



# ALM and Anaplan App Management

Four Corners Workshop  
September 14, 2021



## Day in the Life

- Why ALM?
- ALM Basics
- Approach & Setup
- ALM in Action
- Anaplan App Management

# Why use Anaplan's ALM?

## Governance

- Enhances traceability back to individuals, project requirements, bug IDs, etc.
- Separation of duties
- Data privacy

## Development

- Methodical build, test & deployment of changes
- Flexible with planned updates or emergency fixes

## Operations

- Minimize business disruptions
- Scalable with organization

# ALM Basics

# ALM Basics

- Create source/target relationships based on compatibility such that structural changes can be synced between models to manage development cycle between Dev/Test/Prod while minimizing business disruption.

Model Mode	Description	Used For...
Standard	The default, standard mode provides full access to model data, including <b>structural information</b> . Depending on user access and administrative rights, all actions are permitted in models in standard mode. <i>Standard mode can also be online or offline. Standard mode provides all the features and functionality you're used to. Model building in standard mode is identical to model building in versions of Anaplan before ALM features were introduced.</i>	Development models
Deployed	Deployed mode blocks any modifications from being made to a model's structural information. The model can be online or offline. It's not possible to add revision tags in deployed mode. End users (or testers) can change a model's production data by <b>working with dashboards, entering data into modules</b> , managing users, and other similar activities. <i>Workspace administrators can only restore a model back to a restore point ID that existed after the model was last synchronized. Enabling deployed mode doesn't move a model to a different workspace or environment. Do not disable deployed mode in production models or sync target models. If structural changes are made after disabling deployed mode, the model will become incompatible for synchronization with previously compatible source models. In some circumstances, you can recover a sync incompatible model by reverting to the most recent revision tag.</i>	Production models Test models Preventing accidental changes while a model is in production use
Locked	Locking a model makes it read-only for all users, including workspace administrators.	Making a model temporarily read-only
Archived	Archiving a model enables you to store the model in Anaplan without it contributing to the workspace allowance. You can restore an archived model, with all its data, into the same workspace at any time ( <b>provided the workspace has enough space</b> ).	Maintenance Reducing the total size of a model's parent workspace

# Production vs Structural

- **Production data** is operational data that changes during business operations
- **Structural information** consists of a model's configuration setting and lists and cannot be edited when model is in deployed mode.
- The exceptions that are treated as Production data are listed below:
  - Lists
    - Must be marked as "Production Data"
    - Anaplan enforces [Formula Reference Protection](#) – cannot make direct formula reference to item in a Production List
  - Imports & Import Data Sources
    - A single "Master" model with multiple spoke models of the same structure might have difference data sources
    - E.g. T&Q models split by Region
  - Users
- Items to Remember
  - Switchover Period is **Structural** – it is potentially destructive for forecast data
  - Native Versions are **Structural**

# Approach & Setup

## Questions to Ponder

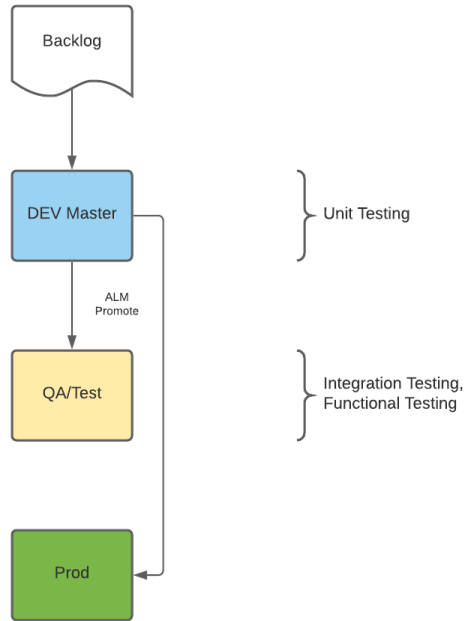
- What ALM structure will work best?
- What naming conventions will your team enforce?
  - “DEV Model Name” vs “Model Name DEV”
    - Sorts are different
    - Model Name you see as a model builder – the whole name is not always visible
  - Revision Tags
    - E.g. Major.Minor.HistoryID
- What is the right cadence of ALM pushes?
  - Different throughout each project phase
- What types of users need access in each environment?
- What is release strategy after go-live?

Understanding & estimating **model size** across Dev/Test/Prod models is key factor in thinking through ALM approach



# Single Threaded Development

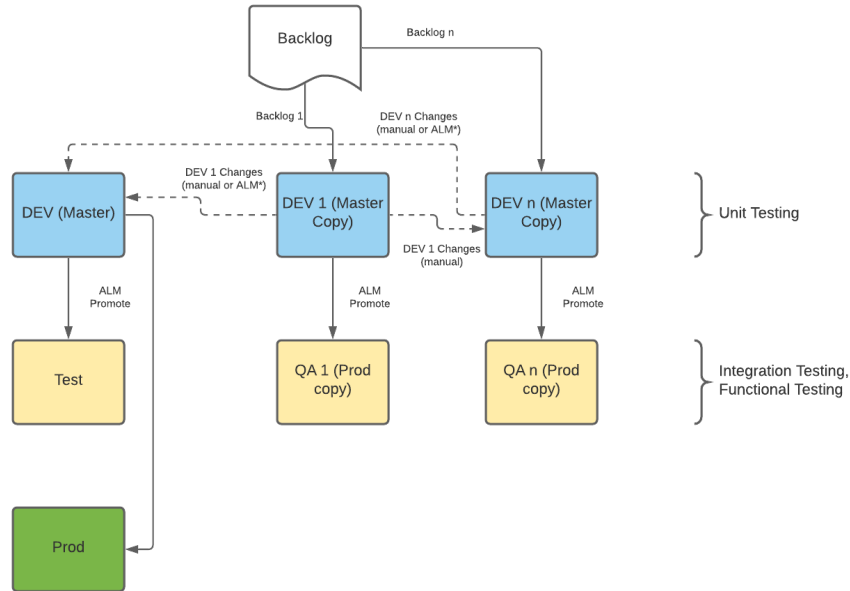
This is the standard ALM strategy and works well when the work is serialized.



1. Make changes directly in **DEV Master**
  - Unit test the changes in DEV
2. Create a Revision Tag
  - Update Revision (Tracker) List
  - Note the History ID after the change above and use this for the Revision Tag.
3. Promote changes to Test from **DEV Master**
  - Use production quality data
  - Involve users if necessary
  - Perform integration and functional tests as needed
4. Promote changes to Prod from **DEV Master**
  - Promote only when Testing is complete
  - Put the model in offline mode if the changes are extensive so you have a chance to confirm the changes
  - Verify the changes by viewing the details of the Revision Tag
  - Confirm key changes in Prod
  - Make sure the Model is Online

# Multi-Threaded Development (aka Sandbox)

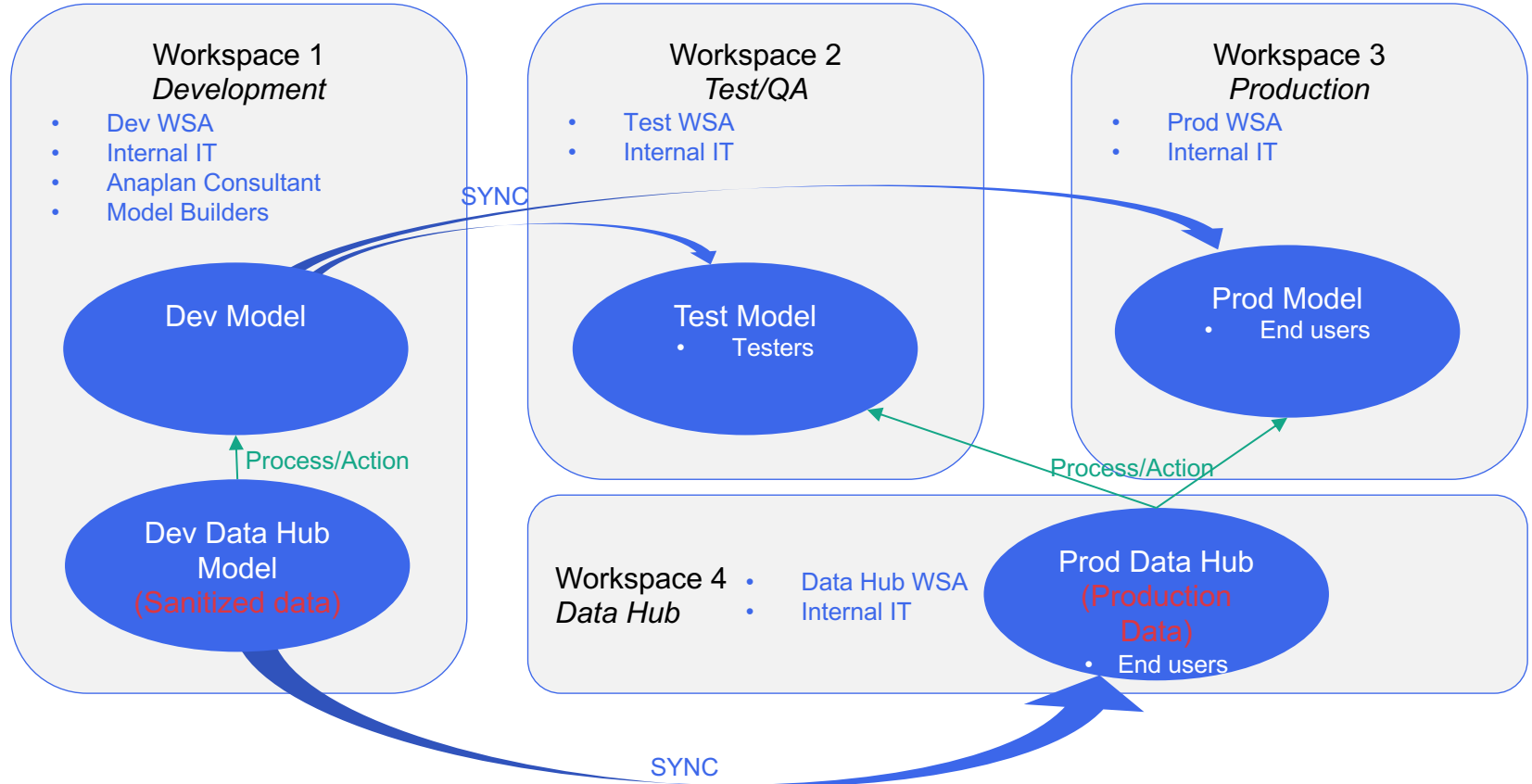
- When parallel development work is needed you can use this strategy. It does require in many cases for the developer to replicate the work manually due to ALM requirements.
- If one thread depends on changes from another thread, you have to manually re-create those changes.



# Multi-Threaded Development Steps (aka Sandbox)

1. Copy DEV Master into a new DEV Model
  - Always use DEV Master as the source for copying when multiple models (threads) are created
  - Name the model appropriately for the work being done – e.g. use a Backlog (User Story) ID number
2. Copy PROD (or TEST) as a target for the DEV copy
  - The copy should reside in a TEST workspace, separate from PROD
3. Make changes in [DEV x](#)
  - If there are changes in another DEV Thread that are needed, manually re-create those changes.
  - Unit Test
  - Create Revision Tag
    - DEV copy revision tag names should suffix with the backlog ID: major.minor.historyID.backLogID
4. Promote to QA x [from DEV x](#)
  - Perform Integration and Functional testing
  - Include business if needed
  - Make sure all testing is complete and passed before the next step
5. Replicate changes [from DEV x to DEV Master](#)
  - If no changes have taken place to DEV Master you can use ALM to promote the Revision tag. Otherwise manually re-create the changes. Use the Compare function in DEV x and use the output as a guide to re-creating the changes in DEV Master.
  - Perform Unit Testing to verify changes
  - Update Revision Tracker List
  - Create a new Revision Tag (this one is specific to DEV Master)
6. Promote [DEV Master to TEST](#)
  - Include Business if needed if they were not already involved in step 4
  - Make sure all testing is complete and passed before the next step
7. Promote [DEV Master to PROD](#)
  - See Single Threaded Steps, Step #4

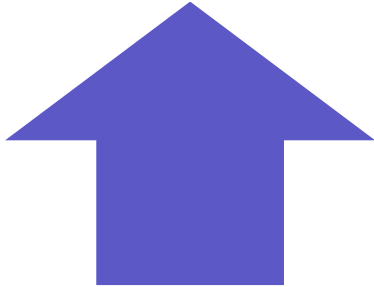
# Separation of Duties



# Release Strategy for Production Model (Example)

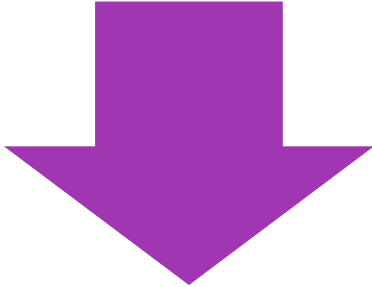
- ***Determined and managed by Anaplan Center of Excellence***
- Release cadence based on the release type & process calendar
  - Identify one time during the Week/Month where changes can be promoted, ***when the business is not disrupted (i.e. not during month-end)***. The exception is for Stopper Bug fixes.
  - Major –these are large changes that affect functionality and user experience. You must use change management to make sure the business is prepared.
  - Minor –these are smaller changes that do not require training and can include fixes that are not classified as blockers.
  - Stopper Bug Fixes –when the business is blocked, this must be released as soon as possible.
- Revision Tag Naming Conventions
  - Major.Minor.HistoryID–e.g. 1.0.123456
  - HistoryID is the latest ID from History that corresponds to the structural changes.
  - Use a Summary description about this release. Avoid too much detail –instead use the “Compare” function to see the specific changes if needed. Include the backlog ID in the description.
- Take Prod Offline
  - Sync changes, spot check the model and then turn Prod back into Online mode
- In DEV Master
  - Use a Revision Tracker List (non Production) that can be viewed in a Dashboard and can be used to finalize HistoryID for revision tag naming. You can copy the description you add here to the Revision tag.
- Copy-Archive PROD Model before any major change
  - You cannot roll back a model to a point in history if it affects structure so create the archive in case you need to revert the changes.

## Two Options to Create New Model



### Copy

- Quickly roll out to a deployed model because you can selective clear lists/modules and only reload subset of data



### Empty Model from Revision Tag

- Opportunity to fully re-test ALL integration actions to load master/transactional data

Regardless of approach, it is best practice to build “Delete from list” actions within each model.

# ALM in Action

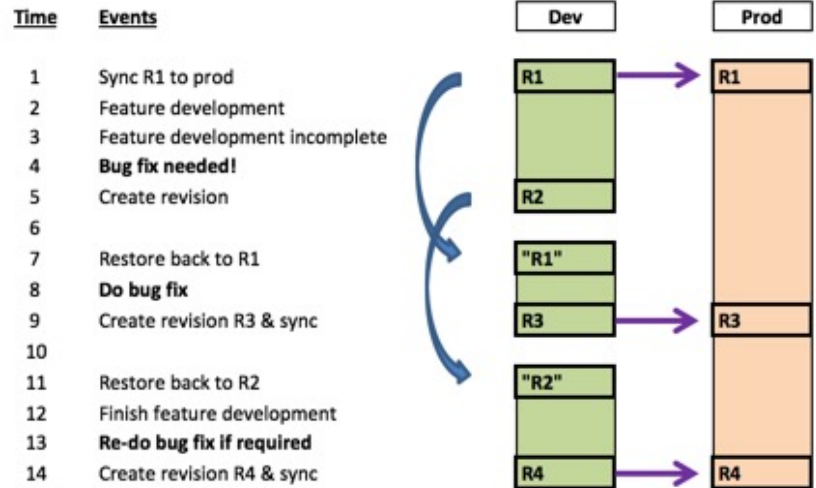
# Hotfix aka Back to the Future

- What is a Hotfix to a Deployed model
  - You have changes in progress that are not ready for production deployment but a bug fix is needed right away and you don't want the WIP to deploy.
- Prerequisite
  - Proper Revision Tag naming
    - Major.Minor.HistoryID – e.g. 2.1.123456
    - Or Date.Version\_HistoryID
  - Good team communications







# HotFix Steps in Dev Model

- Steps in DEV model
  1. Inform all model builders: do not make changes in DEV
  2. Note most current **History ID** – you will need this in a later step
  3. Identify **History ID** of latest revision to Production – that's where the Revision Tag naming comes in handy
  4. Restore to **History ID** (latest Prod revision tag)
  5. Implement Bug Fix
  6. Create Revision Tag – same naming convention but add Hotfix (e.g. 2.1.123457.Hotfix)
  7. Promote changes (optionally to TEST model first)
  8. Restore to **History ID** from Step 2
  9. Re-implement Bug Fix



## Revert to Last Revision

- This only activate under these conditions
  - Production Model was taken out of deployed mode, then someone made structural changes. Model likely put back into deployed mode. This effectively breaks the ALM chain.
  - If you take model out of deployed mode, the “Revert to Last Revision” will be active. Use this to restore ALM connectivity.
- Revert will not work under these conditions
  - The target model is Copied (creates quasi revision tag) – even if in deployed mode.
  - Someone explicitly creates a Revision tag in the target model

Revision Tags		
 Add Revision Tag	 Create Model From Revision	 Revert to Last Revision    Refresh
Synced	Title	Description

## Worst Case – ALM is Broken

- Create new DEV model from PROD
  1. Rename existing DEV model
  2. Copy PROD to new DEV (standard mode)
  3. In new DEV,
    - Re-create any changes from old DEV – use Compare Revisions to see specific changes. Create new Revision Tag
  4. Note: this DEV model will not have any prior History or Revision tags. This is the case with any copied model.
    - Your old DEV can act has an historical archive (for reference only)

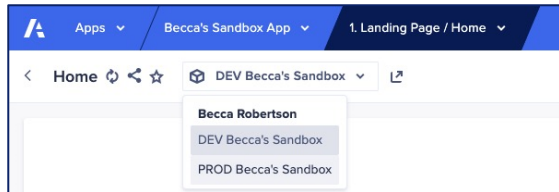
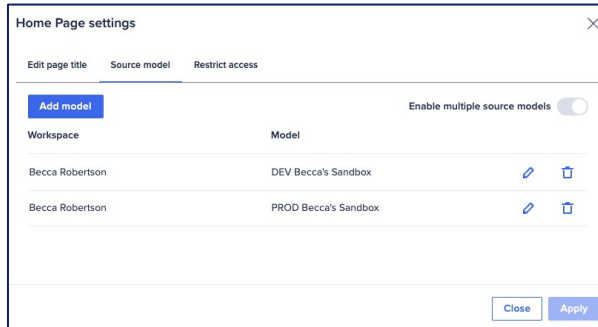
# Things to Remember...

- Model copy (Copy/Import/Create from Tag) erases all Model History in newly created model
- Revision Tags
  - Add at least one revision tag per day during model building
  - Add a revision tag after completing a significant piece of model functionality
- ALWAYS use Dev as source, mitigate chance of breaking the chain
- You can sync between two models in Standard Mode << but WHY?
- Cannot copy model across tenants (must go to Anaplan Support)
- ALM across workspaces:
  - Use Manage Model > Import to copy a Dev Model from a different workspace OR
  - Use Create Empty model from Revision (can only be saved to the same Workspace) - then Manage Model > Import
    - Delete the original empty model
- ALM can be automated by utilizing [Anaplan's public REST API](#)
- If all else fails, contact Support ASAP!

# Anaplan App Management

# ALM for Pages

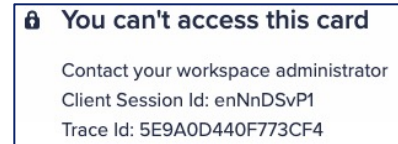
- Associate multiple models for a page and [associate additional models as potential data sources](#).
  - All models associated with a page should be part of the same ALM chain
  - Users can toggle to see page using different underlying models' data (they must have access to the models)



- Save vs Publish page changes
  - Save - Create/update pages against data from the development model in coordination with other model structural changes (Page name will included "Unpublished Changes" status in upper-right upon opening design mode)



- Publish – AFTER sync between models so that page/cards display properly, otherwise you get...



# Apps & Models

- Apps can ONLY be associated with models in the same workspace
- If you archive a model you automatically archive the app that is tied to it.
  - Repoint the App to a model that isn't going to get archived so that it doesn't go away
  - If you don't use ALM (WHY??), and you don't have the space to have two copies Unarchived, then point App to an "empty" model. Cards won't render, but repoint back to the right model, then they will work
- If you just create a copy of the App, then you lose any personal pages that were made
- App is tied to the Model GUID, NOT the model name
- If you DELETE a model, then you'll need to reach out to Support to restore/retrieve the APP – do this ASAP!!
- You can restore deleted APPS (as a Page Builder) from the home App page
- Apps cannot have the same name (even against those that are archived)
- If App has pages pointed to multiple models (A&B), and model B gets archived, then all pages that pointed to Model B go away
- History Log for Pages is on the roadmap!

