knowledge15

Everything as a Service

# Team Development:
## Beyond Update Sets


Joanna Nelson
Sr. Solutions Consultant – Developer Specialist
ServiceNow

Andrew Martin
Sr. Software Engineer
ServiceNow




servicenow

# Agenda

Motivation and Concepts

Instance Architectures

Using Team Development: Basics

Using Team Development: Additional Features

Best Practices

# Motivation and Concepts

**Team Development simplifies parallel development.**

# Motivation and Concepts

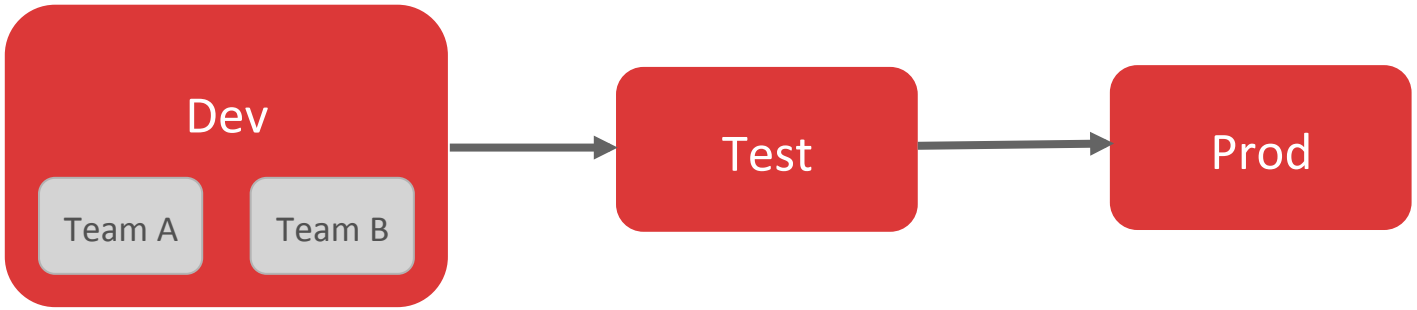
## Overview

- Problem Statement
- Solution
- Instance Architectures

# Motivation and Concepts

Problem Statement

- Most customers have more than one ServiceNow admin/developer working on more than one project




  - Separate update sets track changes independently, but only the latest change is active
  - What happens when I send my update set upstream?

# Motivation and Concepts

## Problem Statement (cont.)

- Releases of different customer-developed applications can share customized elements, for example
  - ACME Corp may have an ACME Script Include that is used by all ACME Corp developed applications (e.g. 'ACME Anvil Designer' and 'ACME Façade Designer')
- When different in-development releases customize the same element, the releases become dependent on one another, for example
  - ACME Anvil Designer is in the middle of a 2 month release cycle that has changed the ACME Script Include
  - ACME Façade Designer developed a new feature that changed the ACME Script Include and this change was made in the midst of the ACME Anvil Designer effort. ACME Façade Designer is ready to release but cannot because their changes include changes from the incomplete ACME Anvil Designer release

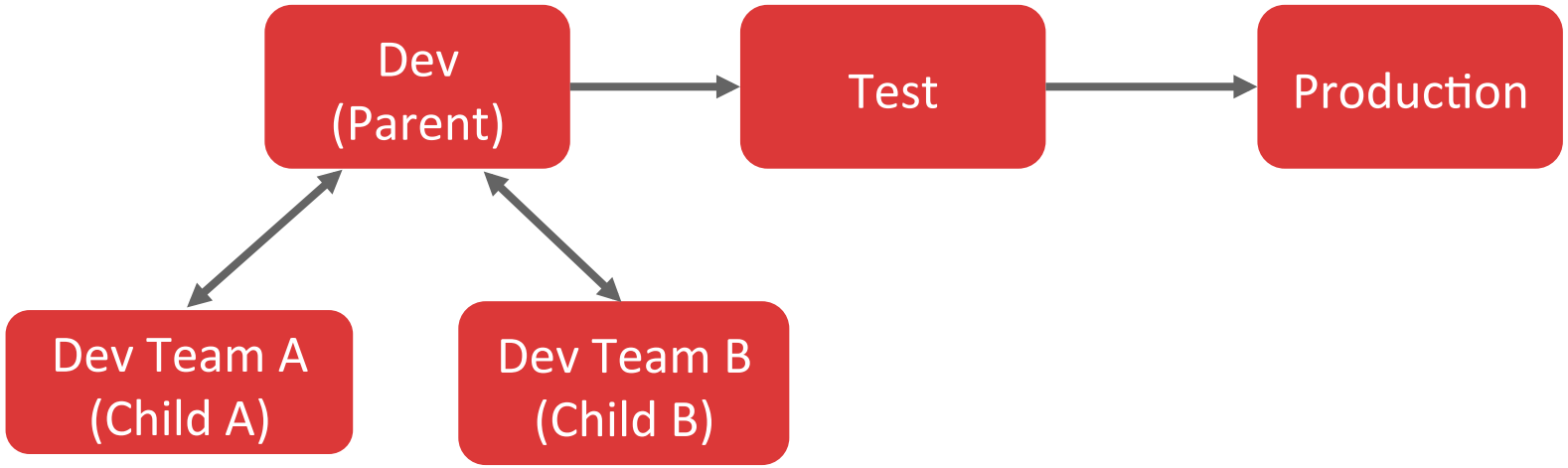
# Motivation and Concepts

## Solution

- Team Development Solves this problem by providing an instance architecture and associated functionality to support simultaneous customization of elements, with the ability to merge those customizations at a later date
  - This is functionally equivalent to branching in traditional source code control systems
- Child Instances serve as separate branches, allowing customizations independent of those on other Child Instances
- A Parent (or Hub) instance serves as a master branch
  - Children can pull customizations from the parent instance to their instance
    - Allows longer running development effort to stay up-to-date with current code base
  - Children can push their customizations to the parent instance
    - Conflicts must be merged on the child prior to pushing to parent

# Parallel Releases

Solution (cont.)




- Small fix related customizations and large development efforts can be done concurrently.
- Example: Your team deployed Version one of a product last release, but there are 4 bugs that need to be fixed. Your team has already begun development on the next product. In this example, the bugs can be fixed and the other product can be continue development without impact on each other.

# Motivation and Concepts

## Solution (cont.)

- Instances are provisioned as normal
- Instance architecture is defined by specifying a Child's Parent Instance
- Operations Include
  - Reconciling
  - Pulling
  - Resolving Collisions (i.e. merging)
  - Pushing
  - Comparing Instances
  - Back Out changes
  - Code Review

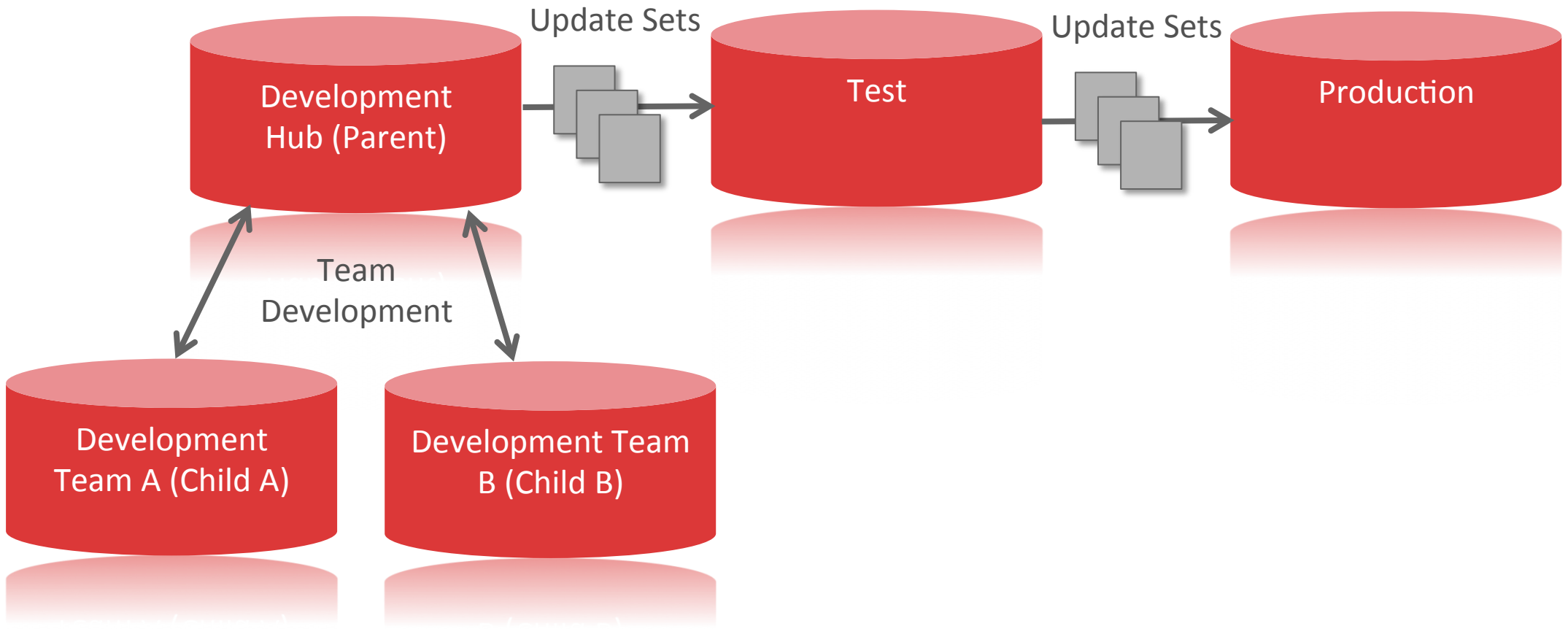
# Motivation and Concepts

## Solution (cont.)

- Here how ACME Corp can use Team Development to support the needs of both teams
  - ACME Anvil Designer works on Child Instance A. They do not push their changes to the Parent until they are development complete.
  - ACME Façade Designer works on Child Instance B. When they are dev-complete, they push their changes to the Parent. Their changes can now proceed up the rest of the stack and be released.
  - ACME Anvil Designer now pulls the changes made by ACME Façade Designer down from the parent and into their instance. After the pull, the ACME Anvil Designer team will have to merge the changes made by the ACME Façade Designer team into their version of ACME Script Include. When ACME Anvil Designer is then pushed to the Parent, both applications will contain the code they need.

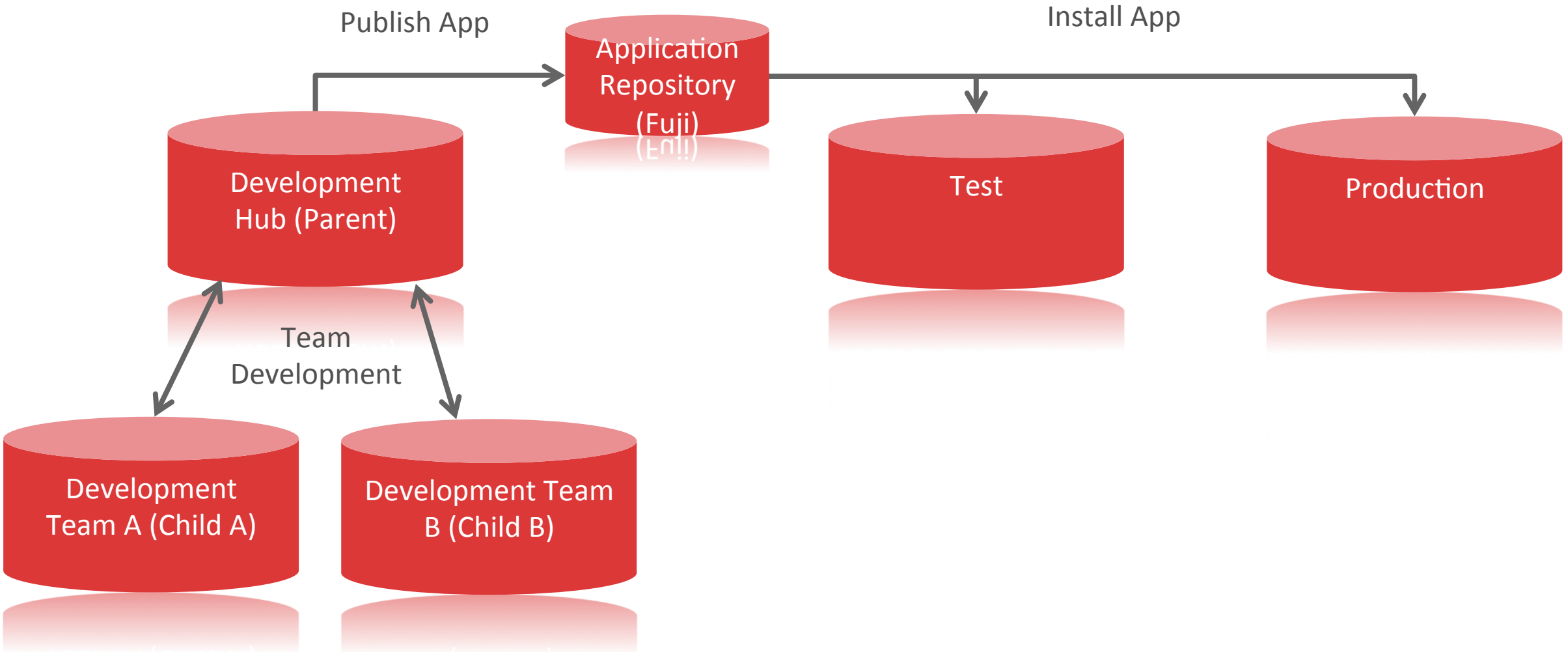
# Motivation and Concepts

Instance Architectures: Typical Architecture



# Motivation and Concepts

Instance Architectures: New Option for Moving Customizations (in Fuji)



# Using Team Development: Basics

**How to connect instances, pull, resolve collisions and push.**

# Using Team Development: Basics

## Overview

- Connecting Instances and Reconciliation
- Versions
- Team Dashboard
- Pulling
- Resolving Collisions
- Pushing

# Using Team Development: Basics

## Connecting Instances and Reconciliation

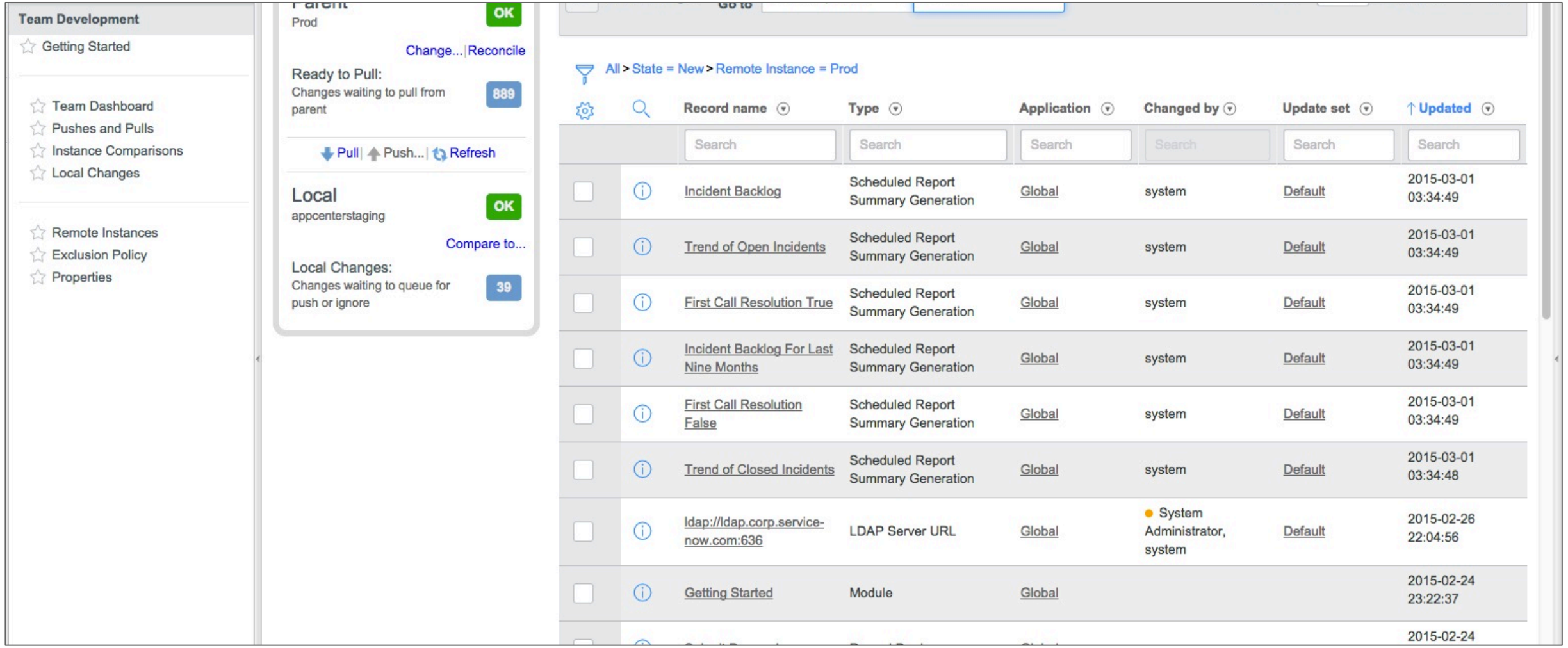
- Typically done just once, after instances have been provisioned and before team development begins
- Child Instance is configured to reference Parent
  - Team Development > Remote Instances > New
- Child Instances can also be configured to reference Peers (i.e. other Children)
  - Allows comparisons between instances
    - Changes can be selectively applied between peers (prior to any pushes to parent)
  - Team Development > Remote Instances > New
- Reconciliation establishes baseline of Local Changes relative to a Parent
  - Done after when a new parent is set or external event occurs (e.g. parent clone or failover)

# Using Team Development: Basics

## Versions

- Team Development functions on Versions
  - The version table tracks changes to a customized record over time so that administrators can compare or revert to specific versions later
  - A Pull operation queries versions from the parent instance and loads then into the child
  - Both current versions and historical versions are pulled, providing a history of changes made upstream
- The Local Changes table shows changes made on the child relative to its parent
  - Changing parents will reconcile the child to its new parent, creating a new list of local changes
  - Local changes are then queued and pushed to the parent when ready

# Team Development Dashboard



# Using Team Development: Basics

## Pulling

- Retrieves Changes from Parent Instance and applies them to Child Instance
- Creates Collisions when a customization has been changed in both Parent and Child
  - Customization associated with collision is not applied until collision is resolved
  - User resolves collision by either accepting remote customization or accepting local customization
  - Accepting remote causes remote customization to become current
  - Manual merge is achieved by editing local customization to include needed code from remote, then accepting local
- Pulling and Resolution of all Collisions is required before Pushing

# Using Team Development: Basics

## Resolving Collisions

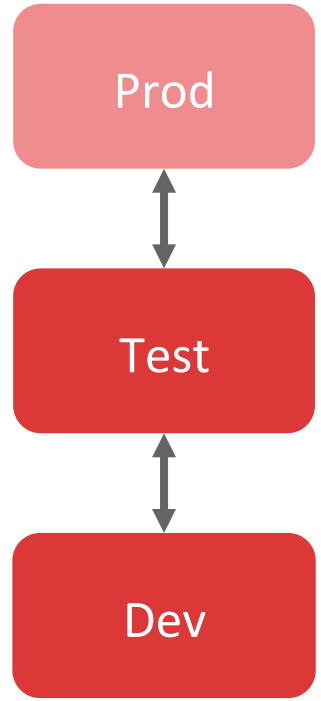
- A collision occurs when two people change the same record on different instances

1. You change the incident list layout on TEST
2. I change the same incident list layout on DEV
3. I pull your layout change from TEST to DEV

Result: Collision

1. You change the incident list layout on Test
2. You pull your layout change from TEST to DEV
3. I change the incident layout in DEV
4. I push my layout change from DEV to TEST

Result: No Collision



# Using Team Development: Basics

## Pushing

- User selects Local Changes on Team Dashboard and selects ‘Queue for Push’
  - Only Local Changes that are actually different between parent and child are shown/selectable
- Before items can be pushed, there can be no items ready to Pull and all Collisions must be resolved
- Clicking ‘Pull…’, user must specify name of update set that the Pushed changes will be placed in
  - Update set will be created on parent if it does not yet exist
- Clicking ‘Push Changes’ pushes changes to parent instances
  - Changes are loaded and committed and become available for pull to other children

# Using Team Development: Basics

## Pushing (cont.)

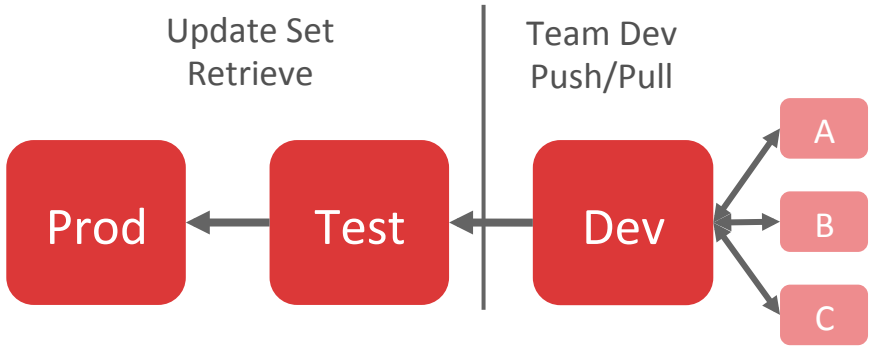
- A Local update set can be used to organize your changes
  - They are “change lists”
  - They can assist in sorting and filtering in the Local Changes list on the Team Dashboard
- Other ways to filter include:
  - Filtering based on User( updated, or created field), application set, or date
- Recommended Filter:
  - Using Application Sets
    - The Local Change appears in the update set of the person that most recently changed it, so the local change effectively moves from one update set to another when a new person changes that file. Similar for Updated By
    - Application assignment is effectively permanent, so local changes will stay put

# Using Team Development: Basics

## Pushing (cont.)

- A Push results in a completed Update Set in the parent instance
  - It records who pushed what and from where
  - Update sets can then be merged in the parent to reduce clutter
  - These update sets have the same functionality as traditional update set use

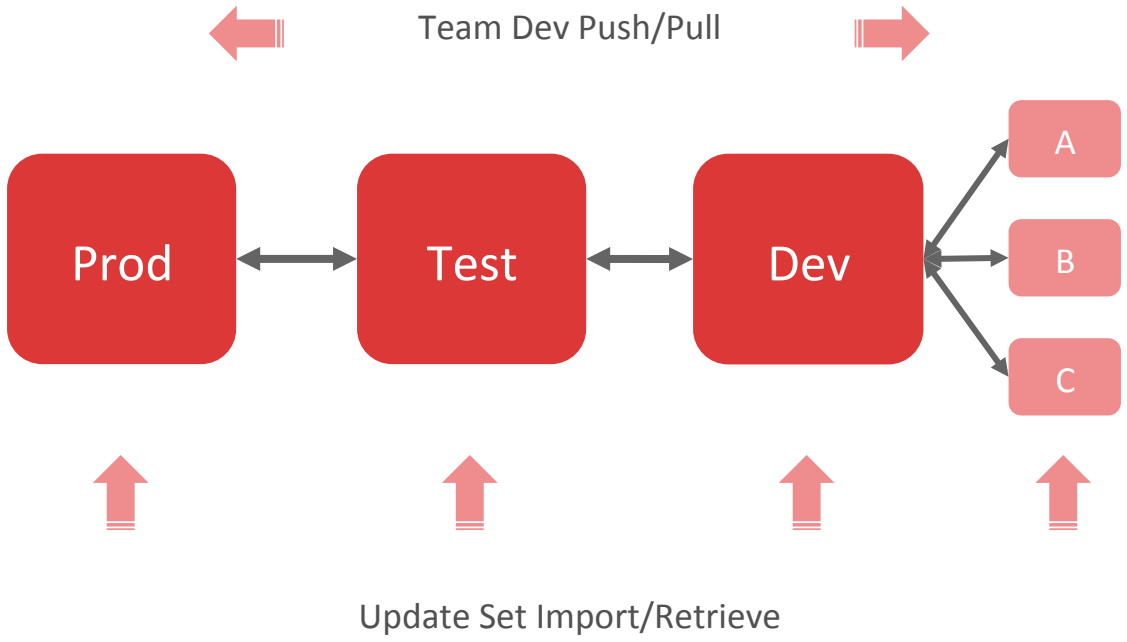
- Example Deployment Process



# Using Team Development: Basics

## Pushing (cont.)

- When using Team Development, you are not forced to use only Team Development



# Using Team Development: Additional Features

## Exclusion Policy, Code Review, Comparing Peer Instances

# Using Team Development: Additional Features

## Overview

- Exclusion Rules and Ignoring Changes
- Code Review
- Comparing Peer Instances

# Using Team Development: Additional Features

## Exclusion Rules and Ignoring Changes

- When in a non production instance, you may have certain configurations that you never want to push, such as disabling the sending of email or changing the CSS color of your banner
- Two Methods for achieving this
  - Ignoring a Change
  - Exclusion Policy
- Ignoring a Change
  - Mark a Local Change (sys_sync_change) as Ignore and it will not be pushed to parent
  - Ignored Local Changes are shown in related list in Team Dashboard
  - Useful for targeting one-off Local Changes that should not be pushed

# Using Team Development: Additional Features

## Exclusion Rules and Ignoring Changes (cont.)

- Exclusion Policy
  - Team Development > Exclusion Policy > New
  - A query (table and conditions) specifies which records are excluded
  - Can specify whether the exclusion applies to pushes, pulls or both
    - Can only specify table (not conditions) if pull is involved
  - Useful for larger scale exclusions
  - For Push exclusions, Local Changes (sys_sync_change) are not generated if customization matches Exclusion Policy
    - Customer Update (sys_update_xml) is still created and appears in Update Set for customizations made on instance
  - For Pull exclusions, Local Changes on Parent are not pulled if they match the table in Exclusion Policy

# Using Team Development: Additional Features

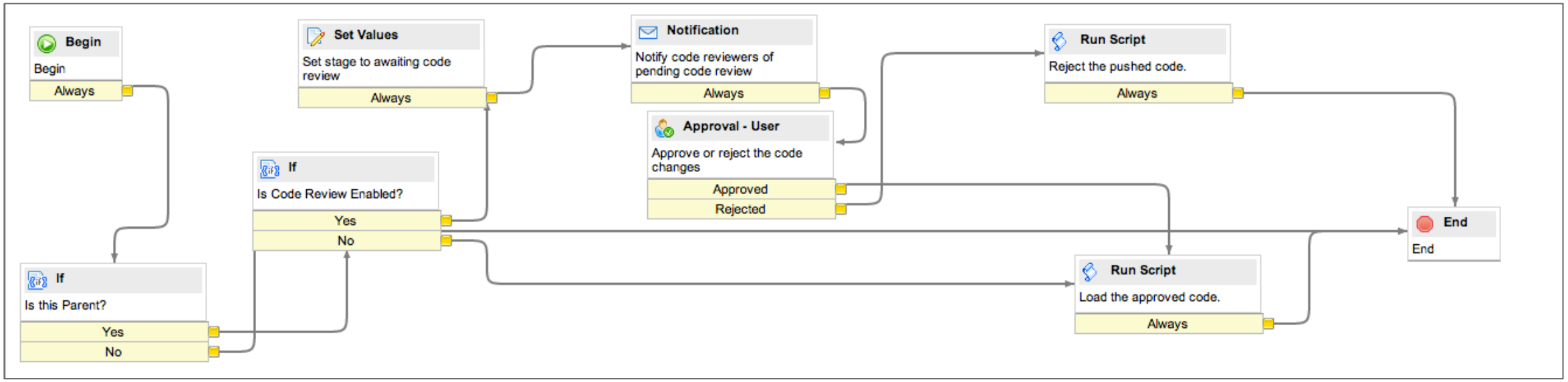
## Code Review

- Code Review is an optional feature that provides a code review checkpoint before pushes can be consumed by a Parent
- Enabled on Parent
  - Team Development > Properties > set code review property to true
  - Introduces ‘Code Review Requests’ module
- Reviewers can accept or reject
  - Submitters can cancel their pushes that are in the ‘Awaiting Code Review’ stage (removes push from parent)
- All pushes and pulls are blocked while there are pending code reviews

# Using Team Development: Additional Features

## Code Review (cont)

- Code review allows a development team to closely monitor what is being pushed through a review process



# Using Team Development: Additional Features

## Comparing Peer Instances

- Two peer instances can be compared and Local Changes from one can be applied to the other without pushing to Parent
  - Define Peer Instance as Remote Instance (once)
  - Team Development > Team Dashboard > Compare To…
    - Select Remote Instance
  - Creates Instance Comparison Record with two related lists of Local Changes
    - On Local and not on Remote
    - On Remote and not on Local
      - These can be compared and the remote change can be committed to local
- Can be useful in reviewing the changes other teams are making and identifying potential future collisions

# Best Practices

**Practices that lead to a quality development and release process**

# Best Practices

## Practices that lead to a quality development and release process

- Understand the technical dependencies between releases and partition the work across teams and child instances so as to minimize collisions
- Use Team Development on the DEV portion of the stack and update sets going from the DEV parent upward (to TEST then to PROD)
- Ensure that the Releases are identified before development work begins and that the order of Releases is established when there are overlapping customizations between releases
  - This drives who gets to push to the DEV parent first and which children should be pulling from the DEV parent
  - This decision needs to be made with input from the development team since they best understand what code is going to be customized

# Best Practices

## Practices that lead to a quality development and release process

- For release involving multiple developers, nominate a single developer who to manage all aspects of the Update Set lifecycle
  - Responsible for creating, pushing, and merging update sets associated with dev effort
  - Responsible for communicating the current update set to rest of team
  - Responsible for reviewing update sets and communicating pushes to entire team
  - Responsible for documenting deployment process
    - Order of applying update sets, execution of scripts, and data that must be loaded
- When possible, favor shorter release cycles
  - Reduces quantity of in-flight work, ensuring better match between prod and dev
  - Simplifies work associated with re-cloning dev stack

# Best Practices

## Practices that lead to a quality development and release process

- Always review updates contained in an Update Set before transferring
  - Look for updates associated with other dev efforts and updates associated with testing
  - Watch for Sys Properties and Integration End-Points changes
    - Ex: pushing sys_property change that directs all email to test email account
  - Move updates to scrap update set rather than deleting the update
- Limit who has admin on the development stack – use impersonation
  - Ensures un-intended customizations are not made by non-developers (e.g. doc, testers)
- Use ability to make customizations directly in DEV HUB judiciously
  - Can be convenient way to provide a change for all children, but can be disruptive too
    - You don't have the ability to verify feature before it's consumed by children

# Top Takeaways

1. Team Development can be used for everyone, from one developer to many developers.

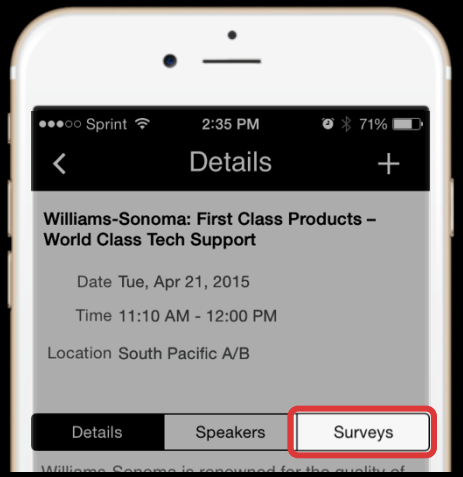
2. Parallel development is possible and powerful.

3. Team Development and Update Sets can live in harmony.

## How Did We Do?

Your feedback on this session helps us deliver great content.

Please take a moment to complete a session survey in the Knowledge15 app or use the survey forms at the back of the room.




# Everything as a Service

# Thank You


**Joanna Nelson**

*Sr. Solutions Consultant – Developer Specialist*

ServiceNow

joanna.nelson@servicenow.com

**Andrew Martin**

*Sr. Software Engineer*

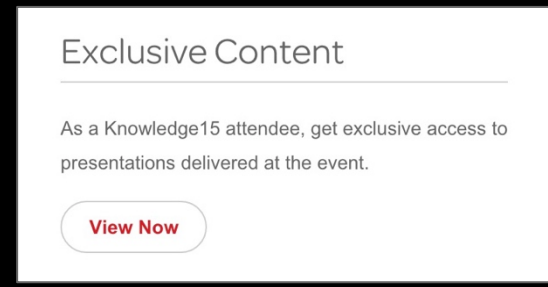
ServiceNow

andrew.martin@servicenow.com


## Get Presentations

As a Knowledge15 attendee, you have exclusive access to breakout and lab session content from the event.

1. Go to knowledge.servicenow.com
2. Log into the community
3. Click on **View Now** button




knowledge.servicenow.com