Multi Tenancy to Completion

May 31, 2017

1 The Problem

FoxCommerce platform has been moving towards multi-tenancy and is almost there. There are a couple of remaining issues that need to be addressed for the system to be completely multi tenant.

2 What We Have

2.1 Organizations

We have a model of organizations which have roles and permissions and users. Organizations are associated with a specific scope. Scopes provide a way to organize data hierarchically and limit access.

2.1.1 Capabilities and Constraints

Organizations can...

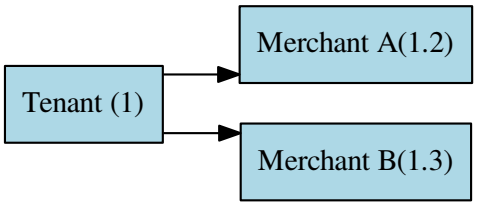
1. Control roles and permissions.
2. Control how a scope is accessed.
3. Have sub organizations that belong to subscopes.
4. Control subscopes.

Organizations cannot...

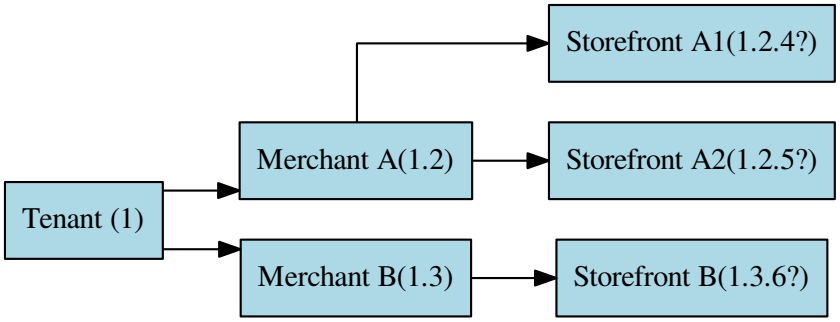
1. Cross sibling scopes.
2. Have unscoped data.
3. Users cannot log into multiple scopes at same time.

2.2 Scopes

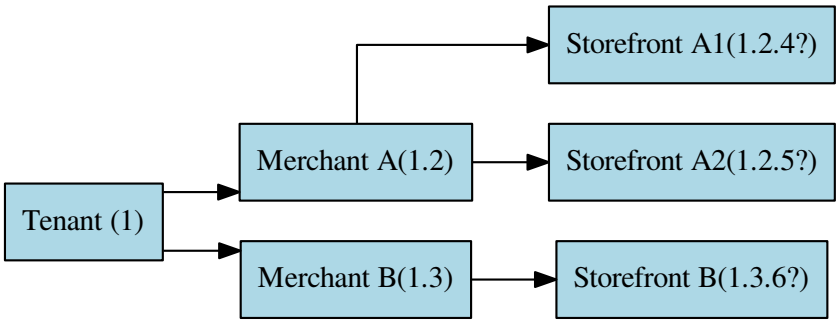
Almost all tables in the system have a scope column. Scopes are hierarchical organization of data like a filesystem tree. Users with access to a scope do not have access to the parent scope.



Each merchant may have one or more storefronts. The question of whether the data of those storefronts is scoped depends on the use cases we want to enable. Is a different organization managing the other store fronts? Then we probably want this



Is the same organization managing various storefronts? Then we want this.



Notice that the scope of the storefronts is the same. Regardless if we have one organization or another we need a different organizing structure for storefront data that is separate from scopes. We want a model of channels.

2.2.1 Capabilities and Constraints

Scopes can...

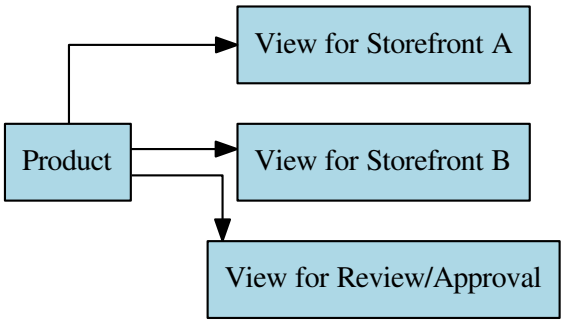
1. Group data like a directory in a filesystem.
2. Control access to data via roles/permissions that are in that scope.

Scopes cannot...

1. Share data with sibling scopes.
2. Provide semantic relationships between data in a scope.
3. Provide semantic relationships between data in different scopes.

2.3 Views (formally Context)

All merchandising information can have several views. Views provide a way to change the information of a product, sku, discount, or other merchandising data for a specific purpose. For example, each storefront could possibly have a different view of a product. A review/approval flow may have it's own view.



2.3.1 Capabilities and Constraints

Views can...

- Control which versions of data in the object store are displayed.
- Act as a git branch on the merchandising data.
- Commits provide branching history between views.

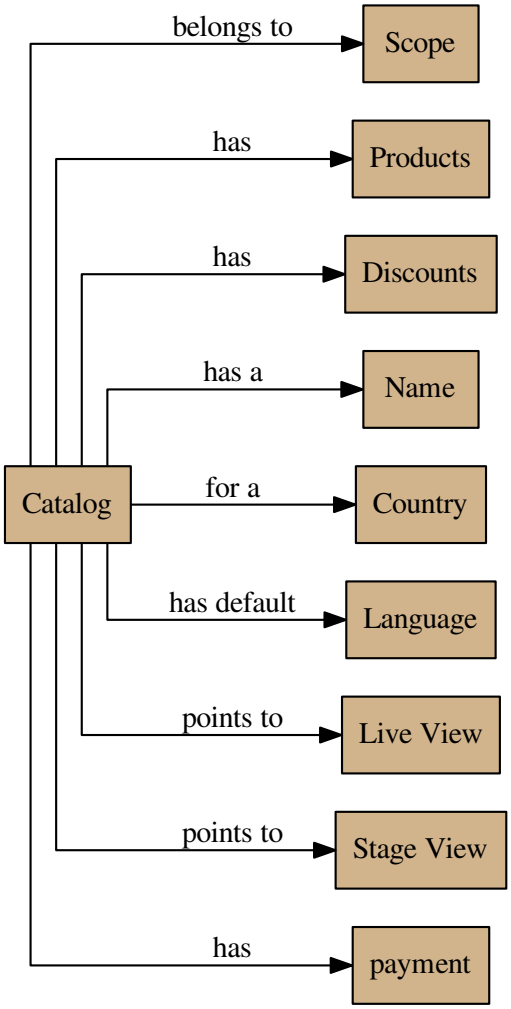
Views cannot...

- Control access to data.
- Shared between sibling scopes.
- Cannot control versions of parts of objects.
- Cannot describe semantic relationships between views.

3 What We Need

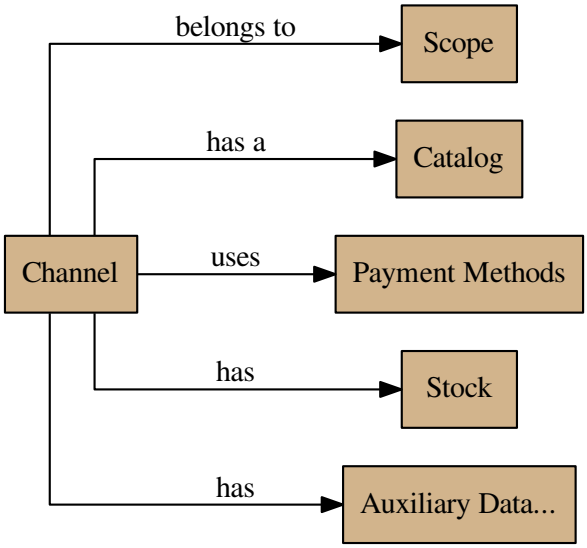
3.1 Catalogs

Catalogs are collections of products you want to sell.



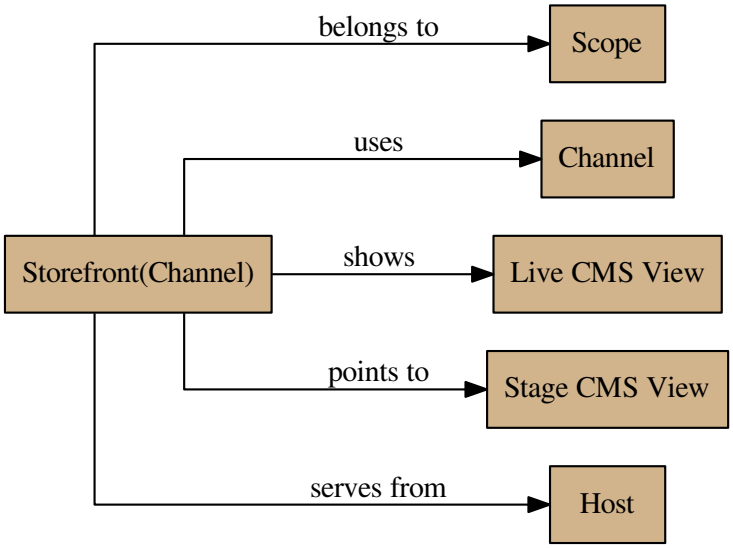
3.2 Channels

Channels should be comprised of three key components



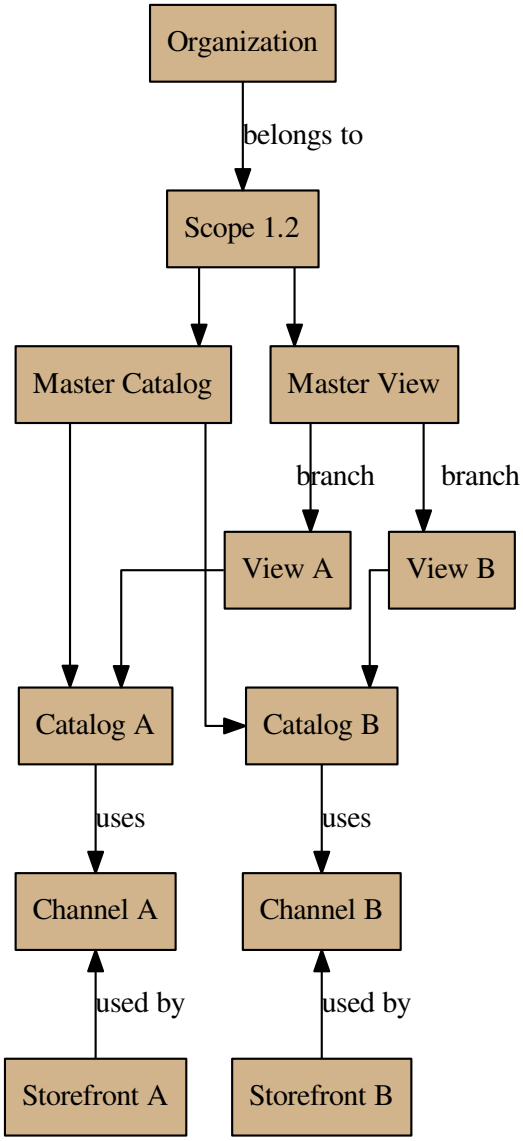
3.3 Storefronts

A storefront is a website that can sell products from a catalog. A storefront uses a channel in addition to data from the CMS.


belongs to
Scope
uses
Channel
Storefront(Channel)
shows
Live CMS View
points to
Stage CMS View
serves from
Host

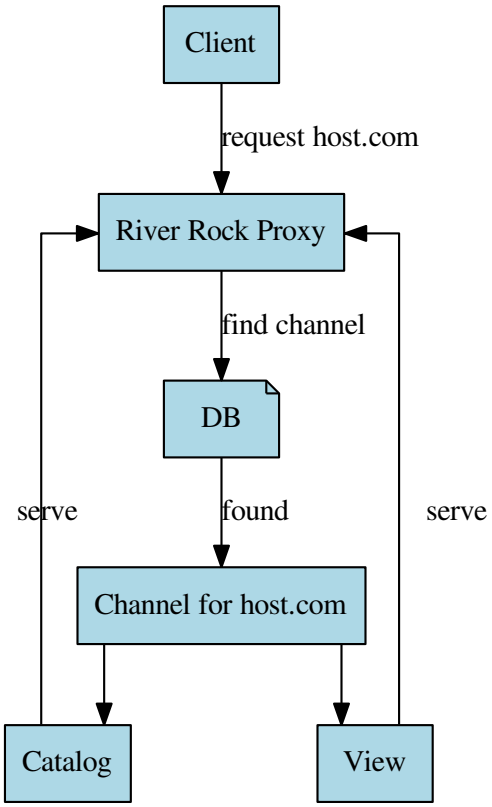
3.4 Back to Organizations

Once we have these new models we can assign them to organizations



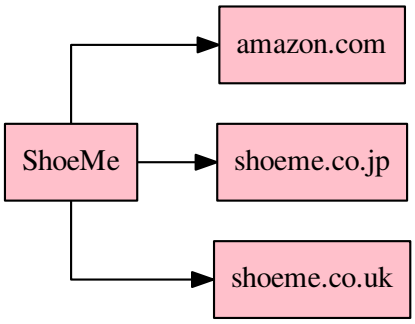
3.5 Public Access

Our current public endpoints for searching the catalog, registering users, and viewing/purchasing products are not multi-tenant aware at the moment. We need to modify them to understand which channel they are serving.

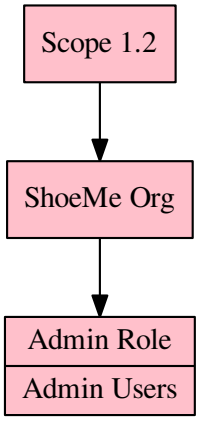


3.6 Example Story

We acquired a customer which sells shoes called ShoeMe that wants to operate two global sites and sell products on amazon. They want to sell on two domains, shoeme.co.uk and shoeme.co.jp and amazon.com.

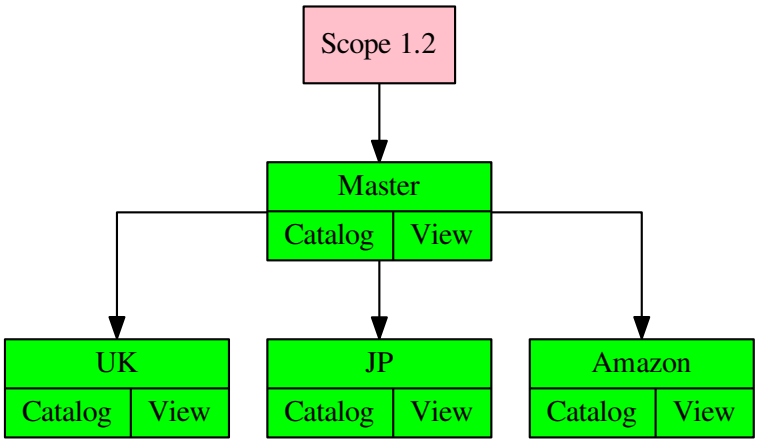


We create a new *Organization* called ShoeMe and *Scope* with id "1.2" . Within that *Organization* we create a *role* called "admin" and a new *user* assigned to that role.

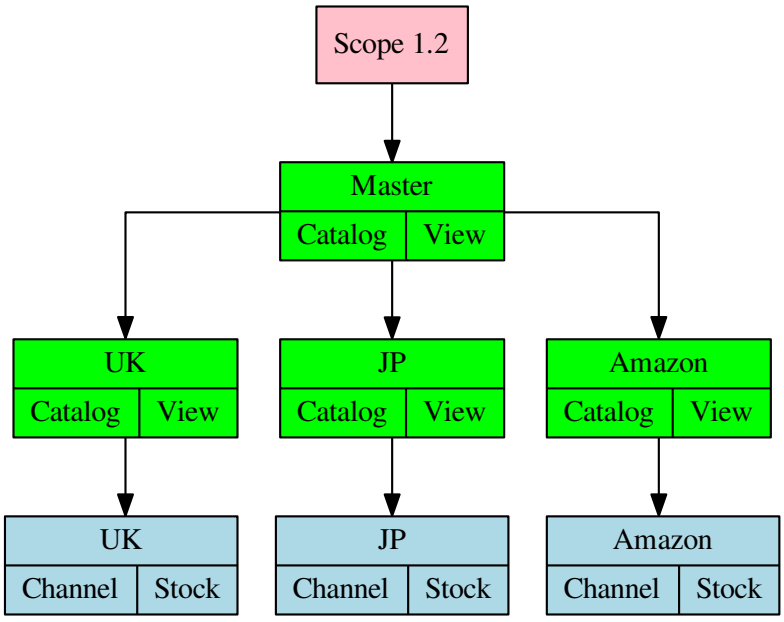


The ShoeMe representative follows the password creation flow for their new account and gets access to ashes.

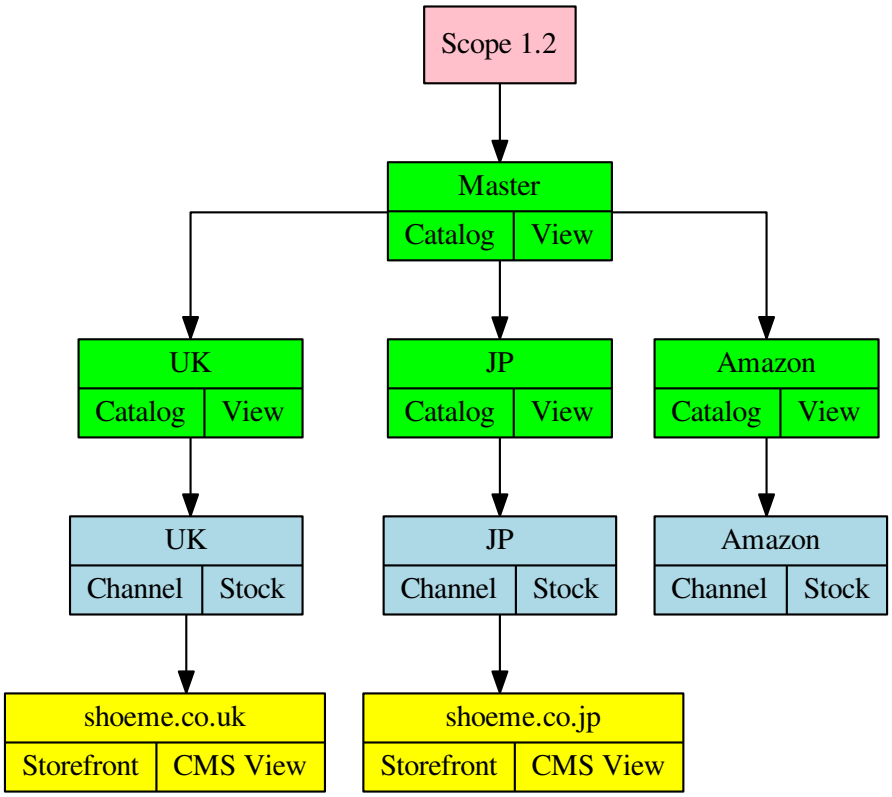
We create for the customer four catalogs called "master", "uk", "jp", and "amazon". Behind the scenes with creates four views "master", "uk", "jp", and "amazon". They then import there products into the master catalog. They then use ashes to selectively assign products into the other three catalogs. This forks the products into the three other views.



We create three channels for the customer called “uk”. “jp”. and “amazon” and assign each appropriate catalog to the channels.



We then create two storefronts, “shoeme.co.uk” and “shoeme.co.jp”. Behind the scenes these two storefronts will create two views which will be used by the CMS to show various content. Just like with catalogs they can first build one storefront and then copy over the data into the other.



Our API will understand how to serve traffic depending on the host of each storefront by selecting the appropriate CMS views and catalog assigned to the storefront.

Behind the scenes will can also provide sister views to each view for staging changing which await approval.

Each catalog may also be assigned the same or different stock item location and pricing information. Each catalog can therefore share or maintain separate stock for the products in the catalog.