Hyrax's access controls are a multi-layered affair. There are four primary permission checks:

- Asking about user's "roles" (e.g. `current_user.admin?`)
  - Often used for expediency as a short-circuit to a larger context
  - Leverages RoleMapper behavior from Hydra::AccessControls (see `def admin?`) - a rudimentary adapter for application-wide role definition (e.g. Admin, Cataloger) but does not provide contextual roles (e.g. editor of a collection)
  - This obfuscates why the role allows (or disallows) taking the given action on the given object
- Checking if user "can" do something (e.g. if `can? :edit, document`)
  - Much of the permissions are done via these checks
  - Leverages the Ability class, as provided by CanCanCan gem, for developer customization
  - Integrates with SOLR searches, originally envisioned as the central place to ask if someone can take the given action on the given object
  - Answers can someone do this but not whom all can do it
- Workflow checks, can someone take an action in the workflow (e.g. `if Hyrax::Workflow::PermissionQuery.authorized_for_processing?`)
  - Used to see if someone can take an action in the given state of the object
  - Implemented for customizable workflows and state-based permissions
  - Does not have a UI for maintaining workflows
  - Can answer "whom all can take the given action on the given item"
  - Can answer "given the item and given user what all actions can they take"
  - At present only part of Hyrax
- As part of SOLR searches (e.g. `Hyrax::My::SearchBuilder` via Blacklight)
  - Used for filtering search results
  - Used as one of the underlying permission checks for the CanCanCan gem
  - Requires knowledge of what indexed fields have permission related information

The current state of permissions appears to be an organic evolution of immediate community needs with minimal guidance provided for developers to know how to consistently and best implement permissions in Hyrax as well as applications that mount Hyrax.

Hyrax implements a rudimentary "site-wide" role mapper (as per Hydra::AccessControls) using YML file configurations. As written, modifying any roles requires a re-deploy and the "site-wide" roles do not integrate with workflow roles.

The implementation of these permission checks is a series of module includes, method overrides, local changes within an engine, and generating a file for application implementers to also override/modify/update.

The system ultimately lacks a cohesive strategy and end point to answer the following (especially when we consider state based permissioning):

- Given who I am and given an action and given a thing can I take the given action on the given thing?
- Given who I am and what all things can I act on?
- Given who I am and given an object what all can I do to that object?
- Given a thing and given an action whom all can take the given action?
- Given a thing what are all of the valid system actions on that given thing?