Seamless Integrated Consistent Personalised

Enactor Training Course
Messaging and Data

Messaging and Data

Types of Data

Queue System Architecture

Types of Queue

Data Distribution

API Based Integration

Data Integration

Types of Data

Types of Data

System Data

- Data that must be present for the system to start up correctly

Static Configuration Data

- Configuration data that changes infrequently

Configuration Data

- Configuration data that is updated regularly

Transaction And Event Data

- Data generated by a running system

Internal and External Lookup Data

- Data that must be loaded on demand

Application Updates

- Data that is used to upgrade the application

System Data

There are certain sets of data that must exist for the Application to operate correctly

These "System Data" items are managed by the Application itself and are automatically loaded into the database as the application starts up

Examples of System Data include:

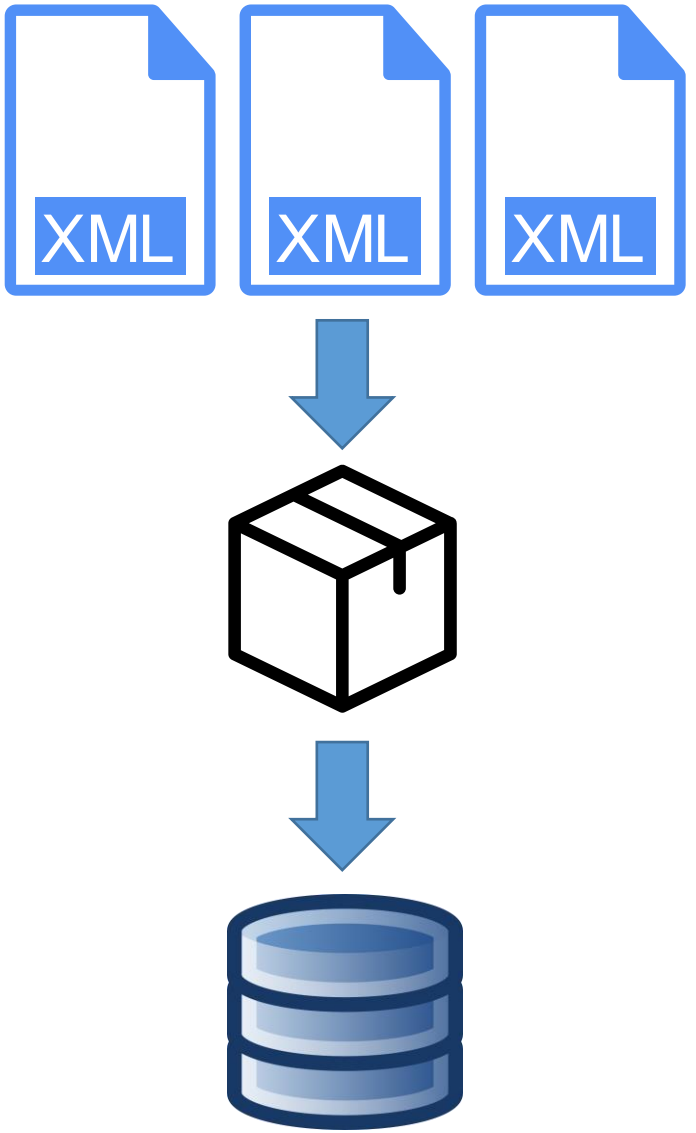
- Service Definitions
- Various Types, including Device Types, System Event Types, etc
- Available Reports

System Data (cont.)

System Data is configured by registering an XML document as a part of the application configuration

As the application starts, the Enactor Deployment Handlers will check if the system data is present in the Database and upload it if it is missing

System Data can be changed once it has been uploaded and it will not be overwritten, however if it is deleted it will be recreated at the next system start up



Static Configuration Data

There are generally items of configuration data that are static, or essentially static, for the life of the business.

This might include things like:

- Company
- Currencies and Languages
- Devices and Locations

Static Configuration Data will not typically need to be integrated with other systems as the cost of developing the integration exceeds to cost of manually maintaining this through the User Interface

Static Configuration Data (cont.)

This configuration can be maintained using a QA or Pre-Prod installation with the Enactor Estate Manager application

Once the data has been updated it is then generally sensible to export the data to a source control system so changes to it can be tracked

The data can then be imported into the Production system once it has been tested

Configuration Data

Configuration Data generally changes relatively frequently. It may include items such as:

- Product and Price data
- Promotions
- Inventory Levels

For this reason it is generally sensible to setup integration for these types of data so that you do not need to manually maintain it in multiple places

Enactor provides a robust Import/Export infrastructure that supports integration using XML documents sent via a Queueing system and/or Rest API

Transaction and Event Data

As the application is used you will generate Transactions and Events.

These must be frequently distributed to external systems for integration and auditing purposes

Transactions and Events are generally distributed using the Enactor Queuing system, this can deliver documents via other industry standard queueing technologies, or drop them into a folder for consumption

Internal and External Lookup Data

Certain types of data must always be managed in an "online" manner, the typical example is Customer Loyalty Points.

This data is typically held in one central place and then accessed via Web Services.

While Enactor supports hosting all the data needed for the application, you frequently want to be able to host data outside Enactor – for this reason Enactor can be easily configured to contact external services to lookup data, rather than relying on its internal services.

Application Updates

To perform an update we must be able to firstly notify an application that an update is ready for it. Once the application receives this notification it must then be able to source the data for the update from somewhere.

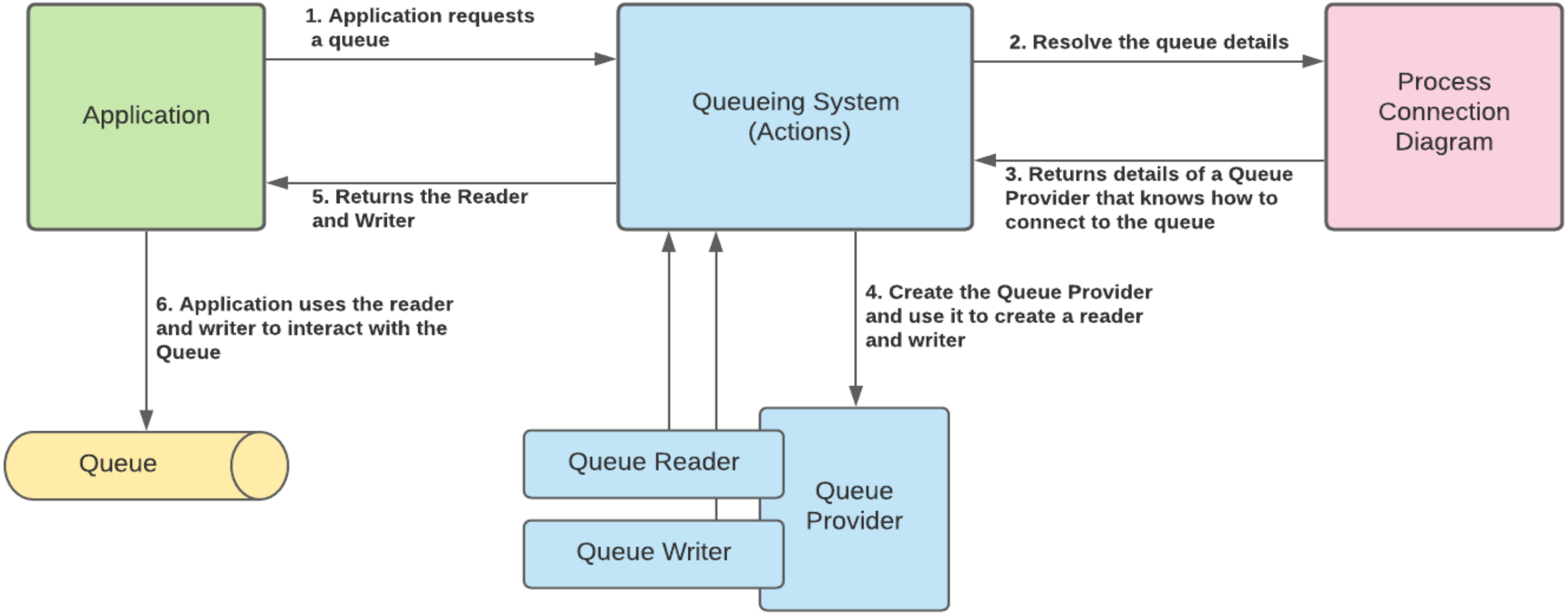
Enactor typically uses the Queueing System to deliver the notification that an upgrade is ready.

Then when the application is ready to apply the update, it will download the data for the update from a file server. This might be the Enactor Estate Manager, or could also be one of a number of supported third party file servers, for example:

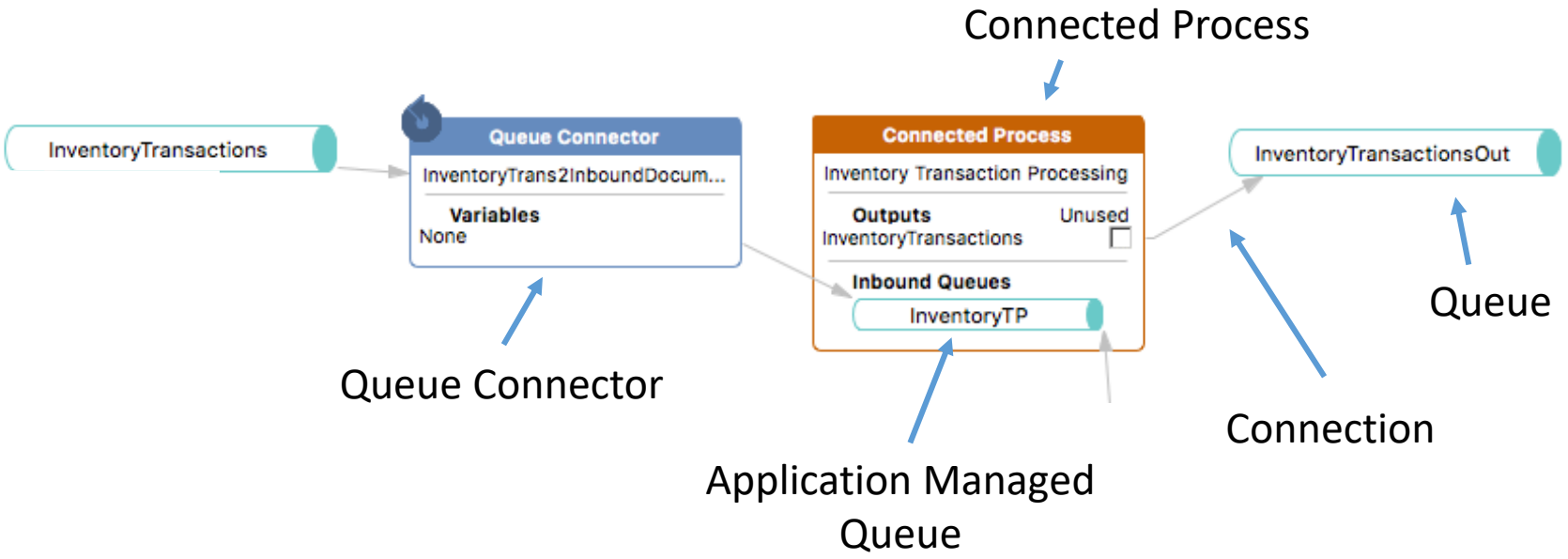
- Azure Blob Storage
- Amazon S3

Queue System Architecture

High Level Architecture



Terminology



Connections will be associated with a "Queue Provider"

The Queue Provider determines "how" to read and write from the associated queue

Queue Providers

Queue Providers are factory classes that provide Queue Readers and Queue Writers for a given type of Queue

The standard Queue Providers are:

- Database Queue Provider
- Http Queue Provider
- File System Queue Provider
- JMS Queue Provider

Queue Connectors

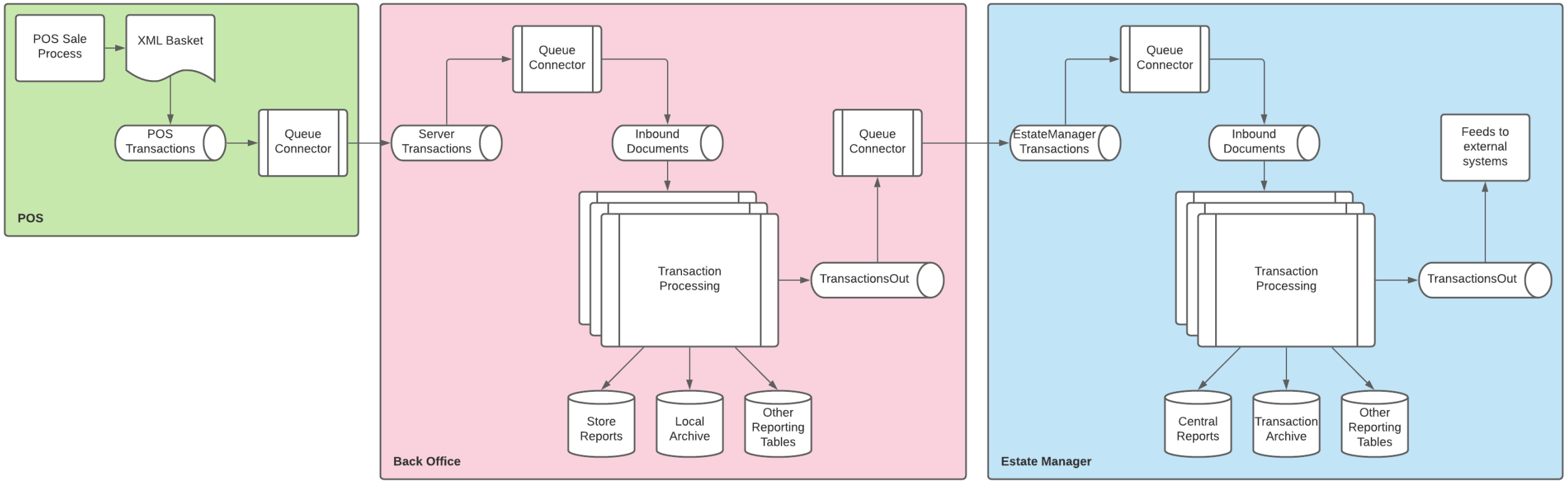
Queue Connectors are responsible for copying messages from one queue to another

The may transform a message, if so configured, but otherwise do not process or understand the contents of the messages

Queue Connectors will attempt to copy a message from the configured "source" queue to the "target" queue at regular intervals until the copy succeeds

- They retry "forever"
- They will never "skip" a message

Example – Transaction Flow



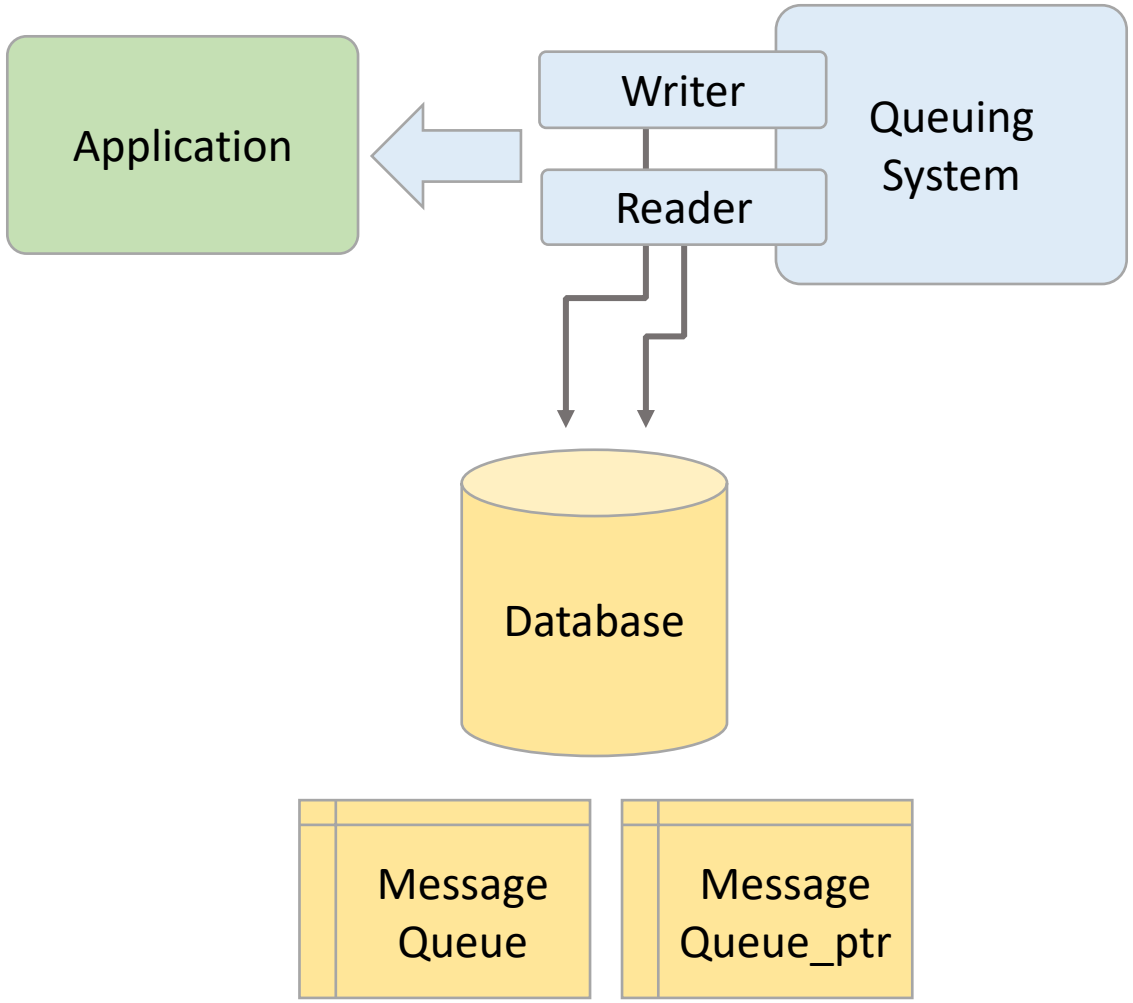
Types of Queue

Database Queues

Database Queues are used to provide “guaranteed delivery”

They allow the application to be certain it is safe to continue with an operation as the message has been committed to a database

In general we only use Database Queues against the local database, however it could be used to communicate to a remote database

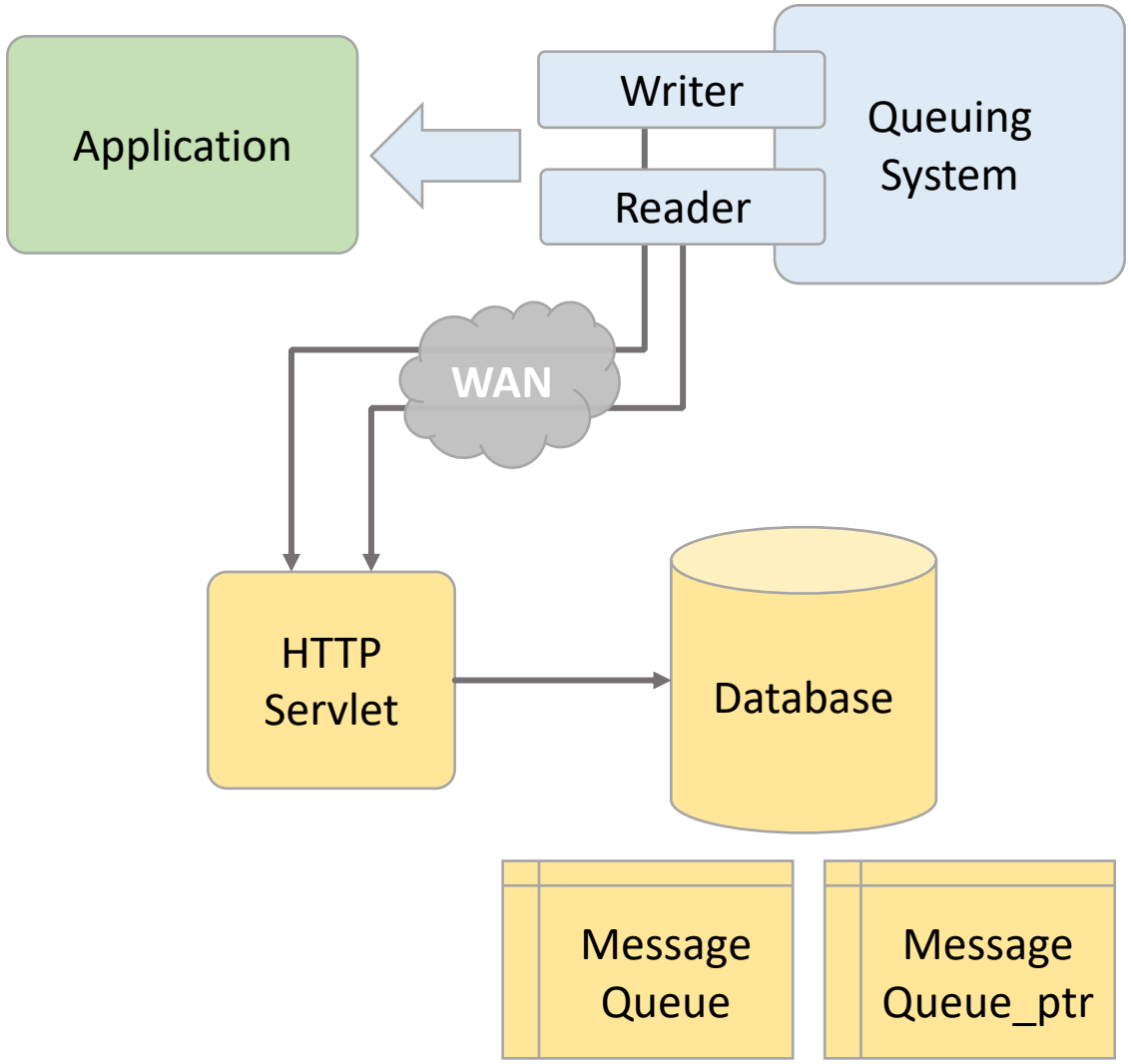


HTTP Queues

HTTP Queues are the out-of-the-box approach for transferring messages from one application to another

HTTP Queues are backed by the same database tables used by the Database Queues

Therefore these also provide “guaranteed delivery”

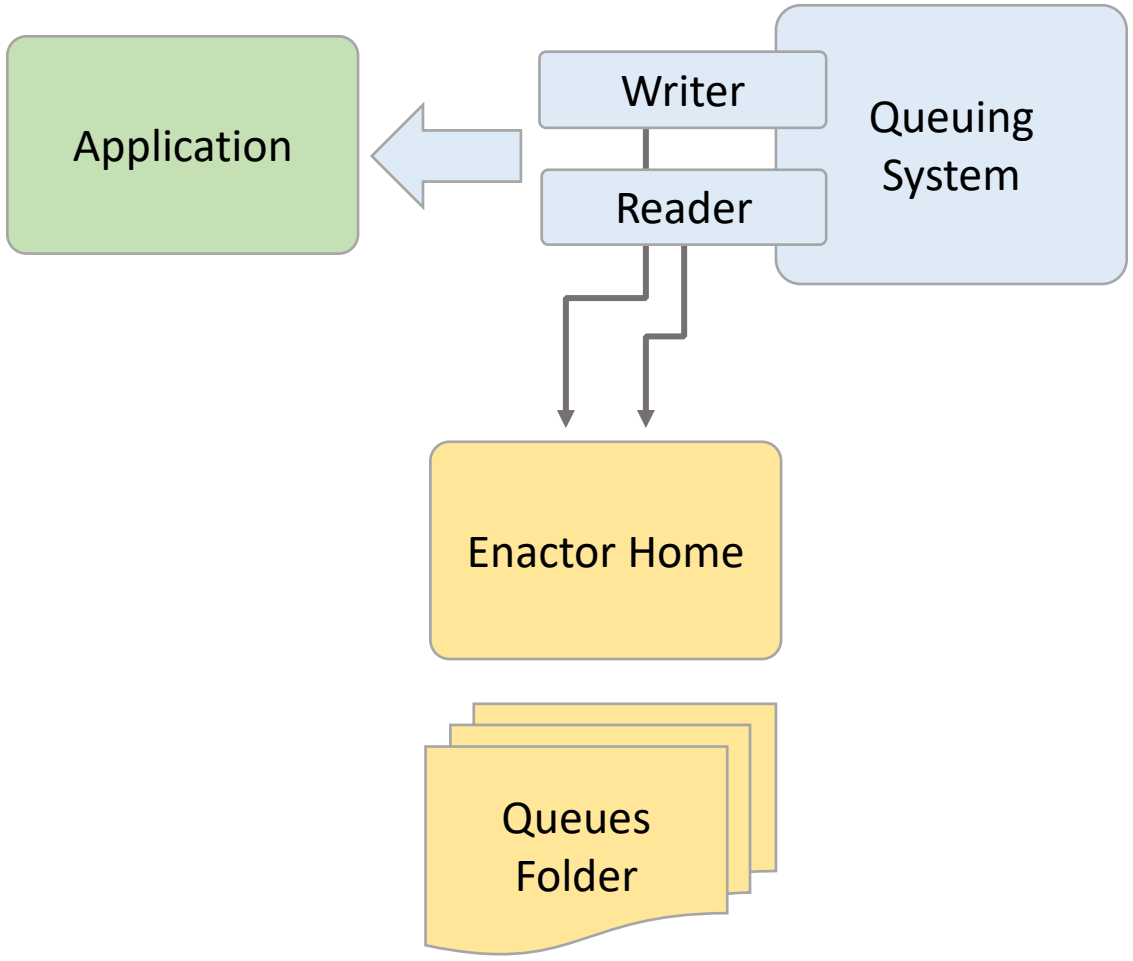


File Queues

File Queues are well suited for transferring bulk data

Messages can be single documents, or can be zip files

Messages are stored in the local file system

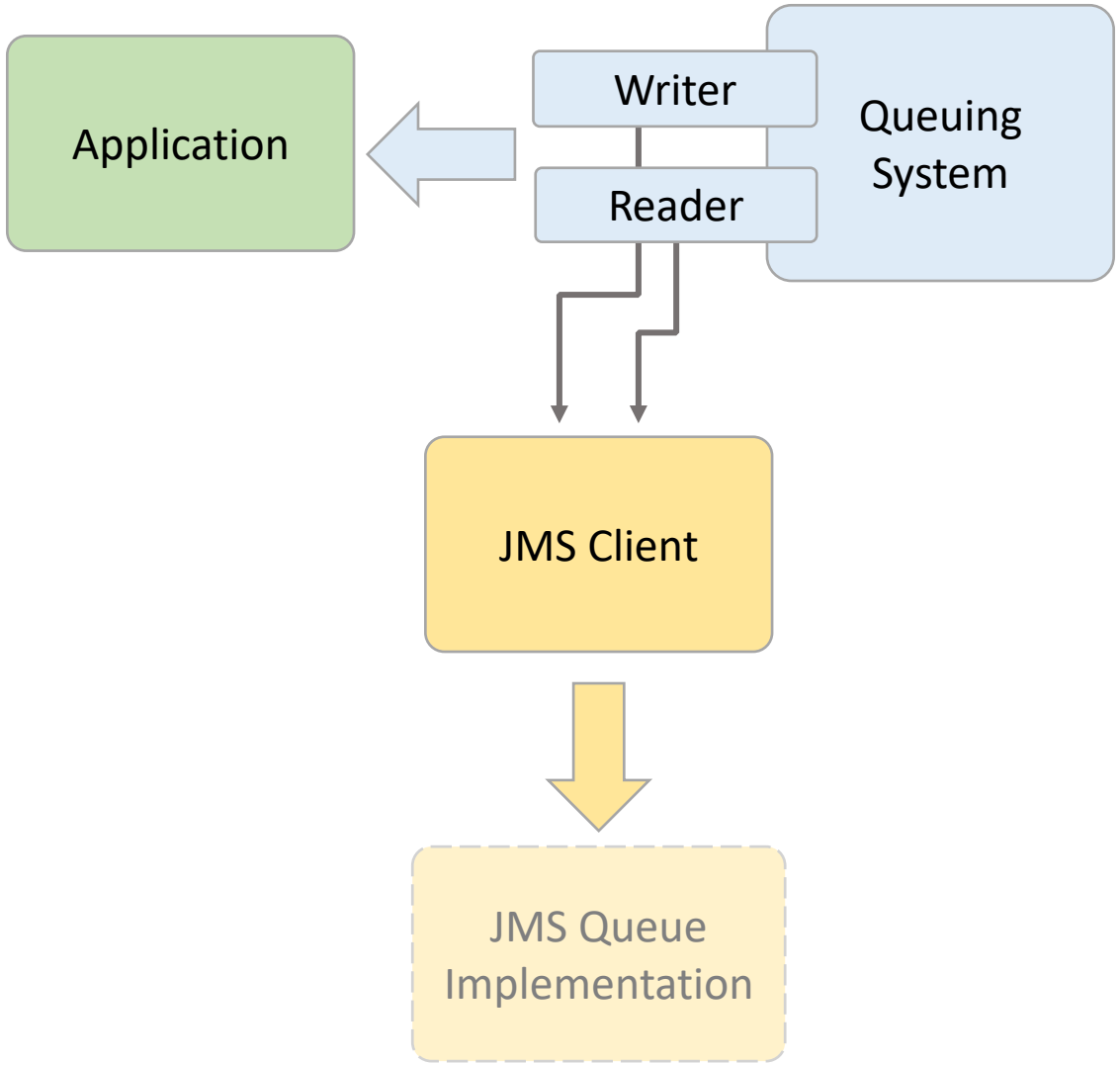


JMS Queues

JMS Queues provide an easy way to integrate with many third party queue providers

As Enactor does not need to know how the queue is implemented, you can introduce new JMS Queues without making code changes

Enactor uses the “Pub-Sub” model when talking to a JMS Queue



Data Distribution

Data Broadcast

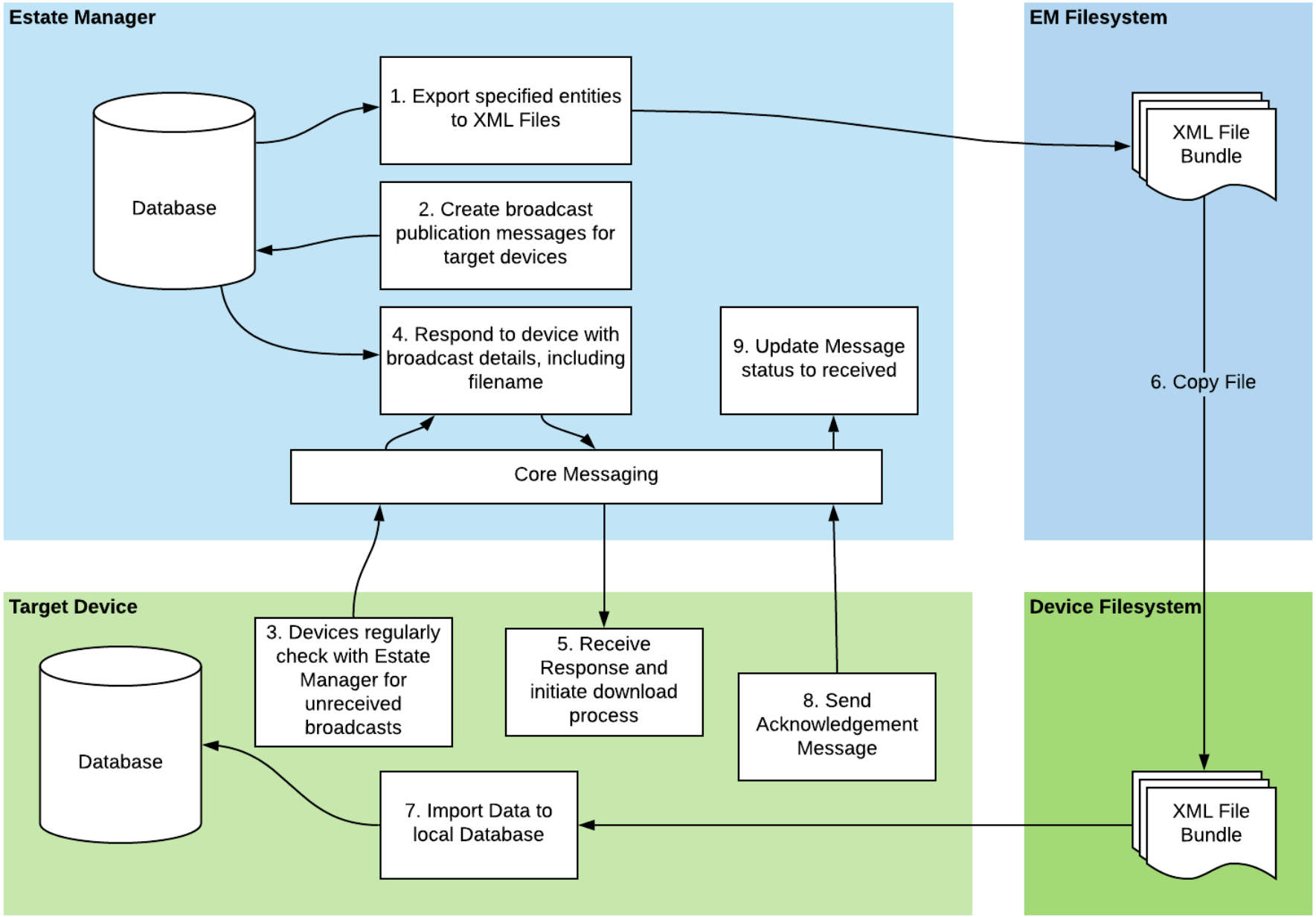
To update data in an application you must send a Data Broadcast

- This sends a notification that a data update is available and includes the location where the update file can be downloaded from

- When the application receives the notification it will connect to the specified end point, download the file and then apply it to its local database

- You can configure update windows where the application is allowed to apply downloaded files if required

Data Broadcasting



File Storage

Enactor supports several different types of file storage:

- Enactor Estate Manager File System
  The “out-of-the-box” basic support for file storage

- Azure Blob Storage
  Store files in an Azure account

- Amazon S3
  Store files in an Amazon S3 account

File Download Proxy

It is often the case that there are many devices within the Estate that need to download the same file.

It is also frequently true that many devices within a single location/store need to download the same file.

If this is not properly catered for, it can lead to devices consuming all available network bandwidth as they try to download updates simultaneously.

To solve this problem Enactor supports configuration of a File Download Proxy

File Download Proxy (cont.)

The File Download Proxy sits between the device attempting to download a file and the endpoint configured in a download notification.

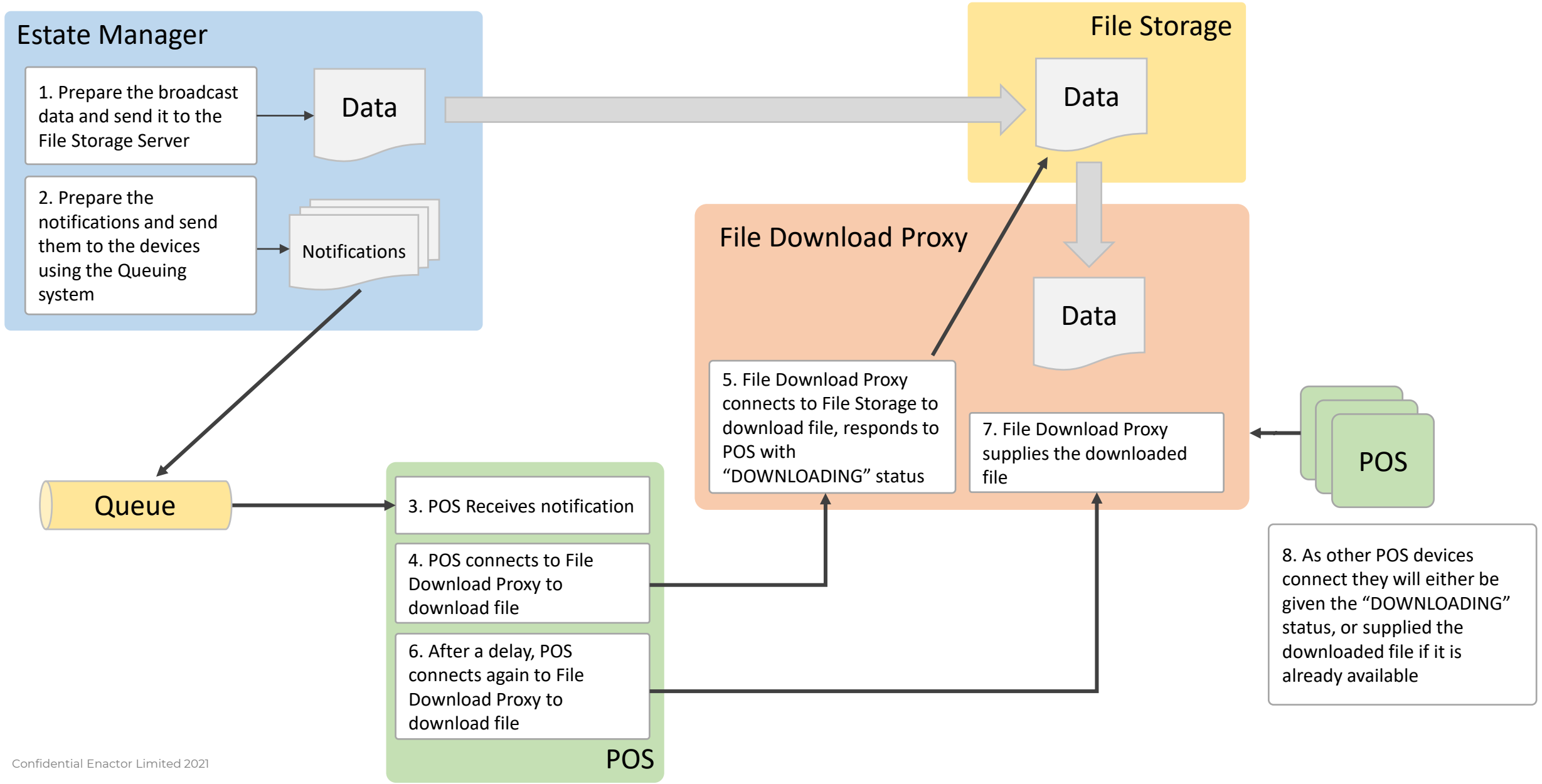
When the notification is received, the device will contact the Proxy and ask it to provide the file specified in the notification on its behalf

The Proxy can then download the file once and serve it to multiple clients

Clients do not hold a connection to the Proxy while the file is downloaded – they check on the progress of the download and if the file is not available yet the Proxy will respond appropriately

In a large estate it is recommended to deploy a Proxy at each Location

File Download Proxy (cont.)



API Based Integration

API Based Integration

In addition to the Queue based mechanisms described previously, Enactor also support API based integration using our Rest API

This API provides services for importing data at the Estate Manager

- We typically do not use APIs to update data at the Store.
- Instead APIs should be used to update data centrally which can then be broadcast out to the Service Devices in the Estate

- Alternatively, the Store Devices can subscribe to an externally managed queue to receive the data directly

Services for Importing Data

To import data into Enactor the general ‘Import’ service should be used:

```
POST {BASE URL}/rest/configuration/import/{filename}
```

This accepts XML formatted data in the same format as the standard Estate Manager Import application.

The import can be done asynchronously if required, in which case the response will describe the `jobReference` assigned to import the data – this can then be reviewed in the Estate Manager

Data Integration

Data Integration

Enactor supports many options for integration with external systems based on the differing requirements of each customer.

For Schwarz, Enactor are recommending the following approach:

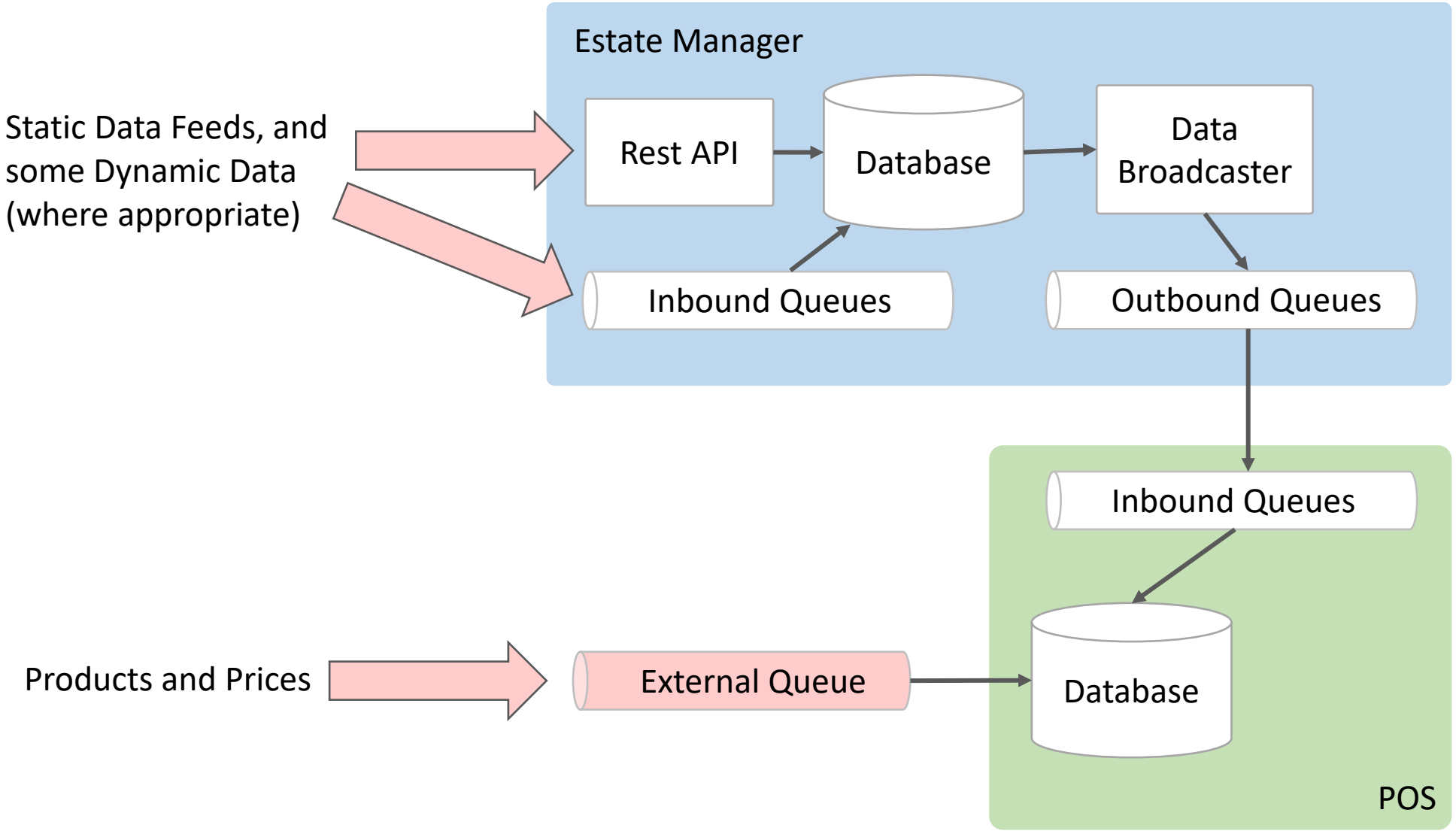
1. Data feeds for the majority of entities into Enactor arrive at the Estate Manager which are then broadcast down to the Store Devices. This can either be delivered using a Queue or via the Rest API

2. Products and Prices are the exception – these are instead sent directly to the relevant Store Devices using an external Queueing System

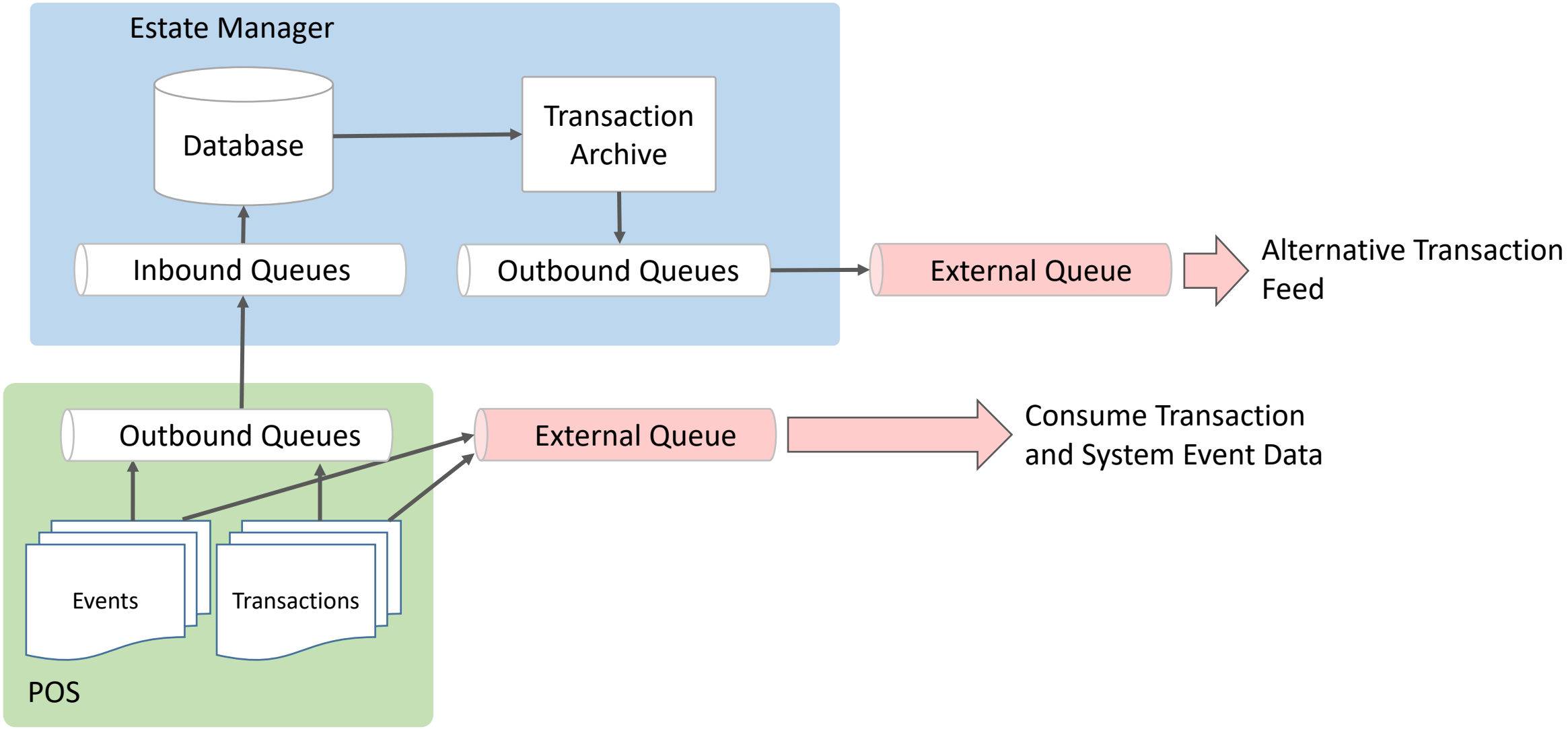
3. Transaction Data out of the Enactor application should be received from Queues written to by the POS applications

4. If necessary the Transaction Data can be reconciled with that emitted by the Estate Manager once Enactor has completed Transaction Processing.

Data Integration – Inbound



Data Integration – Outbound



Data Integration – Outbound (cont.)