Back To Module Home

Basic Building Blocks for Modern System Design

0% completed

Search Module

Introduction to Building Blocks

Domain Name System

Load balancers

Cache

Databases

Key-value Store

Content Delivery Network (CDN)

Sequencer

Distributed Monitoring

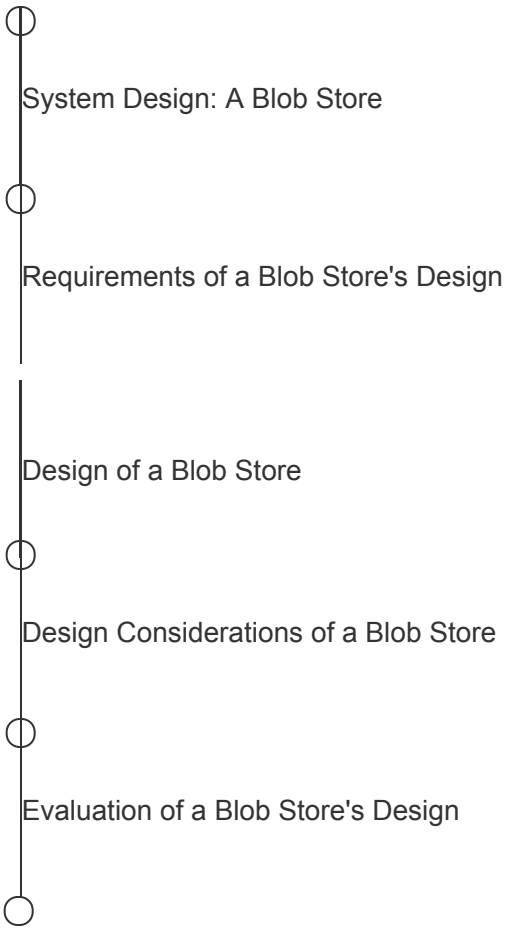
Distributed Cache

Distributed Messaging Queue

Pub-sub

Rate Limiter

Blob Store



Distributed Search

Distributed Task Scheduler

Sharded Counters

Conclusion

Mark Module as Completed

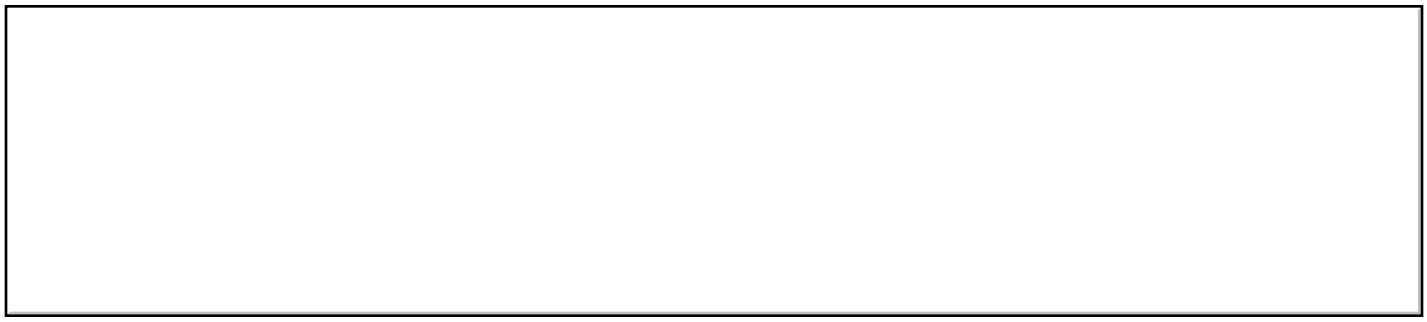
Design of a Blob Store

Learn how to incorporate certain requirements into the design of a blob store.

We'll cover the following 

High-level design#

Let’s identify and connect the components of a blob store system. At a high level, the components are clients, front-end servers, and storage disks.



The high-level design of a blob store

The client's requests are received at the front-end servers that process the request. The front-end servers store the client's blob onto the storage disks attached to them.

API design#

Let's look into the API design of the blob store. All of the functions below can only be performed by a registered and authenticated user. For the sake of brevity, we don't include functionalities like registration and authentication of users.

Create container

The `createContainer` operation creates a new container under the logged-in account from which this request is being generated.

```
createContainer(containerName)
```

Parameter	Description
`containerName`	This is the name of the container. It should be unique within a storage account.

Upload blobs

The client's data is stored in the form of Bytes in the blob store. The data can be put into a container with the following code:

```
putBlob(containerPath, blobName, data)
```

Parameter	Description
`containerPath`	This is the path of the container in which we upload the blob. It consists of the `accountID` `containerID`.
`blobName`	This is the name of the blob. It should be unique within a container, otherwise our system blob that was uploaded later a version number.
`data`	This is a file that the user wants to upload to the blob store.

> **Note:** This API is just a logical way to spell out needs. We might use a multistep streaming call for actual implementation if the data size is very large.

Download blobs

Blobs are identified by their unique name or ID.

```
getBlob(blobPath)
```

Parameter	Description
`blobPath`	This is the fully qualified path of the data or file, including its unique ID.

Delete blob

The `deleteBlob` operation marks the specified blob for deletion. The actual blob is deleted during garbage collection.

```
deleteBlob(blobPath)
```

Parameter	Description
`blobPath`	This is the path of the blob that the user wants to delete.

List blobs

The `listBlobs` operation returns a list of blobs under the specified container or path.

```
listBlobs(containerPath)
```

Parameter	Description
`containerPath`	This is the path to the container from which the user wants to get the list of blobs.

Delete container

The `deleteContainer` operation marks the specified container for deletion. The container and any blobs in it are deleted later during garbage collection.

```
deleteContainer(containerPath)
```

Parameter	Description
`containerPath`	This is the path to the container that the user wants to delete.

List containers

The `listContainers` operation returns a list of the containers under the specified user's blob store account.

```
listContainers(accountID)
```

Parameter	Description
`accountID`	This is the ID of the user who wants to list their containers.

> Note: The APIs used to retrieve blobs provide metadata containing size, version number, access privileges, name, and so on.

Detailed design#

We start this section by identifying the key components that we need to complete our blob store design. Then, we look at how these components connect to fulfill our functional requirements.

Components#

Here is a list of components that we use in the blob store design:

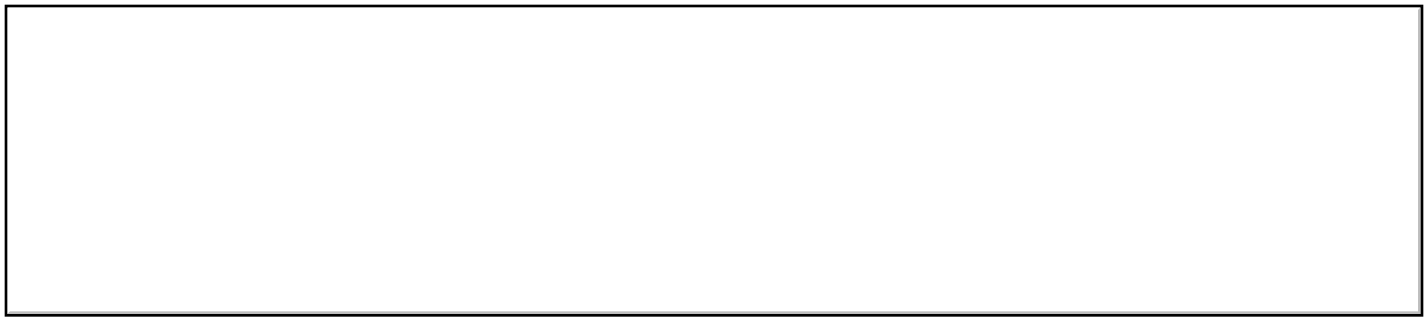
- **Client:** This is a user or program that performs any of the API functions that are specified.
- **Rate limiter:** A rate limiter limits the number of requests based on the user's subscription or limits the number of requests from the same IP address at the same time. It doesn't allow users to exceed the predefined limit.
- **Load balancer:** A load balancer distributes incoming network traffic among a group of servers. It's also used to reroute requests to different regions depending on the location of the user, different data centers within the same region, or different servers within the same data center. **DNS** load balancing can be used to reroute the requests among different regions based on the location of the user.
- **Front-end servers:** Front-end servers forward the users' requests for adding or deleting data to the appropriate storage servers.
- **Data nodes:** Data nodes hold the actual blob data. It's also possible that they contain a part of the blob's data. Blobs are split into small, fixed-size pieces called **chunks**. A data node can accommodate all of the chunks of a blob or at least some of them.

Master node: A master node is the core component that manages all data nodes. It stores information about storage paths and the access privileges of blobs. There are two types of access privileges: private and public. A *private* access privilege means that the blob is only accessible by the account containing that blob. A *public* access privilege means that anyone can access that blob.

> **Note:** Each of the data nodes in the cluster send the master node a heartbeat and a chunk report regularly. The presence of a **heartbeat** indicates that the data node is operational. A **chunk report** lists all the chunks on a data node. If a data node fails to send a heartbeat, the master node considers that node dead and then processes the user requests on the replica nodes. The master node maintains a log of pending operations that should be replayed on the dead data node when it recovers.

- **Metadata storage:** Metadata storage is a distributed database that's used by the master node to store all the metadata. Metadata consists of account metadata, container metadata, and blob metadata.
  - ***Account metadata*** contains the account information for each user and the containers held by each account.
  - ***Container metadata*** consists of the list of the blobs in each container.
  - ***Blob metadata*** consists of where each blob is stored. The blob metadata is discussed in detail in the next lesson.
- **Monitoring service:** A monitoring service monitors the data nodes and the master node. It alerts the administrator in case of disk failures that require human intervention. It also gets information about the total available space left on the disks to alert administrators to add more disks.
- **Administrator:** An administrator is responsible for handling notifications from the monitoring services and conducting routine checkups of the overall service to ensure reliability.

The architecture of how these components interconnect is shown in the diagram below:



The detailed design of a blob store

Workflow#

We describe the workflow based on the basic operations we can perform on a blob store. We assume that the user has successfully logged in and a container has already been created. A unique ID is assigned to each user and container. The user performs the following operations in a specific container.

Write a blob

1. The client generates the upload blob request. If the client's request successfully passes through the rate limiter, the load balancer forwards the client's request to one of the front-end servers. The front-end server then requests the master node for the data nodes it should contact to store the blob.
2. The master node assigns the blob a unique ID using a unique ID generator system. It then splits the large-size blob into smaller, fixed-size chunks and assigns each chunk a data node where that chunk is eventually stored. The master node determines the amount of storage space that's available on the data nodes using a **free-space management system**.
3. After determining the mapping of chunks to data nodes, the front-end servers write the chunks to the assigned data nodes.
4. We replicate each chunk for redundancy purposes. All choices regarding chunk replication are made at the master node. Hence, the master node also allocates the storage and data nodes for storing replicas.
5. The master node stores the blob metadata in the metadata storage. We discuss the

blob's metadata schema in detail in the next lesson.

6. After writing the blob, a fully qualified path of the blob is returned to the client. The path consists of the user ID, container ID where the user has added the blob, the blob ID, and the access level of the blob.

Point to Ponder

Question

What does the master node do if a user concurrently writes two blobs with the same name inside the same container?

Show Answer

Reading a blob

1. When a read request for a blob reaches the front-end server, it asks the master node for that blob's metadata.
2. The master node first checks whether that blob is private or public, based on the path of the blob and whether we're authorized to transfer that blob or not.
3. After authorizing the blob, the master node looks for the chunks for that blob in the metadata and looks at their mappings to data nodes. The master node returns the chunks and their mappings (data nodes) to the client.
4. The client then reads the chunk data from the data nodes.

> **Note:** The metadata information for reading the blob is cached at the client machine, so that the next time a client wants to read the same blob, we won't have to burden the master node. Additionally, the client's read operation will be

> faster the next time.

Point to Ponder

Question

Suppose the master node moves data from one data node to another because of an impending disk failure. The user will now have stale information if they use the cached metadata to access the data. How do we handle such situations?

Show Answer

Deleting a blob Upon receiving a delete blob request, the master node marks that blob as deleted in the metadata, and frees up the space later using a garbage collector. We learn more about garbage collectors in the next lesson.

Point to Ponder

Question

Can the master node be considered a single point of failure? If yes, then how can we cope with this problem?

Show Answer

In the next lesson, we talk about the design considerations of a blob store.

Requirements of a Blob Store's Design

Next

Design Considerations of a Blob Store

Mark as Completed

Report an Issue