CACHING

Vamos a ver algunos ejemplos de caching que podemos encontrarnos en el despliegue de una nube Cloud Foundry

- Cacheo de imágenes docker
- Cacheo de recursos y buildpacks
- Cacheo de LRP's en la BBS
- Cacheo de Logs y métricas en Loggregator
- Escalado de Cache de Logs
- Cacheo de micro servicios con Pivotal Cloud Cache

1. CACHEO DE IMÁGENES DOCKER

Lo primero que cabe preguntarse es: ¿Por qué tener las imágenes Docker cacheadas en Cloud Foundry?

La respuesta es sencilla, teniéndolas cacheadas en Cloud Foundry, le ahorramos tiempo al desarrollador cada vez que éste quiera ejecutar en la nube Cloud Foundry la aplicación de la imagen Docker, desplegada en un contenedor (contenizada).

Es decir, de no haber caché, cada vez que se quiera ejecutar en la nube Cloud Foundry la aplicación del desarrollador, Cloud Foundry tendría que llevar a cabo el proceso de autentificarse en el registro Docker (si es privado) donde se haya la imagen Docker que contiene la aplicación para posteriormente traérsela a la nube Cloud Foundry y ponerla en funcionamiento en tiempo de ejecución a través de su despliegue en un contenedor (contenización), que es llevado a cabo por el backend "Garden-runC" del subsistema Diego que es el que crea los contenedores donde se ejecutarán las aplicaciones dockerizadas.

Sin embargo gracias a la caché, concretamente la Diego-Docker-Caché, se trae la imagen una primera vez a Cloud Foundry desde el registro Docker donde se haya la imagen y se cachea, para que las posteriores veces que haga falta se acceda directamente a ella desde la caché.

Las imágenes Docker se cachean subdividiéndolas en capas. Cuando el backend "garden" quiera recuperar la imagen, sólo necesita recuperar todas las capas de la caché y luego usar las bibliotecas de Docker para unirlas y montarlas como un sistema de archivos raíz.

Otra ventaja de esto es que garantiza la disponibilidad de las imágenes sin depender de la disponibilidad de DockerHub o cualquier otro registro/repositorio de imágenes Docker donde se hallase inicialmente la imagen Docker.

Si éste está caído/inaccesible o surge cualquier otro problema de disponibilidad, siempre se puede recuperar de la Diego-Docker-Caché.

También se garantiza mayor facilidad de escalado para nuestras aplicaciones, ya que al tenerlas cacheadas no tenemos que depender del registro docker para poder instanciarlas y llevar a cabo un escalado, tan solo hace falta recuperarla de la cache e instanciarlas.

Diego será el encargado de indicarle a Garden que la imagen a extraer es la que se halla en caché en vez de la que está en el registro remoto. Esto tiene la ventaja adicional de asegurarse

de que siempre esté ejecutando exactamente la imagen Docker que el desarrollador montó, en lugar de algo que puede haber cambiado en el registro remoto desde donde nos trajimos por primera vez la imagen docker.

En definitiva, a lo que buscamos dar respuesta es:

- Una aplicación que no podemos iniciar en Cloud Foundry puesto que el registro Docker Hub donde se haya la imagen docker tiene problemas de disponibilidad.

- La aplicación se ha estropeado porque alguien ha cambiado la imagen docker del repositorio, de la cual dependíamos.

- Se tarda mucho tiempo en instanciar la aplicación ya que dependemos del Docker Hub para traérnosla.

2. CACHEO DE RECURSOS Y BUILDPACKS

También conocido como cacheo de archivos dependientes de una aplicación y cacheo de paquetes de compilación.

Sabemos que el Cloud Controller (CC) contiene un BLOBSTORE, esto es un almacén para objetos binarios de gran tamaño.

Aquí se guardan droplets, imágenes contenizadas, buildpacks (paquetes de compilación), resource files (archivos de recursos) y “Apps Code Packages”(Páquetes de código de aplicación).

Cuando los archivos de recursos (application files -> archivos de aplicaciones), son cargados al Cloud Controller, justamente después son cacheados en la BLOBSTORE haciendo uso del algoritmo de hasheo “SHA” (HASH -> archivo), de esta manera se garantiza la reusabilidad de dichos archivos sin necesidad de tener que volver a subirlos al controlador.

El funcionamiento de la caché de recursos es el siguiente: antes de cargar todos los archivos de la aplicación, el CLI de Cloud Foundry emite un archivo de solicitud de coincidencias de recursos al Cloud Controller para determinar si alguno de los archivos de la aplicación ya existe en la caché de recursos (blobstore) del Cloud Controller.

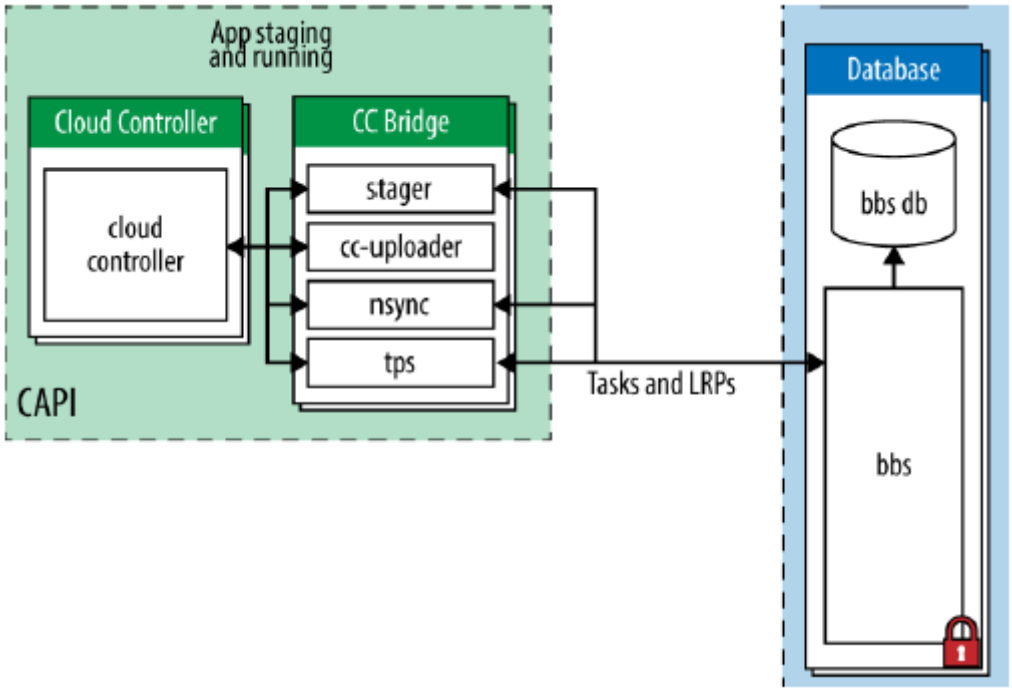
Cuando se cargan los archivos/ficheros de la aplicación, la CLI de Cloud Foundry omite los archivos que existen en el caché de recursos proporcionando el resultado de la solicitud de coincidencia de recursos.

Se trata de archivos grandes de paquetes de aplicaciones que Cloud Controller almacena con un algoritmo de hasheo “SHA” para su reutilización posterior. Para ahorrar ancho de banda, la Interfaz de línea de comandos de Cloud Foundry (Cloud Foundry CLI) solo carga archivos de aplicaciones grandes que el Cloud Controller aún no ha almacenado en la caché de recursos.

Los archivos/ficheros de aplicación cargados se combinan con el fichero de la caché de recursos para crear el paquete completo de la aplicación.

En la blobstore también se cachean buildpacks, concretamente archivos grandes que los paquetes de compilación generan durante la preparación, almacenados para su posterior reutilización. Este caché permite que los paquetes de compilación se ejecuten más rápidamente cuando se almacenan aplicaciones que se han almacenado previamente.

3. CACHEO DE LRP'S EN LA BBS



Se entiende por LRP los Long Range Process, procesos de larga duración, los "Desired" son los deseados/esperados mientras que los "Actual" son los actuales.

Se entiende por "Task" las tareas.

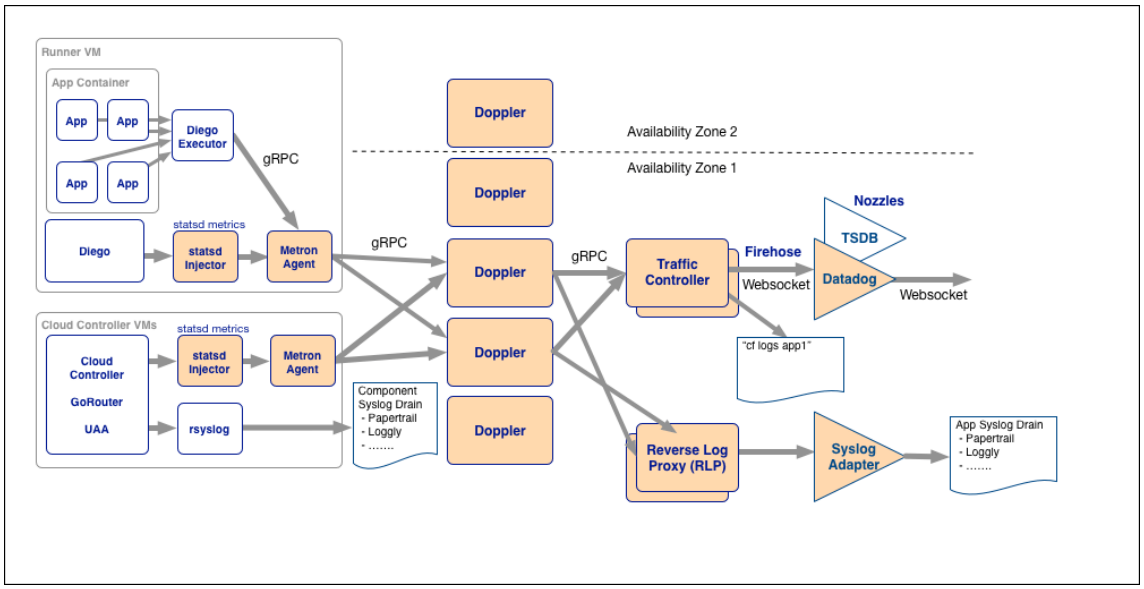
El BBS (Bulletin Board System/Sistema de tablón de anuncios) es uno de los componentes de Diego. Maneja la base de datos de Diego, manteniendo una caché actualizada del estado en tiempo real del cluster de Diego, incluyendo una representación del momento de todas las Desired LRP (Los procesos de larga duración deseados), las instancias Actual LRP (Los procesos de larga duración que están actualmente en ejecución) que se están ejecutando, y las Tareas a bordo.

BBS está constantemente monitorizando si las LRP actuales son iguales que las LRP deseadas. Hablamos de una comparación periódica que lleva acabo BBS, y cuando las actuales son menores que las deseadas es porque existe una deficiencia y por tanto BBS se pone en funcionamiento llevando a cabo una nueva subasta de tareas y LRP's. Cuando existe un excedente, mata instancias de tareas o LRP's.

En cualquier caso, BBS tiene que llevar a cabo constantemente esta monitorización/comparación, de ahí la importancia de que estén cacheadas y mantener una caché actualizada para poder consultarlas más rápidamente y saber si hay una deficiencia o un excedente.

Cuando hay una deficiencia, se llama al subastador de diego el cual se encarga de asignar nuevos procesos a las máquinas virtuales y equilibrar la carga de éstas.

4. CACHEO DE LOGS Y MÉTRICAS EN LOGGREGATOR



La arquitectura “Loggregator” es un subsistema de la implementación Cloud Foundry que se encarga de recopilar logs y métricas de aplicaciones de usuario que estén corriendo en una nube de Cloud Foundry. También registra logs del funcionamiento de los componentes de la implementación de Cloud Foundry.

Los logs y métricas son generados por los agentes Metrón (“Metron Agents”), que están colocados en las VM’s contenedoras de aplicaciones en funcionamiento.

Los logs son reportes de eventos detectados, eventos que requieren tomar alguna medida, errores que surgen y cualquier otro mensaje que el desarrollador ha querido que se generase por alguna razón intrínseca a éste.

Las métricas son mediciones que se hacen sobre VM’s contenedoras de aplicaciones, así como el estado de dichos contenedores.

Todos estos logs y métricas se mandan a los servidores “Dropplers” que contienen la cache, se cachean los logs y tras esto se envían al controlador de tráfico (“Traffic Controller”).

El controlador de tráfico recibe los logs de todos los servidores “Dropplers” y maneja las solicitudes de logs llevadas a cabo por el cliente interesado en dichos logs. Proporciona una API externa.

Luego el “Firehose” (manguera) se encarga de streamear (flujo) dichos logs y métricas provenientes del controlador de tráfico.

Los datos del flujo de la manguera no son constantes, van pasando y se van borrando, de ahí la importancia de la caché de logs ubicada en los servidores dropplers y que almacena los datos provenientes del loggregator. Dicha caché permite ver/consultar los datos desde la manguera y durante un periodo de tiempo. La cache proporciona una interfaz RESTful para recuperar los logs.

Todos estos datos pueden ser de gran utilidad, y es interesante el hecho de que puedan ser analizados por algún servicio que se encargue de ello.

La caché de registro está situada en la máquina virtual Doppler. Las dependencias de dicha caché son las siguientes:

- La caché de logs depende de **Loggregator** para ver y filtrar los logs, por lo que su fiabilidad depende del rendimiento de Loggregator. La caché de logs utiliza la memoria disponible en un dispositivo para almacenar logs y, por lo tanto, puede afectar al rendimiento del dispositivo durante períodos de alta contención de memoria.

- La caché de logs está ubicada en las **máquinas virtuales Dopplers.** Acelera la recuperación de datos del sistema Loggregator, especialmente para implementaciones Cloud Foundry que cuentan con un gran número de Dopplers.

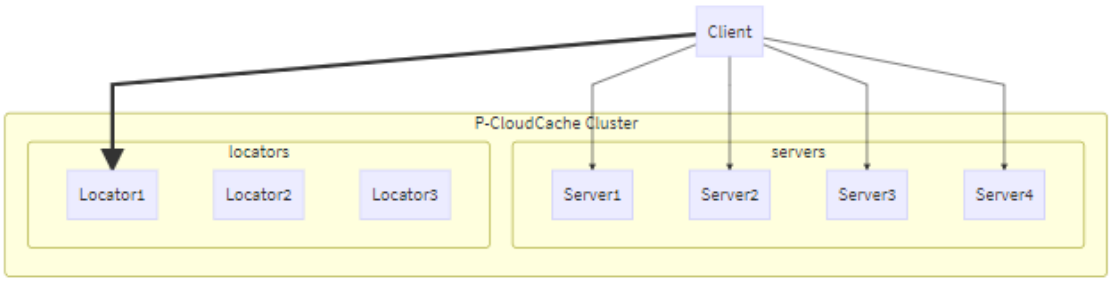
5. ESCALADO DE CACHE DE LOGS

Tal y como hemos mencionado anteriormente, se consigue una aceleración en la recuperación de logs en aquellos despliegues Cloud Foundry que cuentan con muchas VM's Dopplers, ergo escalar la caché de logs consiste en incrementar el número de máquinas virtuales Dopplers, de esta manera aceleramos la recuperación de logs.

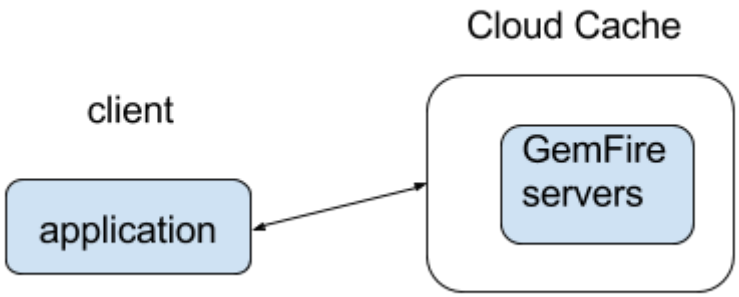
Estamos hablando de escalar la arquitectura, es decir, un componente del despliegue Cloud Foundry.

6. CACHEO DE MICRO SERVICIOS CON PIVOTAL CLOUD CACHE

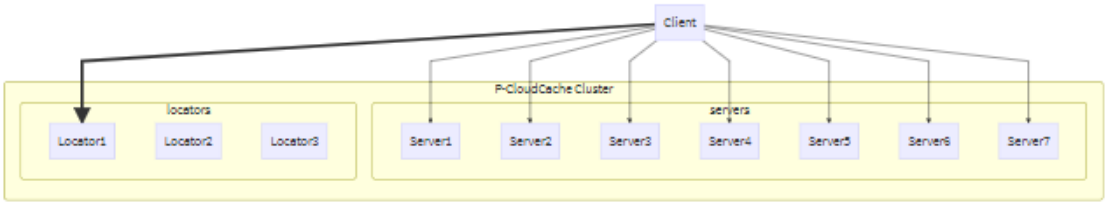
La proveedora de Cloud Foundry, Pivotal Software, ofrece para Pivotal Cloud Foundry una solución de caching para nuestras aplicaciones desplegadas en la nube. Esta solución está basada en el clustering. Se trata de un clúster basado en servidores "GemFire" para proporcionar alta disponibilidad, garantías de replicación y consistencia eventual. Cada uno de éstos proporciona un almacén de datos consistente, de baja latencia y tolerante a fallos.



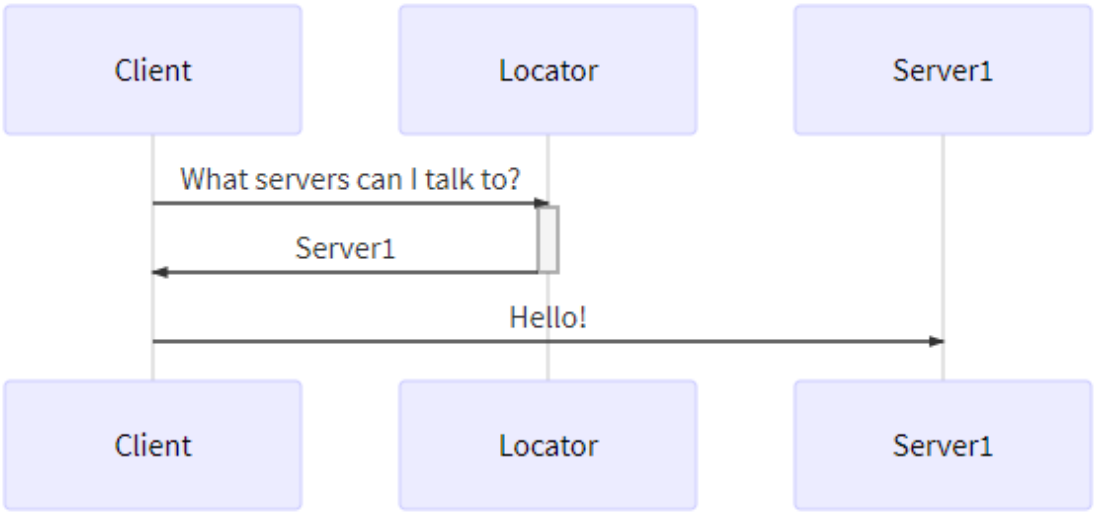
Un server “GemFire” mantiene los datos en pares clave/valor. Cada par se llama “entrada”. Las entradas se agrupan lógicamente en conjuntos llamados regiones. Una región es una estructura de datos de un mapa (o diccionario).

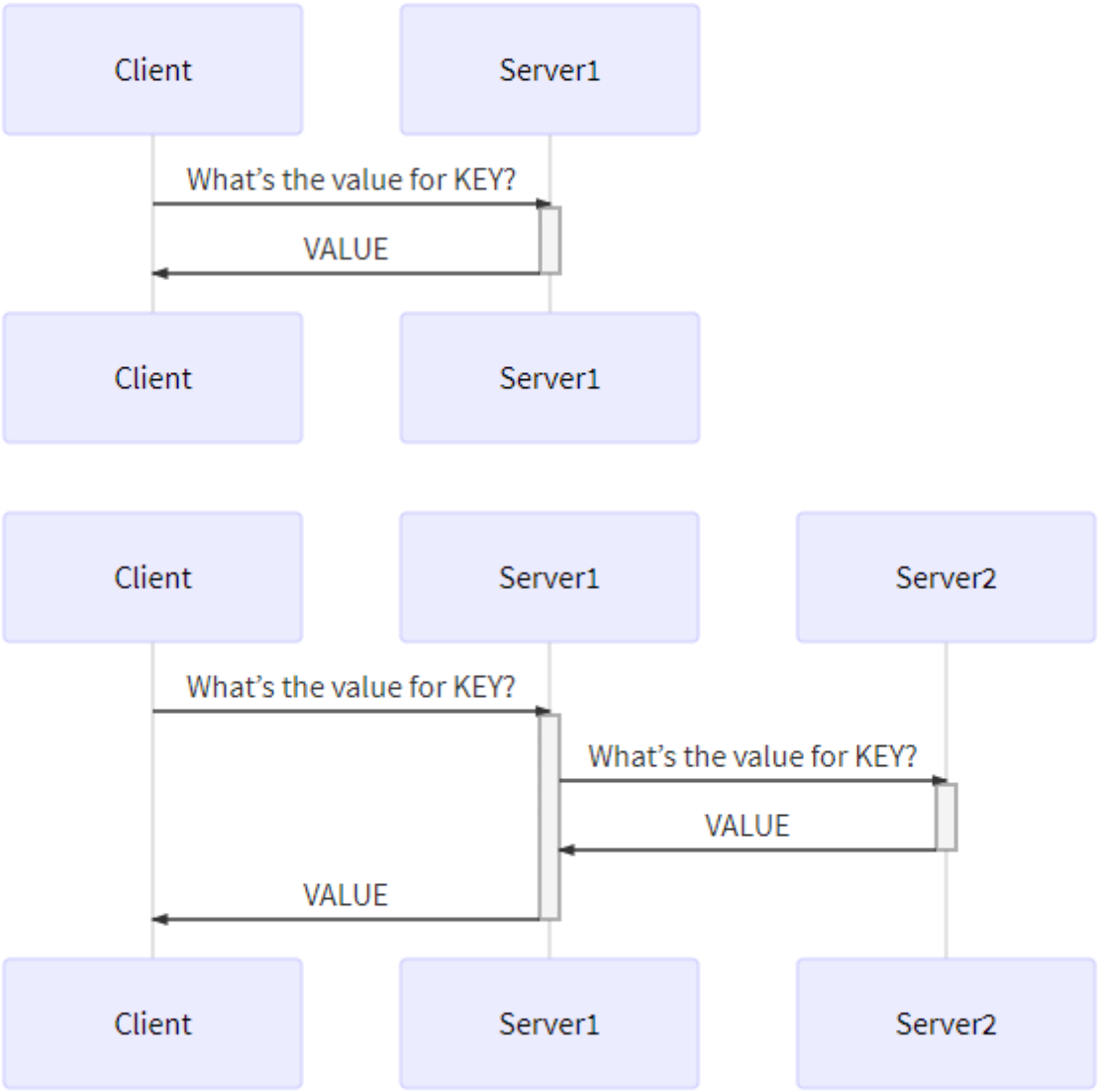


El clúster de servers “GemFire” es escalable, aumentando así el número de servers “GemFire”, lo que aumenta la capacidad de la memoria caché.



Localizadores hay siempre tres, independientemente de que se escale o no se haga. Se llama localizador porque precisamente su función es localizar la ubicación de un server “GemFire” disponible cuando nosotros como servicio, hacemos una petición de datos a caché. Una vez que nos estemos comunicando con el server, le pedimos los datos y si éste no los tiene, le pregunta a otro server del clúster a ver si los tiene.





Simple pero efectivo, se trata de una arquitectura cliente-servidor

REFERENCIAS

https://docs.cloudfoundry.org/concepts/diego/diego-architecture.html

https://docs.cloudfoundry.org/concepts/architecture/cloud-controller.html

https://docs.cloudfoundry.org/concepts/how-applications-are-staged.html **(Punto 4, para cacheo de archivos de aplicación y paquetes de compilación [buildpacks])**

https://assets.dynatrace.com/en/docs/report/cloud-foundry-definitive-guide.pdf **(Páginas 34, 70, 87)**

https://www.altoros.com/blog/how-to-push-private-docker-images-and-enable-caching-on-cloud-foundry-diego/ **(Cacheo de imágenes docker)**

https://docs.cloudfoundry.org/loggregator/architecture.html **(Cacheo de logs y métricas)**

https://docs.pivotal.io/p-cloud-cache/1-6/architecture.html