redhat.

ARQUITECTURA DE LOS COMPONENTES OPENSTACK

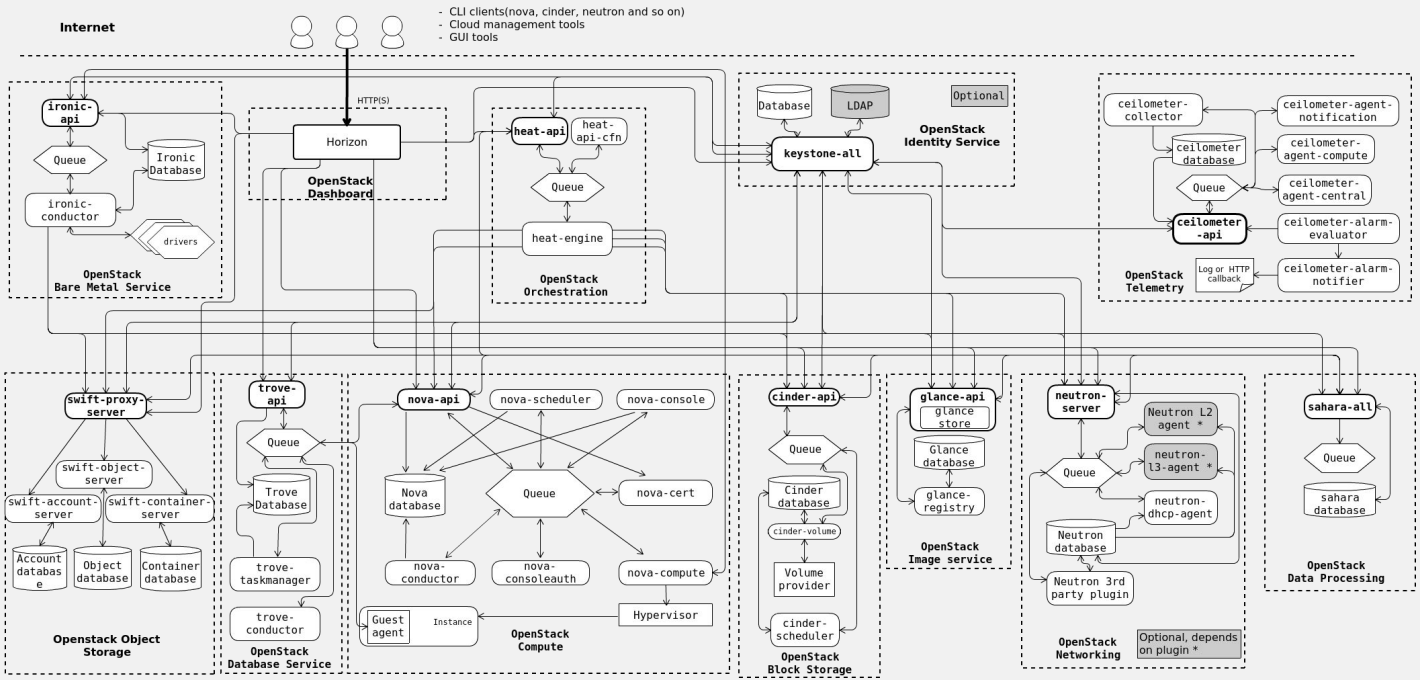
OPENSTACK MINI-CONF

Gorka Eguileor <geguileo@redhat.com>
Madrid Feb-2016

MAD FOR openstack
CLOUD SOFTWARE

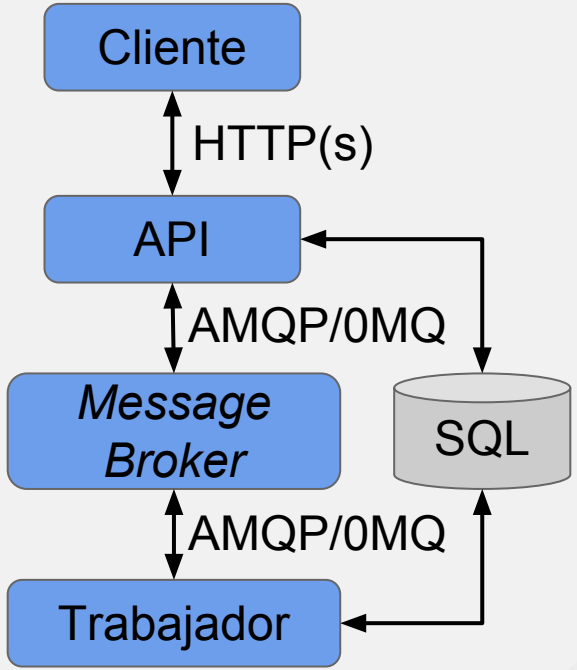
Ecosistema Openstack

- Gran número de componentes
- Más componentes en cada release



Arquitectura

- Comunicación interna
  - REST: Glance, Swift
  - Mensajería: Cinder, Nova, Neutron, Magnum
- Librerías Oslo para las funciones comunes
  - Servicios ⇒ oslo.service
  - Mensajería ⇒ oslo.messaging
  - Acceso a DB ⇒ oslo.db
  - Logs ⇒ oslo.log
  - Configuraciones ⇒ oslo.config



Arquitectura - Clientes

- Clientes REST clients ⇒ curl, scripts, programas, tempest
- python-*component*client
  - Deprecados en favor de openstackclient https://review.openstack.org/243348
  - Funcionalidad atrasada con respecto al servidor
  - CLI + librería Python
    - Usada en Nova, Horizon, scripts de automatización
    - Un fallo en Nova o Horizon puede ser causado por un bug en un cliente
- python-openstackclient
  - Cliente oficial unificado: *Compute*, *Identity*, *Image*, *Swift*, *Volumes*…
  - Usa los clientes de *cinder*, *nova*, *keystone*, y *glance*

Arquitectura - API

- *Frontend*: Enruta **todas** las peticiones del “exterior”
  - Incluidas las de otros componentes
- Aplicación WSGI: REST API + JSON
  - Algunos también XML ⇒ Deprecado, desaparecerá
  - Docs: http://developer.openstack.org/api-ref.html
  - Cualquiera puede acceder directamente
  - Sin funcionalidades ocultas o sólo interna a OpenStack
- Multiples versiones:
  - Diferentes acciones disponibles
  - En una versión se añaden argumentos opcionales en las releases
  - En algunos componentes no habrá más versiones → microversionado
- Soportan configuraciones Activo-Activo ⇒ Balanceador de carga

Arquitectura - API - peticiones

- Peticiones de metadatos:
  - Procesados directamente en el nodo (ej: *list*)
- Otras peticiones:
  - Reflejar operación en la BBDD
  - Llamada RPC - mensaje AMQP-0MQ
    - Normalmente a través del *Scheduler*
    - Algunos componentes directamente al trabajador
  - Devuelven la respuesta
    - De la llamada RPC
    - Del contenido escrito a la BBDD

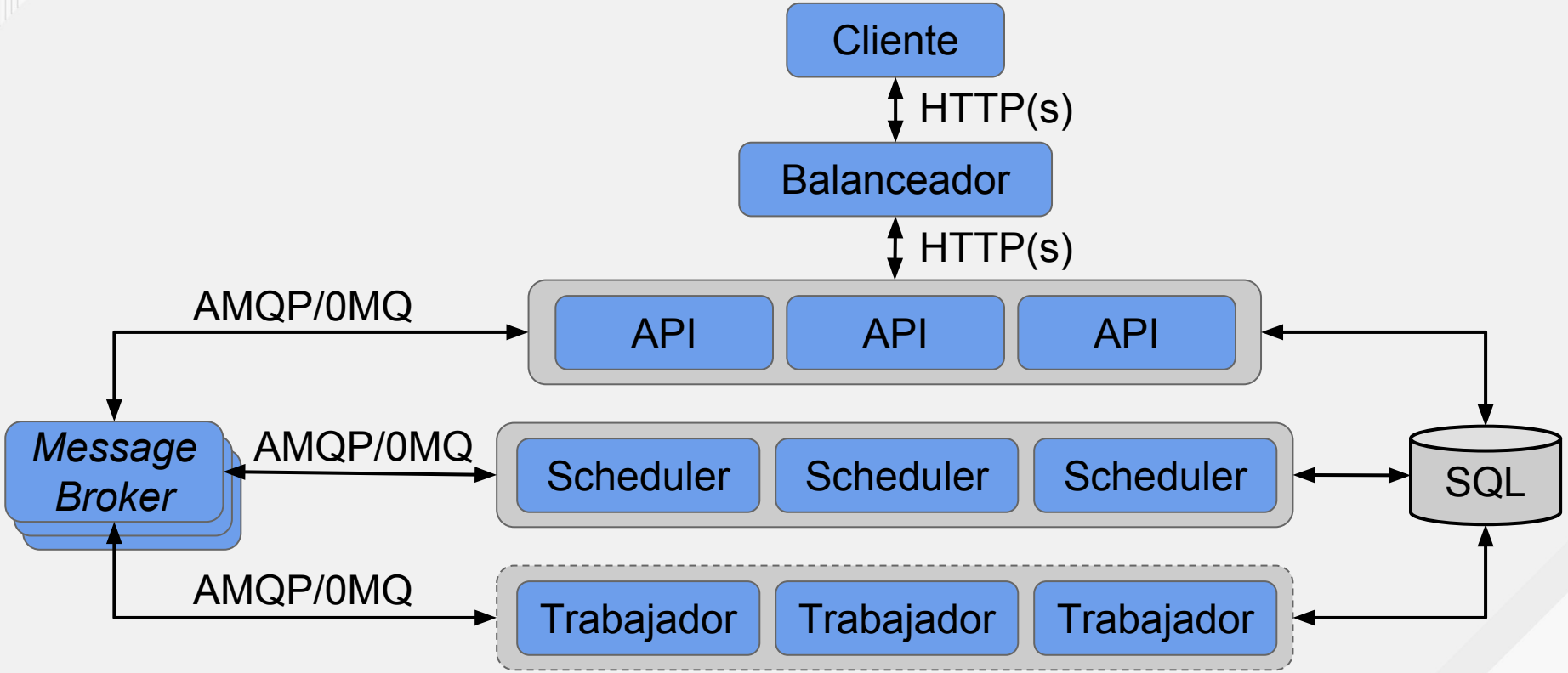
Arquitectura - API - llamadas RPC

- Oslo Messaging
- Tipos
  - Síncronas
    - Esperan a que completen las operaciones
    - API devuelve respuesta de la llamada
    - Cuidado timeouts de HAProxy
  - Asíncronas
    - Se añade la llamada a la cola de mensajería
    - No se espera a que se complete
    - API devuelve datos creados en la BBDD
    - Si no hay nodos para procesar la operación, se queda en el agente de mensajería
- La mayoría de las llamadas desde los clientes son asíncronas

Arquitectura - Trabajador

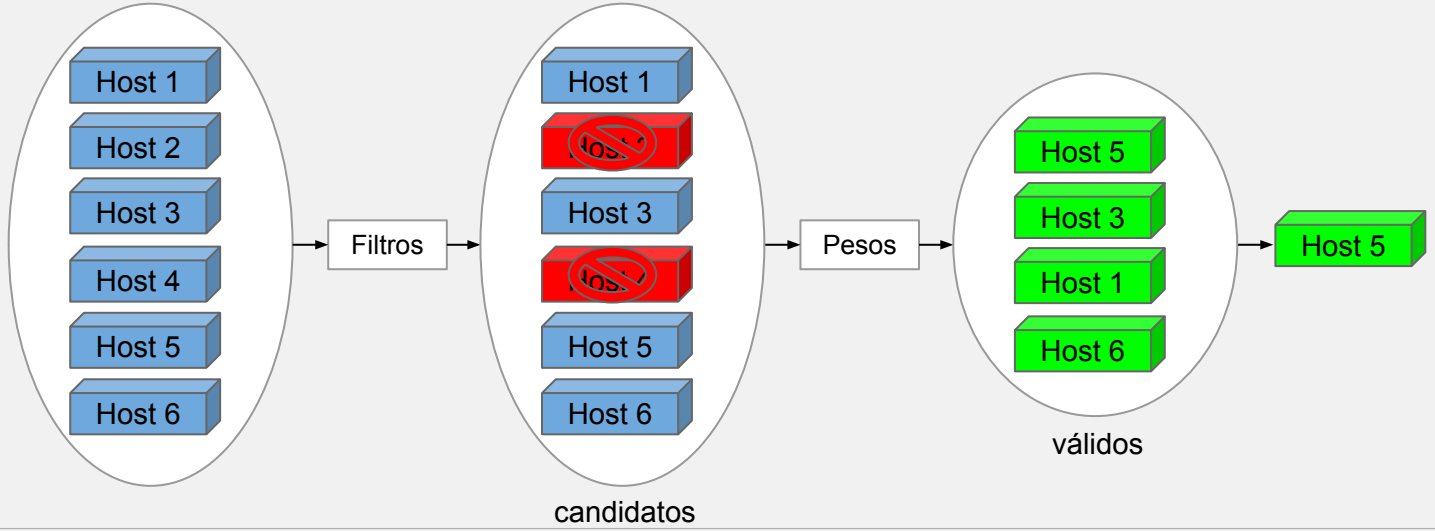
- Responsable de gestionar recursos
  - Almacenamiento
  - VMs
- Driver específico para cada recurso
- Distintos recursos simultáneos en el mismo nodo
  - Sólo posible en algunos componentes
  - Un proceso por tipo de recurso
- Usa *green threads* → Posibles problemas con librerías nativas
- Configuración Activa-Activa
  - No siempre posible → Recursos locales al nodo
  - Puede no estar soportado → Cinder

Arquitectura 2

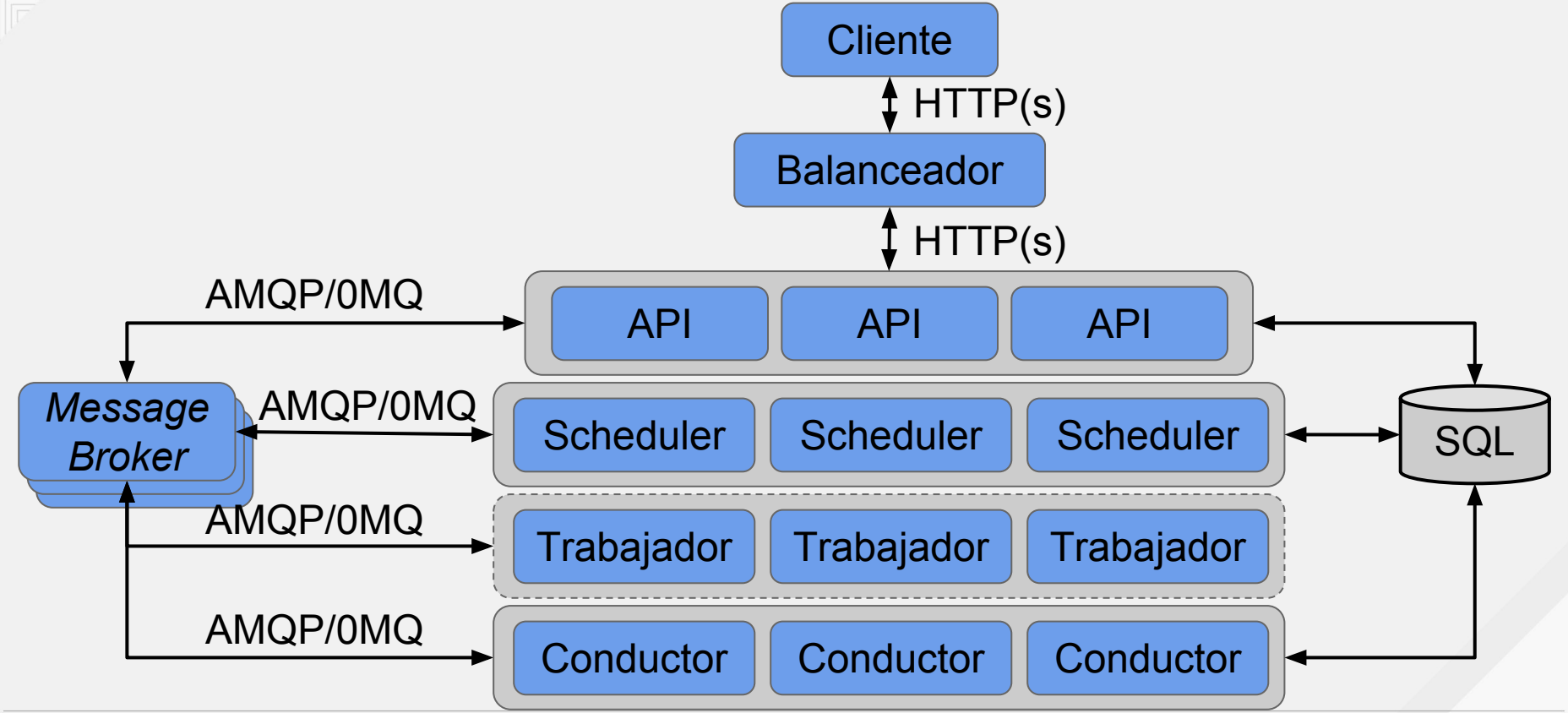


Arquitectura 2 - Scheduler

- Determina trabajador para procesar petición
  - Para volúmenes ⇒ Crear, migrar
  - Para instancias ⇒ Ejecutar, migrar
- Decisión basada en filtros y pesos: *Availability zone*, *Extra Specs*, Afinidad, etc.
- Envía la petición por RPC al trabajador seleccionado



Arquitectura 3



Arquitectura 3 - Conductor

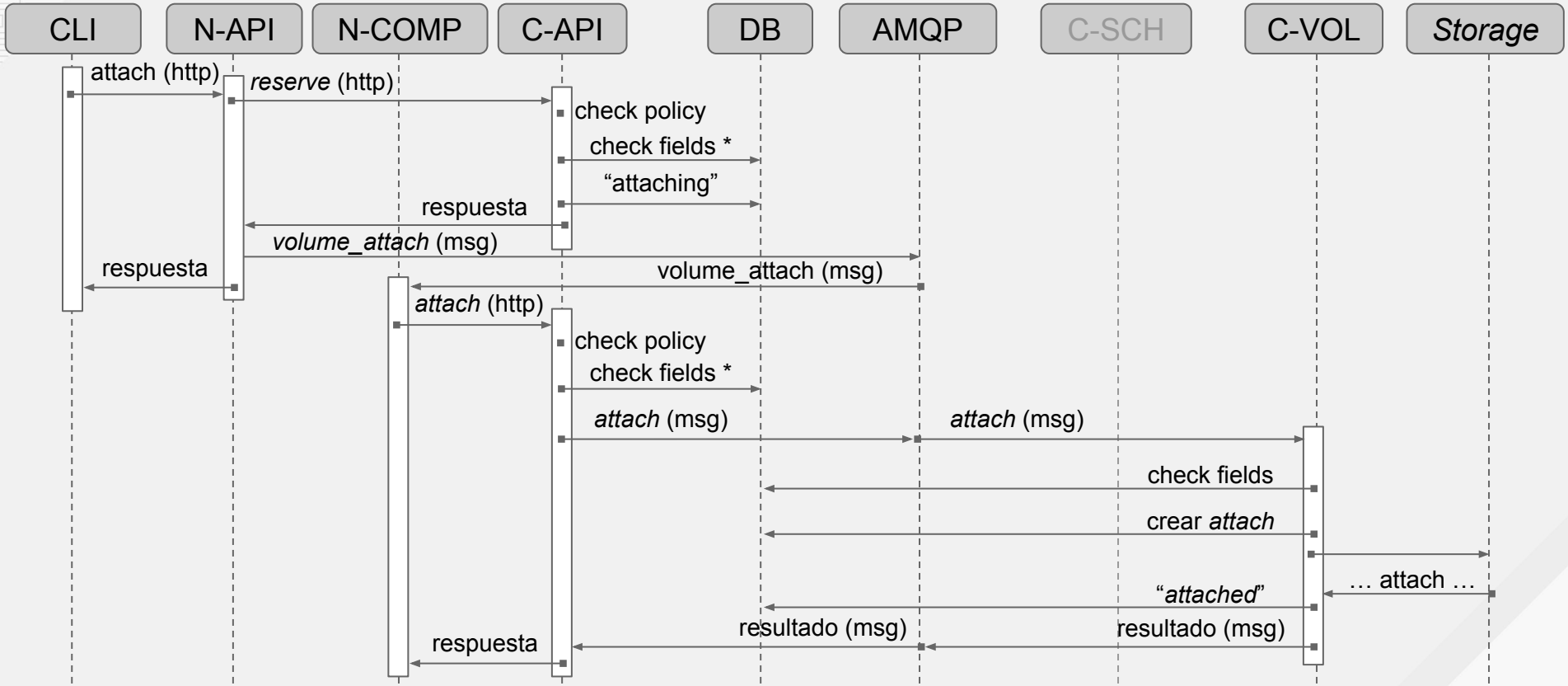
- En Nova - Cinder a lo mejor en el futuro
- Gestiona acceso a la BBDD
  - Restringe acceso
  - Para los trabajadores
  - Mayor seguridad
- Originalmente peticiones simples
- En el futuro
  - Los trabajadores serán menos inteligentes
  - Lógica de operaciones complejas
    - Migraciones
    - Resizes

¿En qué me ayuda saber esto?

Utilidad (II)

- Conocer además los flujos facilita
  - Entender el comportamiento de los componentes
  - Navegar los logs
  - Localizar problemas
  - Generalización de problemas comunes

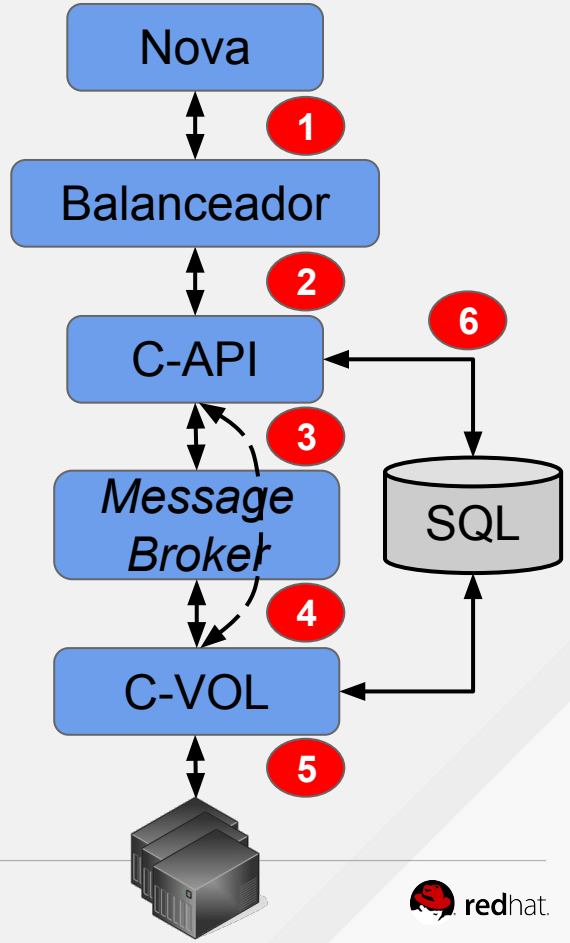
Flujo volume-attach



* Posibles condiciones de carrera. Ahora hacemos actualizaciones condicionadas

Timeouts - Cinder attach

1. *cinderclient*: timeout de peticiones por defecto infinito
2. *timeout server* → La operación continúa
3. - rabbit: Pasado a Kombu *[oslo.messaging]*
     - *rabbit_max_retries* (def: infinito)
     - *rabbit_retry_interval* (def: 1)
     - *rabbit_retry_backoff* (def: 2)
     - *rabbit_interval_max* (def: 30 seg)
   - *zmq: rpc_cast_timeout, rpc_poll_timeout… [oslo.messaging]*
4. *rpc_response_timeout* (def: 60 seg) *[oslo.messaging]*
     - La operación continúa
5. - Genéricos:
     - *migration_create_volume_timeout_secs* (def: 300 seg)
     - *glance* (def: infinito)
   - Específicos de cada Driver:
     - emc → *default_timeout*
     - rbd → *rados_connect_timeout*
6. *pool_timeout* (def: 30 seg)



Utilidad

- Mismas configuraciones para las librerías comunes
- Oslo.config
  - Secciones: *[DEFAULT] [keystone_authtoken] [oslo_concurrency]*
  - No hay propagación de [DEFAULT] a secciones dinámicas ⇒ Cinder multi-backend
  - No avisa de *keys* que no existen
  - --config-dir --config-file
    - Orden es importante
    - `/usr/bin/cinder-api --config-file /usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf --logfile /var/log/cinder/ap…`
- Oslo.services: Proporción greenthreads a conexiones a la BBDD
  - 1000 greenthreads vs. 15 conexiones a la BBDD
    - *wsgi_default_pool_size* vs. *max_pool_size* + *max_overflow*
  - No usar valor por defecto de *max_overflow*

redhat.

GRACIAS

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos