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

- Aproximación clásica a la ejecución de trabajos:
  - *Cluster* de ordenadores
  - Computación en malla o *grid*
- Nueva aproximación a la ejecución de trabajos:
  - Computación en la nube
    - Acceder a una aplicación cuya lógica y datos están situados en una localización remota
    - Problema: administración de la infraestructura

Introducción

- Herramientas de gestión de configuración
  - Considerable avance en los últimos años
  - Describir y llevar a un sistema informático a un cierto estado
  - Iteración y convergencia
  - Capaces de administrar nodos en entornos heterogéneos y complejos
  - Incapaces de administrar infraestructuras distribuidas como una entidad propia

Puppet

- Lenguaje declarativo: especificación de los distintos elementos de conguración (recursos)
- Programada en Ruby
- Permite ser extendida
- Configuración mediante manifiestos:

```
file { 'testfile':
  path    => '/tmp/testfile',
  ensure  => present,
  mode    => 0640,
  content => "I'm a test file.",
}
```

Infraestructuras de ejecución de trabajos distribuidos

- AppScale
  - Implementación de código abierto del App Engine de Google
  - Alojamiento de aplicaciones web y ejecución de trabajos
  - Nodos de muy diversos tipos
- TORQUE
  - Infraestructura clásica de ejecución de trabajos
  - Nodos de tipo maestro o computación

Infraestructura de servicios web en tres niveles

- Balanceador de carga: distribuir las peticiones web a los servidores web
- Servidores web: procesar peticiones web; lectura o escritura en una base de datos
- Base de datos: almacenamiento de la información

Modelado de recursos

- Recursos locales
  - Clásicos en Puppet
  - Locales a un nodo
- Recursos distribuidos
  - No existen en Puppet
  - Características propias comunes:
    - Dependencia
    - Disponibilidad

Modelado de recursos

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

Modelado de recursos

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

```
file { 'testfile':
  path    => '/tmp/testfile',
  ensure  => present,
  mode    => 0640,
  content => "I'm a test file.",
}
```

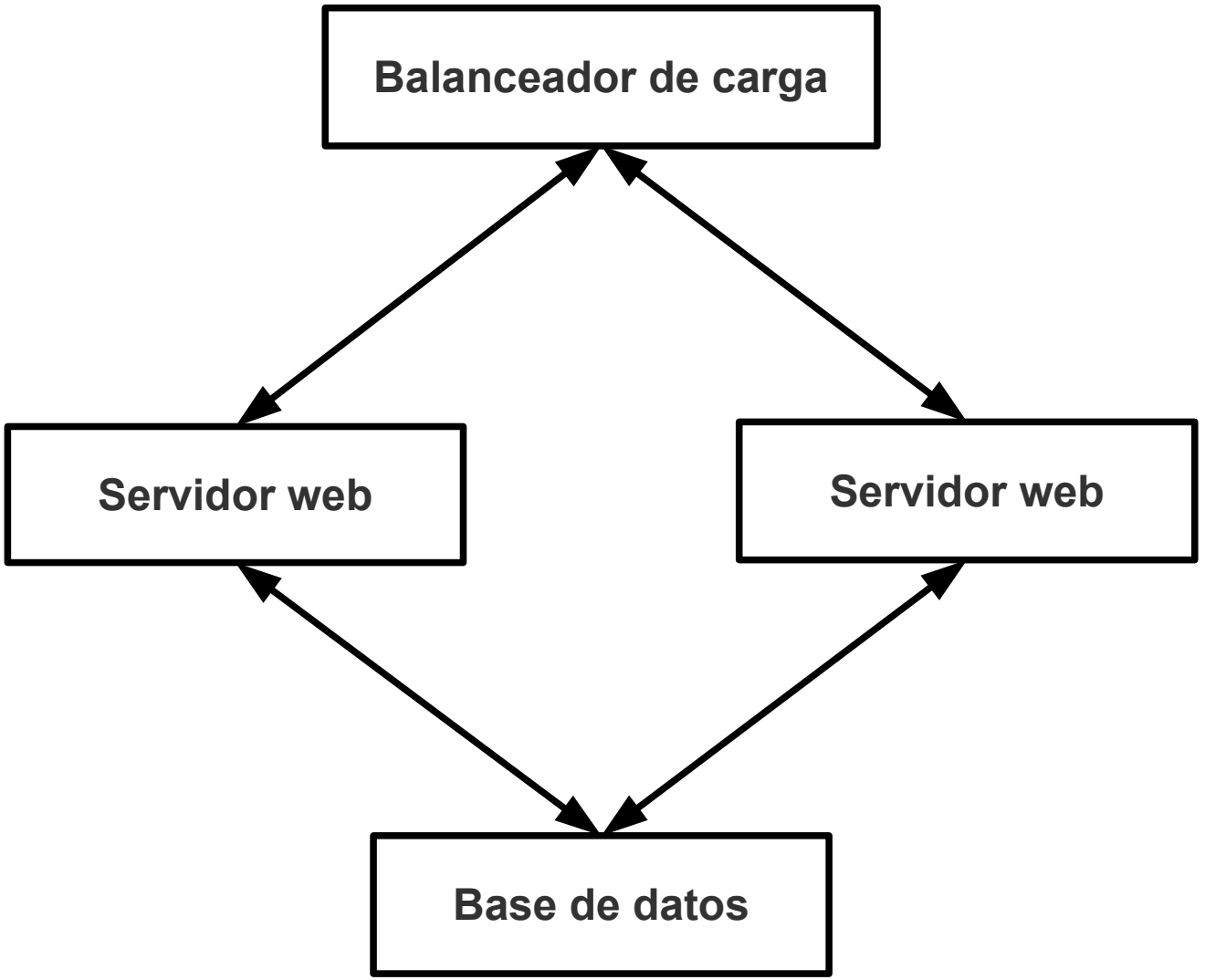
Modelado de recursos distribuidos

- Tipo **Recurso distribuido**
  - Nombre
  - Fichero de dominio
  - Conjunto de máquinas físicas
- Proveedor **Recurso distribuido**
  - Puesta en marcha
  - Monitorización
  - Parada

Modelado de recursos distribuidos

- Tipo
  - Añadir los atributos particulares de cada recurso
- Proveedor
  - Primera aproximación: proveedor genérico 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: puesta en marcha, monitorización y parada

Ejemplo: Infraestructura de servicios web de tres niveles

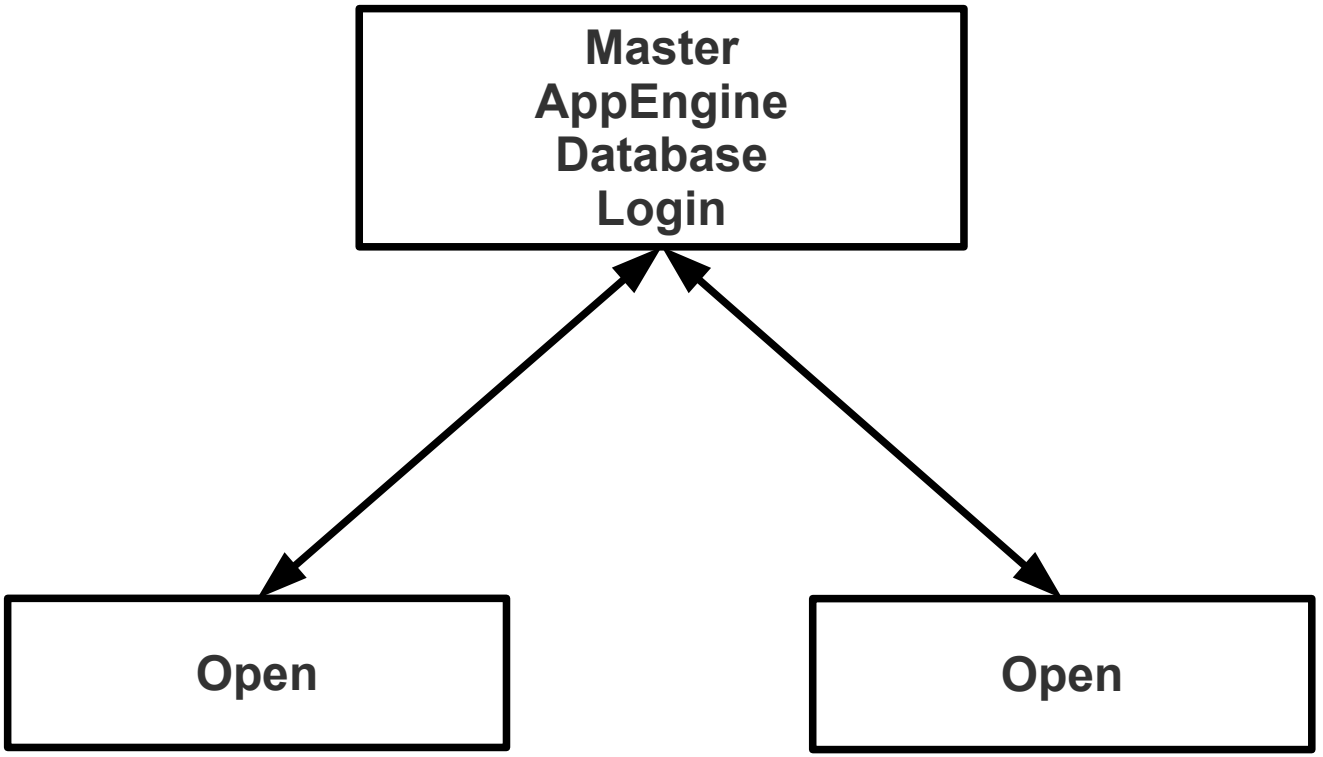


Ejemplo: Infraestructura de servicios web de tres niveles

- 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: AppScale

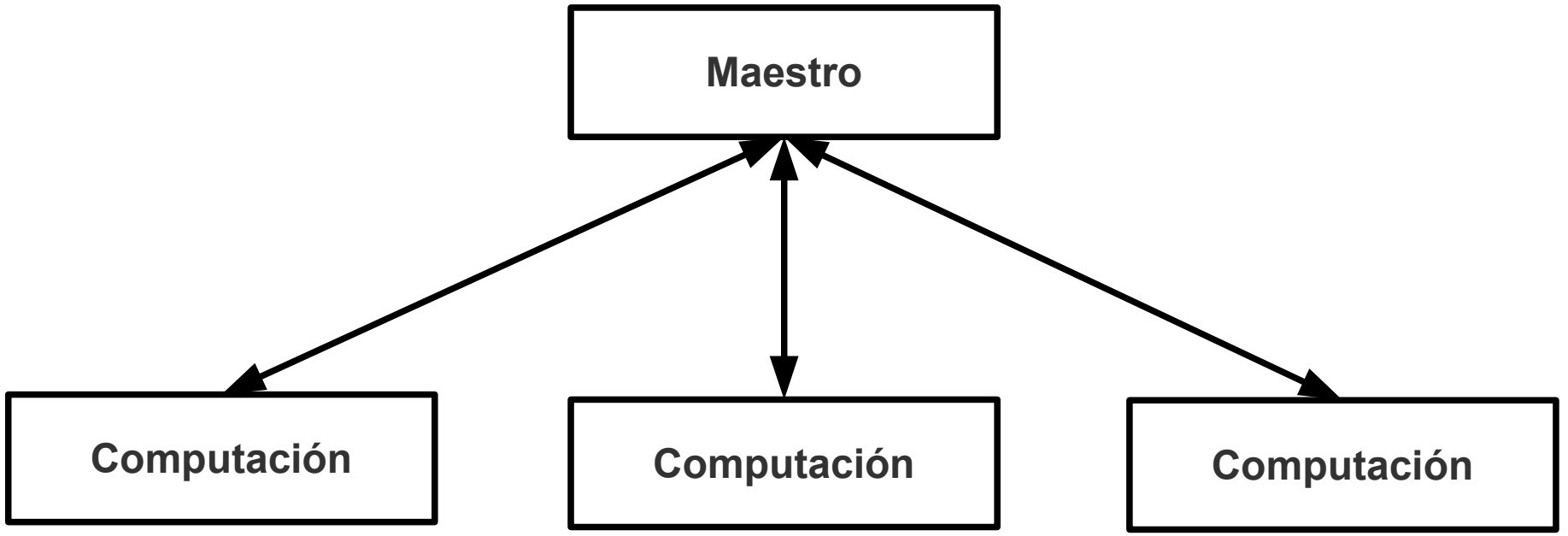


Ejemplo: AppScale

- 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: TORQUE

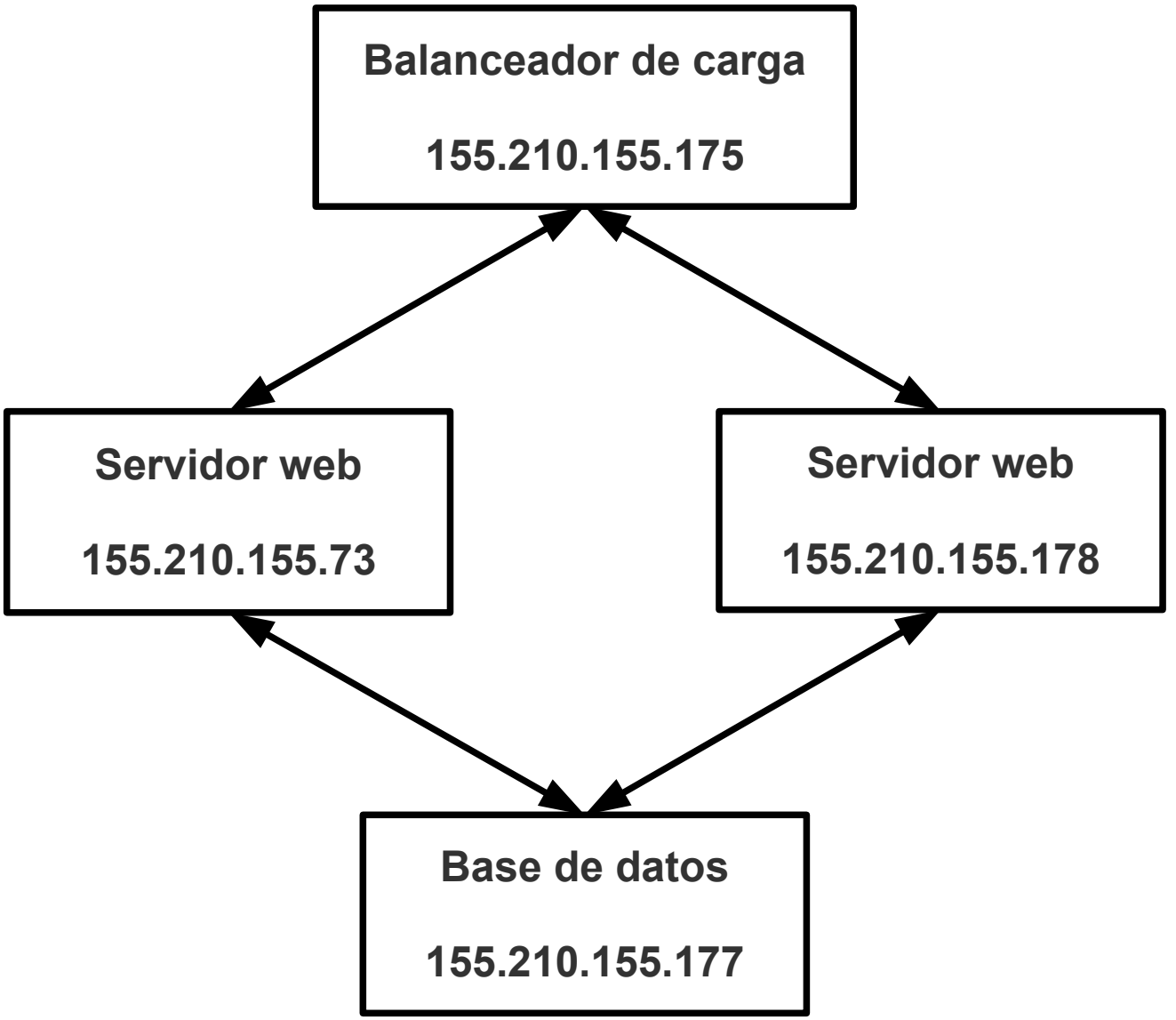


Ejemplo: TORQUE

- 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,
}
```

Demostración



Demostración

Conclusiones

- Creación de la abstracción recurso distribuido
- Facilidad de puesta en marcha y administración de infraestructuras complejas
  - De ejecución de trabajos
  - De servicios web

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

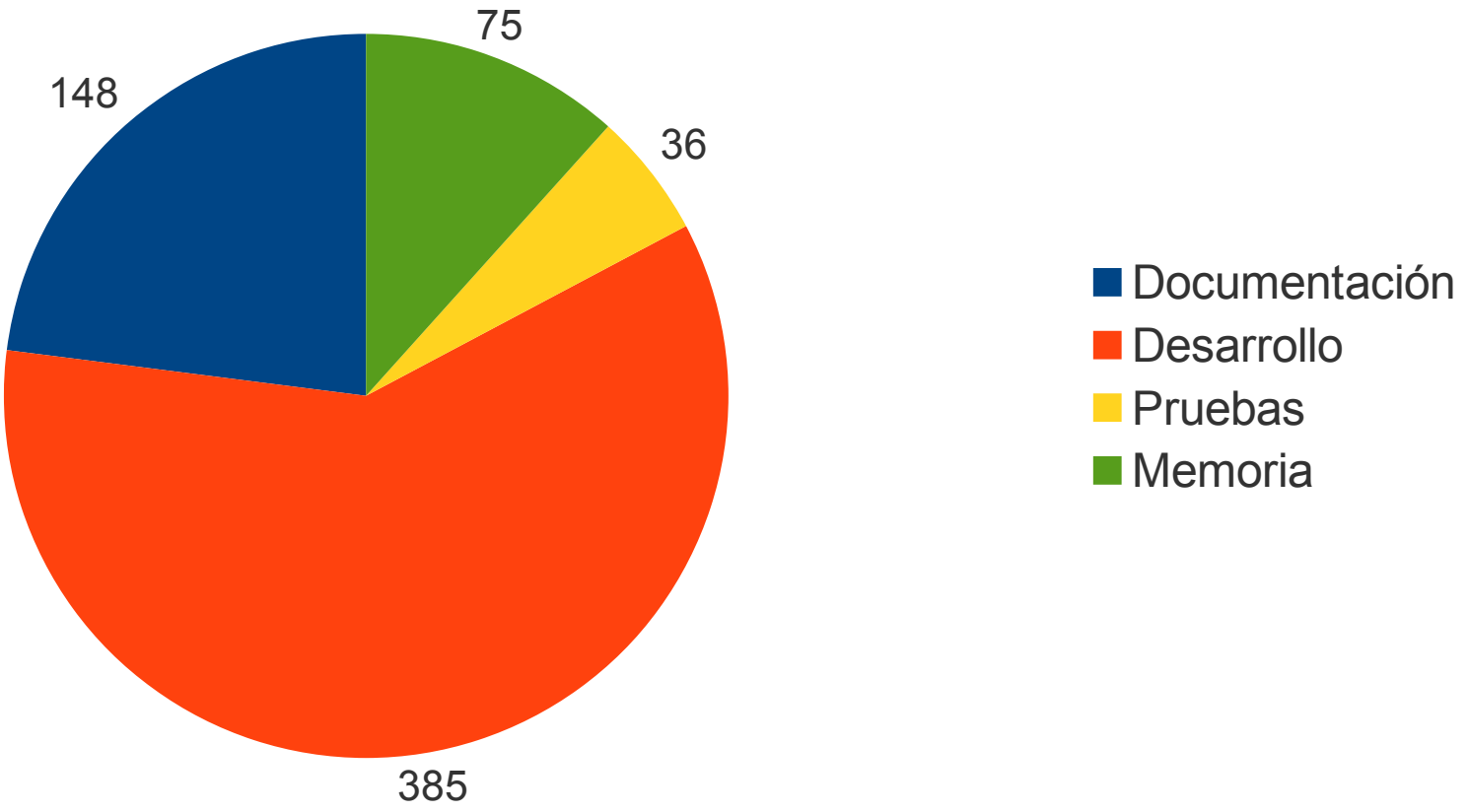
Extras

- Elección de líder
- Gestión del proyecto
- Servicios web
- AppScale
- Iaas, Paas, Saas

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



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
  - Consumición a través de la web
  - Ejemplos: Google Apps, SalesForce