Operations

Architecture

is composed of five main components(see Figure onshareddir): a modified Docker client, layer buffer, file cache, registries, backend storage system that implements deduplication. The modified Docker client sends put or pull layer/manifest requests. After it sends the pull layer requests. it receives multiple partial layers and decompresses them together as a whole uncompressed layer, then verifies the integrity of the uncompress layer by calculating the digest of the uncompress layer. Layer buffer is used to buffer all the put layer requests and cache prefetched layers for later use. Layer buffer can be implemented on distributed in-memory store, Plusma. File cache is a flash cache that implements file-level deduplication which stores unique files that belong to a certain amount of layers. File cache is implemented on distributed log structured store, CORFU. The unique files after removing duplicate files are flash-friendly workload because the files can be appended as a log and there is no modification to these files, meaning that there is no small write. Backend storage system with deduplication feature stores the layers and does deduplication. Many modern storage system withs deduplication feature can be used as our backend storage system. However, the only requirement is that the storage systems can recognize compressed layer file and decompress them before running deduplication. And the storage systems also can restore partial compressed layer file in parallel can send them back to clients. Note that there is no consistency issue between either layer buffer, file cache or backend storage because layers won't be changed in the future.

At a high-level, registries act like a distributed cache which sits close to clients to provide fast performance. Layer buffer caches hot layers for active users. To improve the capacity limitation of main memory cache, we utilize flash fast random read access and store unique files on a distributed flash-based store.

Workflow The registry API is almost the same as original registry. The user simply pushes and pulles image to registry.

Push. After receiving a layer from the client, first buffers the layer in layer buffer. Meantime, will also submitted this layer to the backend storage system. Our layer buffer and file cache use write through policies and since there is no modification to the layer and files, so there is no data consistency issue between cache and backend storage system. Storing a layer in layer buffer might trigger cold layer eviction. The cold layer will first be evicted to file cache and deduplicated. A deduplication process performs the following steps for every victim layer that is evicted from layer buffer. compactenumerate

d ecompress and unpack the layer's tarball into individual files;

c ompute a fingerprint for every file in the layer;

c heck all file fingerprints against the file index to identify if identical files are already stored in the file cache;

s tore non-deduplicated files in 's file cache and update the file index with file fingerprint and file location along with its host address;

c reate and store a layer recipe that includes the file path, metadata, fingerprint of every file in the layer;

r emove the layer's tarball from the layer buffer.