Architecture: Scalable e-commerce workloads using microservices

This is the sample showcase of how we architect scalable solution for your business using cloud native stack

Retail commerce requirements and microservices

Retail commerce workloads require a number of cloud-native features in order to meet demand from an ever-growing number of consumer devices and platforms:

- Typically, these deployments must be multi-region to serve a global customer base.
- They must support some degree of autoscaling or scheduled scaling, scaling up to meet peak demand during busy seasons, and scaling down to reduce infrastructure costs when demand is lower.
- Retail commerce deployments must be able to deliver features and functionality to customers quickly and efficiently to meet changing market demands.
- Retail commerce deployments should also take advantage of managed infrastructure to allow developers to focus on customer-facing functionality.
- Finally, these deployments must be centrally secured and managed.

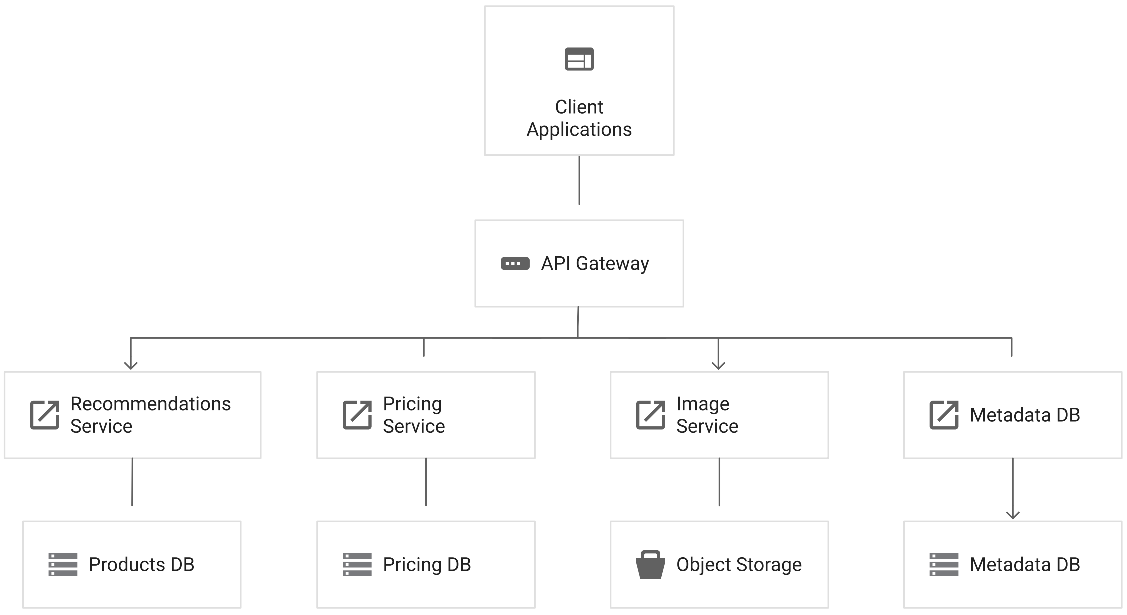
Microservices are a good fit for all of these requirements. Individual microservices can be deployed and scaled independently of one another, which lets you rapidly deliver new features and functionality. Services can be small, modular, loosely coupled, and organized around your specific business capabilities and needs. Microservices can leverage service discovery and use simple mechanisms (such as HTTP) for easy connectivity from a wide variety of devices.

Backend architecture

For retail commerce workloads, you organize microservices into the discrete functions that are needed to build the customer-facing user experience. For example, you might have a product metadata service that retrieves (and optionally, caches) metadata for a particular product. Or you might have a product pricing service that retrieves the price of a product for a given customer.

Your microservices are exposed to clients via REST APIs, and your client applications communicate with the REST APIs through an API gateway.

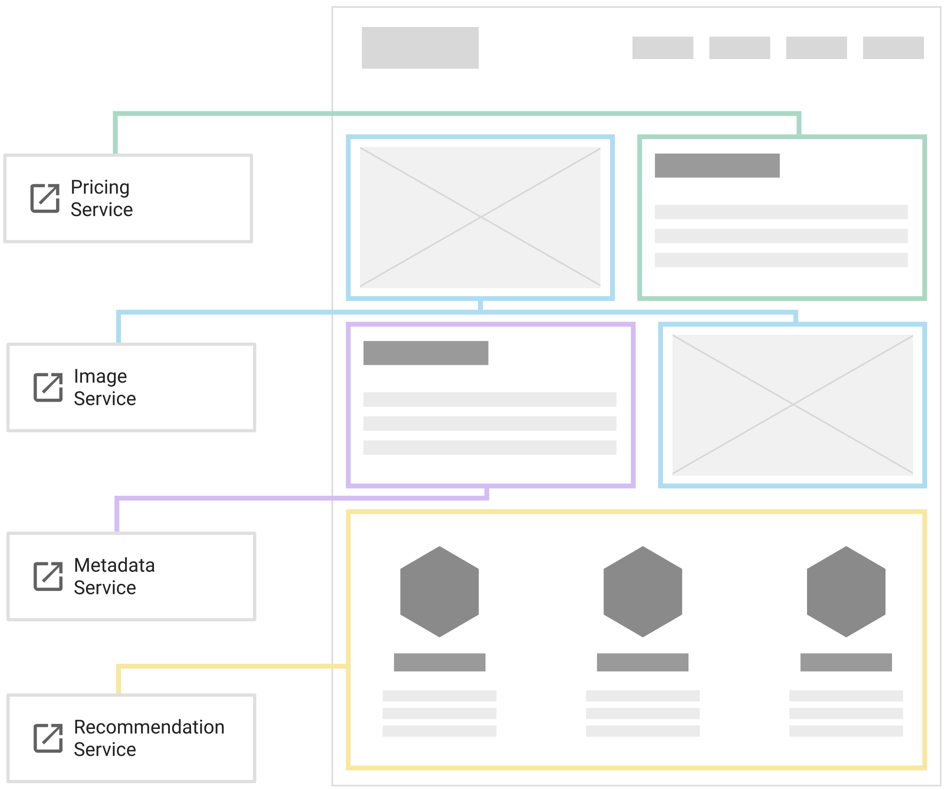
The following diagram shows an example commerce-oriented backend microservices architecture.



Frontend architecture

The customer-facing user experience in your retail commerce workloads typically includes responsive web applications (often delivered as Progressive Web Apps) and optionally as native mobile applications. In combination with the backend architecture shown previously, you build your applications by assembling multiple frontend components that correspond to and communicate with backend APIs and services.

The following diagram shows an example commerce-oriented web application frontend.

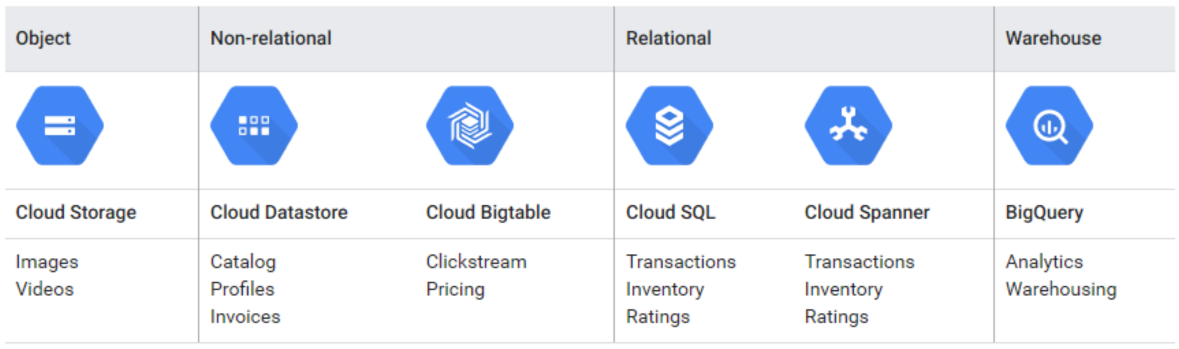


Data storage

Retail commerce workloads must persist multiple categories of data. Categories include:

- **Product catalog**—product attributes such as name, description, colour, and size.
- **Shopper profiles**—customer data such as name, age, preferences, and billing and shipping addresses.
- **Shopper transactions**—information about customer purchases such as items purchased and date of purchase.
- **Clickstream data**—information that traces a shopper's path through the website.
- **Product images and videos**—media related to a particular product, including first-party content and customer-supplied content.
- **Ratings and reviews**—opinions and feedback from customers for a product.
- **Product inventory**—data about whether an item is in stock and expected arrival of new stock.

Each category of data can be mapped to a Google Cloud storage mechanism, as shown in the following table.



Object	Non-relational		Relational		Warehouse
Cloud Storage	Cloud Datastore	Cloud Bigtable	Cloud SQL	Cloud Spanner	BigQuery
Images Videos	Catalog Profiles Invoices	Clickstream Pricing	Transactions Inventory Ratings	Transactions Inventory Ratings	Analytics Warehousing

Product catalog

In product catalogs, products have a set of attributes—name, description, and so on. But as your product catalog diversity grows, the number of distinct attributes grows as well. Each new category of products has its own set of attributes that can be used to search or filter on, such as item sizes and colors, or item type and model.

For product catalogs, the most appropriate storage option is therefore a NoSQL document-oriented database, which has a flexible schema and can store per-category or per-object attributes. Datastore is a fully-managed NoSQL document-oriented database and provides support for this use case. In Datastore, you store objects as entities, and each entity supports nested key-value pairs, similar to the structure of JSON. Datastore is available within multiple Google Cloud regions and runs as an always-on service.

Product media

Every product in a product catalog can have first-party images or videos, and might also have customer-supplied images or videos. You can store these sorts of assets in a scalable object storage system, capable of serving those assets directly to web applications or mobile applications. Cloud Storage is a managed object storage service that can serve data across multiple regions. Cloud Storage offers different tiers of data access and availability depending on your needs. For high performance, Cloud CDN leverages Google's globally distributed edge locations to accelerate delivery for content served from Cloud Storage. This ensures that your static assets are located as close as possible to end users in order to minimize download latency.

Shopper profiles

Shopper profiles have a consistent set of attributes and are often multidimensional. For example, some of your customers might have multiple shipping addresses or multiple payment methods, each with their own billing address.

You can store shopper profiles in relational databases using multiple tables. However, you might also use NoSQL document-oriented databases to store customer profiles. This lets your shopper profiles be stored as single, rich objects that hold all of the data for a given customer. Datastore is a fully-managed NoSQL document-oriented database that can provide support for this use case.

Ratings and reviews

Product ratings and reviews left by customers consist of relatively simple data sets, and you can persist this information using different storage mechanisms. It's typical to use relational schemas containing fields such as product ID, customer ID, rating value, and review text. You can store this data using either Cloud SQL or Spanner. For most use cases Cloud SQL is the most appropriate system to store ratings and reviews data. If your applications require higher transactional throughput and horizontal scalability, Spanner is the right choice. For more information on which database service to use, see Choosing a storage option.

Transactions and invoices

As with ratings and reviews, you can persist shopper transactions and invoices or order details using different storage mechanisms. Transactions must be stored in database systems that support ACID semantics, specifically the ability to atomically commit writes. Datastore, Cloud SQL, and Spanner all support atomic operations. For most use cases, relational systems are a good choice for transactions, because the data is consistently structured from one write to the next. The choice of storage system largely depends on your comfort with either SQL or NoSQL systems, and the ability to customize applications to the chosen database.

Invoices can also be stored using either NoSQL or relational databases; the downstream use cases should drive which system you choose. In modern commerce workloads, NoSQL document-oriented databases such as Datastore are often used to persist invoices or order details, because the entire state of the invoice can be stored as a single rich object. For more traditional commerce workloads, Cloud SQL or Spanner may also be appropriate choices.

For more information on which database service to use for transactions and invoices, see Choosing a storage option.

If your environment is entirely cloud based, your transaction and invoice data lives entirely in the cloud infrastructure. On the other hand, if you work with a hybrid environment, you need to synchronize data between the cloud environment and your on-premises environment. In the hybrid scenario, the transaction and invoice data usually resides in the on-premises infrastructure. In that case, you can synchronize backend systems with the cloud data infrastructure using a combination of custom applications, Pub/Sub, or database replication.

Clickstream data

Data about customer traffic is often captured through analytics packages such as Google Analytics. However, you might want to gather this navigation data (clickstream data) in real time.

There are many methods of capturing clickstream data; one way is to use Serverless pixel tracking using Google Cloud. The datasets produced for clickstream tracking tend to be very large and are often used as sources for machine learning or predictive analytics. This type of data is usually stored in NoSQL wide-column systems such as Bigtable. Big table supports large-scale datasets (up to hundreds of petabytes), and provides low latency and high throughput, which is helpful for this use case.

Product inventory

Data about whether a product is available is critically important to the overall customer experience. Inventory data often consists of datasets containing product SKUs, current inventory, and expected date of additional inventory. But given the way this data is often used in applications, the storage mechanism must support transactions and atomic operations to accurately reflect inventory levels. Datastore, Cloud SQL, and Spanner all support atomic operations. For most use cases, relational systems are a good choice for inventory data because the data is consistently structured. For more information on which database service to use, see Choosing a storage option.

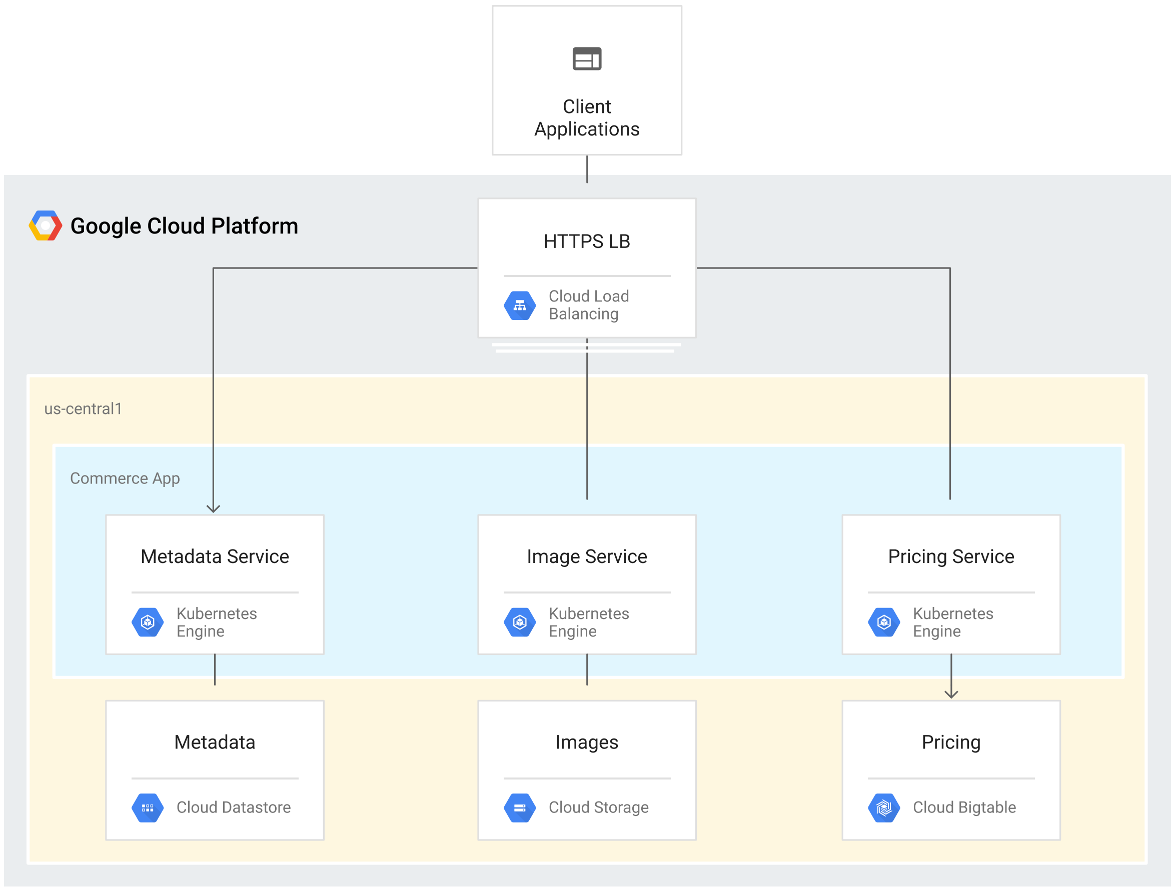
As with transaction data, if your environment is entirely cloud-based, inventory data lives in the cloud data infrastructure. If you work with a hybrid environment, you need to synchronize data across the cloud environment and your on-premises environment. In the hybrid scenario, inventory data usually resides in the on-premises infrastructure. In that case, you can synchronize backend systems with the cloud data infrastructure using a combination of custom applications, Pub/Sub, or database replication.

Deployment architectures

When you use Google Cloud, you typically deploy microservices using either App Engine flexible environment or Google Kubernetes Engine. App Engine flexible environment is a fully-managed Platform as a Service (PaaS) that provides auto-scaling, load balancing, and support for common languages and frameworks. GKE is built on top of Kubernetes, an open source container-orchestration and cluster-management mechanism. The choice of platform for your deployment depends on the level of flexibility you need and how complex your application infrastructure is.

Using GKE

The following diagram shows an example deployment with microservices using GKE.



GKE supports autoscaling Pods, depending on CPU usage, with the horizontal Pod autoscaler. In addition, GKE clusters also support autoscaling using the GKE cluster autoscaler, which automatically resizes clusters, based on saturated or underutilized resources.

GKE clusters are regional resources, and for deployments that require high availability, you should create deployments across multiple zones. For more information, see Overview of multi-zonal GKE clusters.

For deployments that must serve a global customer base, deploy multiple GKE clusters within a single project, one per region. Data storage for each microservice is provisioned and operated out of the same project as the GKE clusters.