**System Component:**

**Problem Being Solved:**

**Purpose/Function:**

**Requirements:**

Updated June 28, 2018

## Architecture

The DTP solution was designed so that it is easy for a Provider to adopt and enable the standards, and took into consideration the following constraints:

* **Use existing standards; don’t create new ones**

By supporting existing standards where possible (like OAuth and REST), we aim to minimize the foundational work required before DTP can be built and put into action. Widespread adoption and understanding of existing standards makes this possible. As new standards are developed and adopted, they will be reviewed and, where applicable, incorporated into DTP (see the alternatives section).

* **Minimize the work required to add a service**

We designed DTP to allow Providers to participate without impacting their own core infrastructure. Providers can build Adapters and enable import and export functionality that works with their existing APIs and Authorization mechanisms.

* **Support standard deployment infrastructure**DTP infrastructure was designed with the flexibility to be hosted by anyone, anywhere. See the [Deployment Models](https://docs.google.com/document/d/12CLdzwRydNu4DsMFD6BhSd3yJnID6GJT35bv49H39zI/edit#heading=h.wz70jv7u76o4) for more details.

## System Components

As noted above, the system comprises three main components:

* **Data Models** are the canonical formats that establish a common understanding of how to transfer data.
* **Adapters** provide a method for converting each Provider’s proprietary data and authentication formats into a form that is usable by the system.
* **Task Management Library** provides the plumbing to power the system.

### Data Models

The Data Model for transferring files consists of two parts: a file type and the additional metadata needed by the receiving Provider to accurately import the data. For example with photos, the file type might be a standard format such as JPEG, and the metadata would include information such as title, description, album, and so forth.

Data Models are clustered together, typically by industry grouping, to form Verticals. A Provider could have data in one or more Verticals. Verticals could be photos, email, contacts, or music. Each Vertical has its own set of Data Models that enable seamless transfer of the relevant file types. For example, the Music vertical could have Data Models for music, playlists and videos.

Ideally, a Vertical will have a small number of well-defined and widely-adopted Data Models. In such a situation, the generally accepted standard will be used as the Data Model for that Vertical across companies. This is not currently the case for most Verticals because Data Models have emerged organically in a largely disconnected ecosystem.

One goal of DTP is to encourage organizations to use common Data Models in their systems, which will happen if organizations take importing and exporting data into consideration when initially designing their systems or providing updates. Using a common Data Model will significantly reduce the need for companies to maintain and update proprietary APIs.

In the case where there is no standard Data Model for a Vertical, companies will want to collaborate and agree upon standardized Data Models, either during DTP development or in collaboration with external standards bodies. Without collaboration, each Provider could have their own Data Model, and would have to create APIs and would have to create and maintain the same number of Data Models as there are companies in the Vertical, which would reduce the usefulness of DTP.

Even where standard Data Models do exist, collaboration will be an ongoing and mutually beneficial shared commitment as APIs will need to be maintained to handle new features, evolving standards, or innovative new formats.

|  |  |
| --- | --- |
|  |  |
| Without DTP Each Provider has to build and maintain adapters  for every other Provider’s proprietary APIs and, potentially, data formats | With DTP Each Provider only has to build and maintain an API  that supports the DTP Data Models, which are based on standard formats where available |

### Adapters

There are two main kinds of Adapters: Data Adapters and Authentication Adapters. These Adapters exist outside of a Provider’s core infrastructure and can be written either by the Provider itself, or by third parties that would like to enable data transfer to and from a Provider.

#### Data Adapters

Data Adapters are pieces of code that translate a given Provider’s APIs into Data Models used by DTP. Data Adapters come in pairs: an exporter that translates from the Provider’s API into the Data Model, and an importer that translates from the Data Model into the Provider’s API.

#### Authentication Adapters

Authentication Adapters are pieces of code that allow users to authenticate their accounts before transferring data out of or into another Provider. OAuth is likely to be the choice for most Providers, however DTP is agnostic to the type of authentication.

#### 

### Task Management

The rest is just plumbing.

The Task Management Libraries handle background tasks, such as calls between the two relevant Adapters, secure data storage, retry logic, rate limiting, pagination management, failure handling, and individual notifications.

DTP has developed a collection of Task Management Libraries as a reference implementation for how to utilize the Adapters to transfer data between two Providers. If preferred, Providers can choose to write their own implementation of the Task Management Libraries that utilize the Data Models and Adapters of DTP.

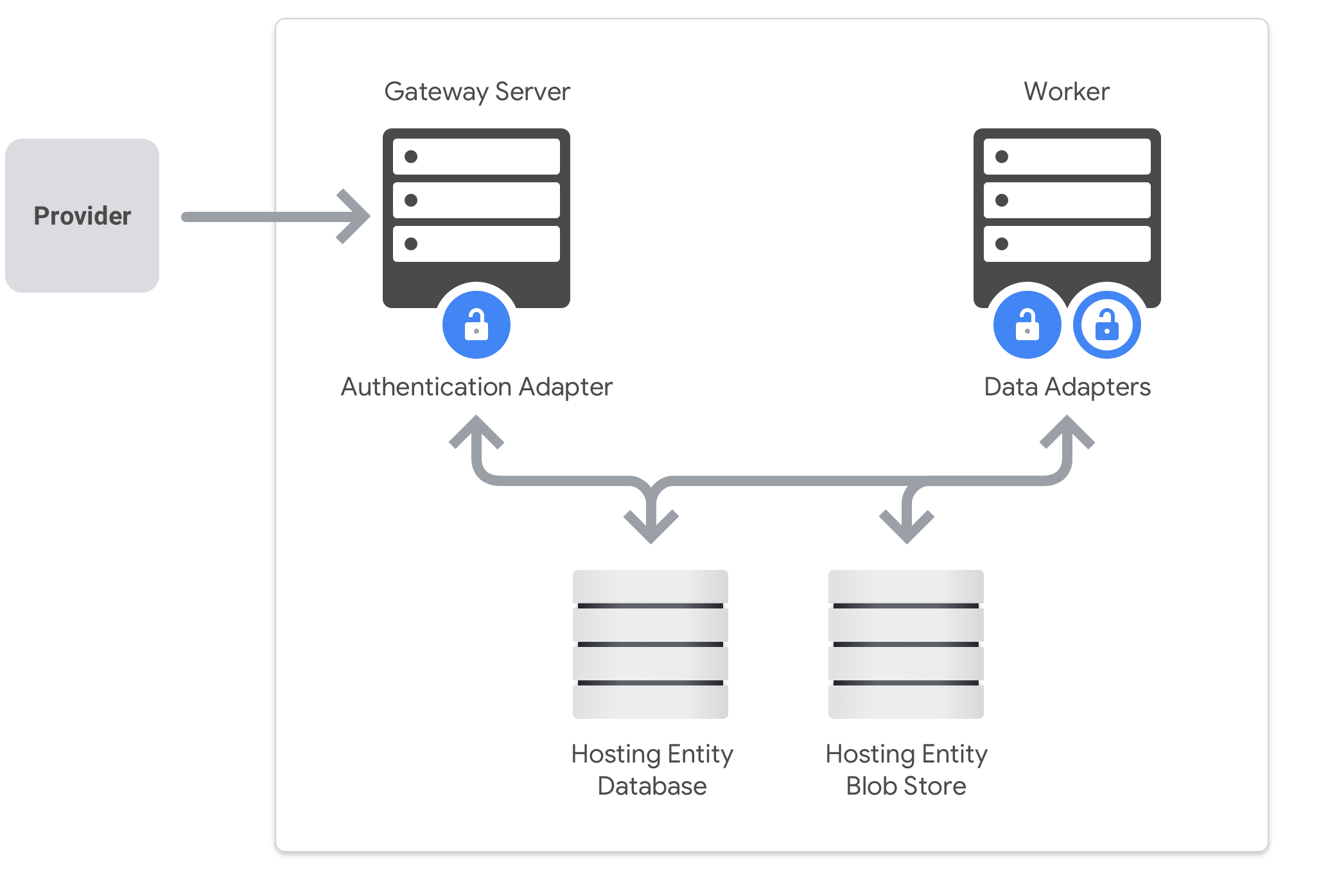


Figure 2: This graphic provides an overview of the interaction between the components of the DTP system. The gateway server facilitates the user’s authorization for data export and import via their Authentication Adapter and stores encrypted credential and metadata for the transfer request in the database. A worker process is assigned to a particular transfer request and invokes the Task Management Library to coordinate and execute export and import tasks, optionally, tentatively storing data in encrypted form in a blob store between export and import.

The Task Management Libraries are built on top of a generic cloud interface so that the Host Platform can be run locally, on a corporate production environment, or on a cloud platform. The cloud interface aims to use only high level abstractions so that it can be implemented on any vendor’s cloud platform.

#### Worker

The Worker utilizes the Task Management Library to execute the Adapters. The Worker is an isolated virtual machine that is created when a data transfer is initiated, and destroyed when that transfer is completed. The worker generates an ephemeral key when it is created, and that key is destroyed when the Worker is destroyed.

#### Securely Storing Data

The security of the data passing through the Data Transfer Project is critical. An important objective of DTP design is that Hosting Entities do not have access to a user’s data either in transit or at rest. DTP security follows industry best practices, and includes requiring transport layer security and storing all data encrypted with the ephemeral key generated by the Worker described above. DTP system design ensures that administrators do not have access to the encryption key, which protects a user’s data.

Details on security measures can be found in the core framework developer documentation on the Data Transfer Project GitHub (see Resources).

### System Access

Each Provider will maintain full control over determining who has access to the data stored on their systems. When a Hosting Entity runs a Host Platform of DTP, the Hosting Entity will need to request keys from each Provider they would like to be able to transfer data to and from. DTP will not mediate data access rights between Providers, nor provide API keys to any API.

This ensures that API quotas continue to be managed by the Provider, thereby helping to mitigate traffic spikes and negative impacts across Providers.

While anyone can contribute Adapters to DTP, each Hosting Entity decides which Providers they will interact with. Each Hosting Entity determines which Providers to request API keys from, and each Provider chooses which Hosting Entity they grant API keys to. When making these choices, the Hosting Entity should consider the privacy practices of the other Provider, as well as its reputation, and benefit to the user to ensure the data will be used appropriately. It is up to all actors in the portability ecosystem (Providers, Hosting Entities, Contributors, and Users) to be diligent in ensuring user data is handled safely.