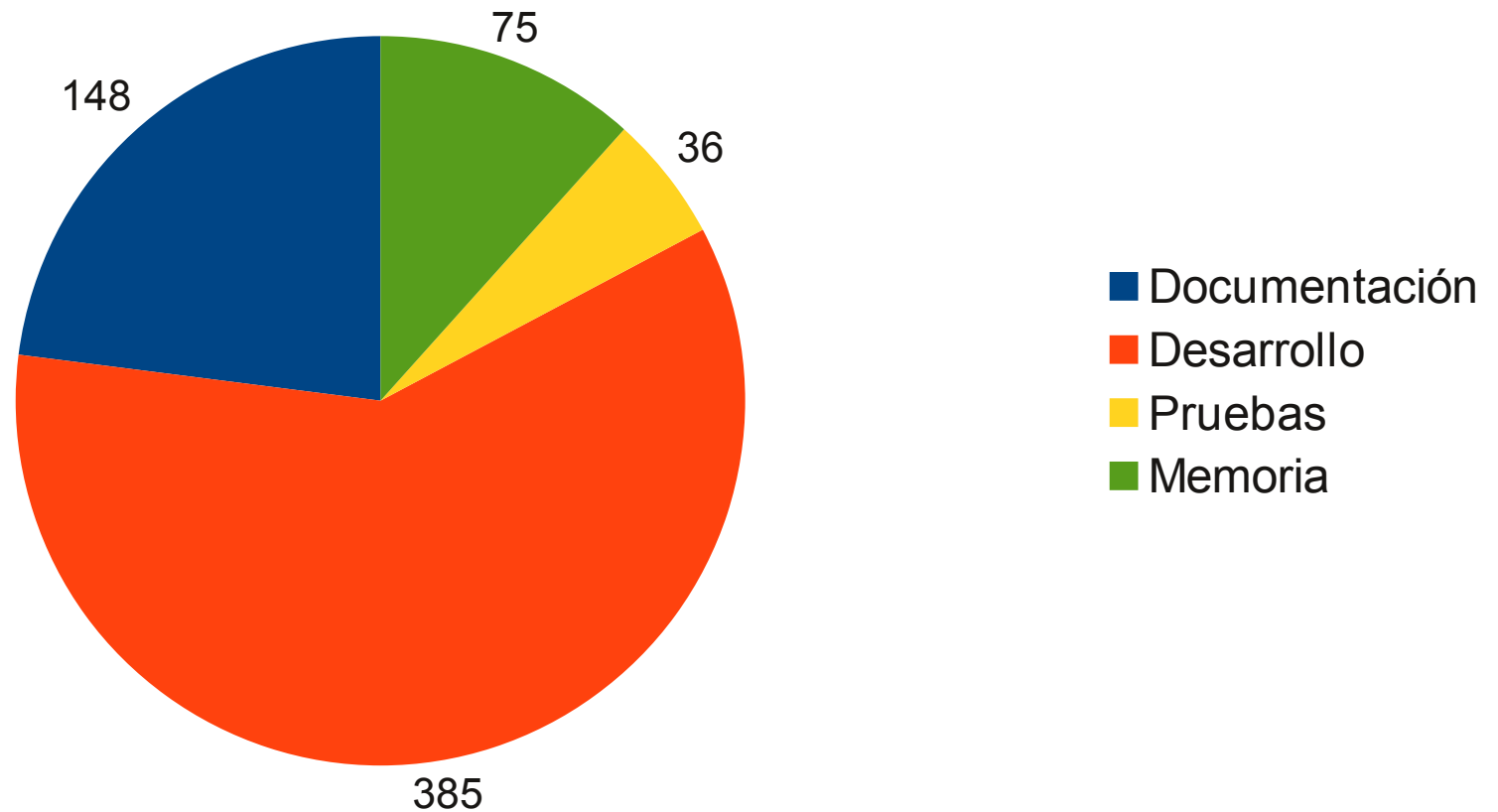
Algoritmo de elección de líder

- Algoritmo peleón (*Bully algorithm*)
 - Todos los nodos tratan periódicamente de convertirse en el líder
 - Si hay un líder, responderá negativamente
 - $ID_Lider < ID_Nodo_Cualquiera$
 - Si no hay líder, el nodo con menor ID se convierte en el líder
 - $ID_Nodo_23 < ID_Nodo_56$
 - $ID_Nodo_23 < ID_Nodo_47$

Gestión del proyecto



Gestión del proyecto



Servicios web

- 3 niveles (clásico)
 - Capa de presentación: Apache
 - Capa lógica: Rails
 - Capa de persistencia: MySQL
- 3 niveles (proyecto)
 - Balanceador de carga
 - Servidor web (Presentación + Lógica)
 - Base de datos (Persistencia)

AppScale

- Roles simples

- Shadow: Comprueba el estado en el que se encuentran el resto de nodos y se asegura de que están ejecutando los servicios que deben.
- Load balancer: Servicio web que lleva a los usuarios a las aplicaciones. Posee también una página en la que informa del estado de todas las máquinas desplegadas.
- AppEngine: Versión modificada de los SDKs de Google App Engine. Además de alojar las aplicaciones web añaden la capacidad de almacenar y recuperar datos de bases de datos que soporten el API de Google Datastore.
- Database: Ejecuta los servicios necesarios para alojar a la base de datos elegida.
- Login: La máquina principal que lleva a los usuarios a las aplicaciones App Engine. Difiere del Load Balancer en que esta es la única máquina desde la que se pueden hacer funciones administrativas. Puede haber muchas máquinas que hagan la función de Load Balancer pero sólo habrá una que haga función de Login.
- Memcache: Proporciona soporte para almacenamiento en caché para las aplicaciones App Engine.
- Zookeeper: Aloja los metadatos necesarios para hacer transacciones en las bases de datos.
- Open: No ejecuta nada por defecto, pero está disponible en caso de que sea necesario. Estas máquinas son las utilizadas para ejecutar trabajos MPI.

AppScale

- Roles compuestos
 - Controller: Shadow, Load Balancer, Database, Login y Zookeeper.
 - Servers: App Engine, Database y Load Balancer.
 - Master: Shadow, Load Balancer y Zookeeper.

IaaS, PaaS, SaaS

- IaaS
 - Ofrecen recursos: CPU, red, disco duro
 - Ejemplos: Amazon EC2, Rackspace cloud
- PaaS
 - Proporcionan una infraestructura gestionada (escalabilidad, disponibilidad)
 - Ofrecen plataforma de desarrollo: lenguajes, tecnologías
 - Ejemplos: Google AppEngine, Heroku
- SaaS
 - Orientado a negocios
 - Consumo a través de la web
 - Ejemplos: Google Apps, Salesforce

