Defect Close-Declined change

The problem we’re trying to solve is when a defect is set to `State = ‘Closed Declined’`, the Closed Date field is not automatically populated.
The Closed Date field is only populated when a defect is set to `State = ‘Closed’`.
We have a Defect Arrival/Kill chart that needs to include ‘Closed Declined’ defects, and right now we can’t include them because there is no Closed Date.

The solution is to populate the Closed Date field on a defect when the State is set to ‘Closed Declined’.

On Defects:

If the State is set to ‘Closed Declined’ AND the Closed Date is blank (not populated), take the following actions:

1. Set the State to ‘Closed’
2. Then Set the State back to ‘Closed Declined’

We can walk through the steps needed to test the webhook when you’re ready to start testing. You will basically create defects in the Cox Training environment and update the defect with the various states that will trigger or not trigger the webhook

Testing

Scenario 1: The state is changed to ‘Closed Declined’ and the Closed Date is blank.

1. In Rally – create a new defect in the "!ift & Shift - DRS" project and click the ‘Create’ button. State defaults to ‘Submitted’, and Creation Date is populated with today’s date.
2. In Rally – change the State to ‘Closed Declined’, and click the ‘Save’ button.
3. This will trigger the Webhook to fire.
4. Validate that the Webhook fired and updated the State and Closed Date.
5. Validate In Rally – The Closed Date is populated with today’s date. The State = ‘Closed Declined”.

Scenario 2: The state is changed to ‘Closed Declined’ and the Closed Date already has a date value in it.

1. In Rally – create a new defect in the "!ift & Shift - DRS" project and click the ‘Create’ button. State defaults to ‘Submitted’, and Creation Date is populated with today’s date.
2. In Rally – change the State to ‘Closed, and click the ‘Save’ button.

Result: This will populate the Closed Date with today’s date. Note the Last Updated date/time. In Rally – change the State to ‘Closed Declined”. *Result:* The Webhook fires and exits the operation. The Late Updated date/time remains unchanged from step #2 above.

Test these 2 scenarios with multiple defects in multiple projects.:w

Serverless Deployment

Prerequisites

- Node.js 6.x+
- A terminal

Updating a deployment

1. Follow the `Removing a deployment` steps in the README from the **PRIOR** version.
2. Follow the `Full Deployment Walkthrough` steps in the README from the **CURRENT** version.

Full Deployment Walkthrough

Setup

1. Download the following files: `cox-webhook-reflector_<version>.zip`
2. `unzip cox-webhook-reflector_<version>.zip -d cox-webhook-reflector_<version>`
3. `cd cox-webhook-reflector_<version>`
4. `npm install yarn`
5. `./node_modules/.bin/yarn`
6. `cp setup.sh.example setup.sh`
7. Edit `setup.sh` and fill in the required values.
8. `source setup.sh`
9. (Optional) If your environment requires AWS Lamda to run using pre-existing IAM Roles.
    1. Edit `serverless.yml`
    2. Add the `role: Your_Role_ARN` in the `provider` section. It should look like:

    ```
    provider:
       name: aws
       role: Your_Role_ARN
       runtime: nodejs6.10
       ...
    ```

Deploy the reflector code

1. `npm run deploy` - This deploys the reflector code to AWS Lambda
2. Test the labmdas by loading the value displayed for the `GET` endpoint in a web browser. It should say `Thank you for visiting`.

Deploy the webhooks

1. Edit `setup.sh` and set the value of `WEBHOOK_TARGET_URL=` to the value of the `POST` endpoint from the above deploy step output, or run `npm run info` to display the endpoint.
2. `source setup.sh`
3. `npm run create-webhooks` - This creates the webhooks in Agile Central
4. Test the webhooks:
    - Update a portfolio item `InvestmentCategory` in the workspace. You should see log output after a few seconds, and see child Portfolio Items InvestmentCategory updated to match the parent when you reload the children items in Agile Central.

    - Create a portfolio item with `Send Epic to SNOW Indicator (DSE ONLY)` or `SNOW Status (Don't Touch Admin ONLY)` set. After a few seconds the reflector will clear these values as they should not be set if user copies a portfolio item with them set.

Removing a deployment

1. `source setup.sh` - Make sure the values in `setup.sh` are set as described above
2. `npm run delete-webhooks` - This will remove all webhooks in all workspaces related to this reflector.
3. `npm run undeploy` - This will remove the AWS Lambda reflector code.

Rolling update of an existing deployment

1. Follow the steps for `Full Deployment Walkthrough` for the new version.
2. Once it is running, follow the steps for the `Removing a deployment` in the old version.

Monitor reflector logs

1. `./node_modules/.bin/serverless logs -f app -t`

Listing deployed webhooks

1. `source setup.sh` - Make sure the values in `setup.sh` are set as described above
2. `npm run list-webhooks`

Enabling the reflector for additional Agile Central Workspaces

The same AWS lambda deployed above can be used by multiple workspaces. However, the webhooks are workspace specific.

First, make sure your AWS lambda's are still deployed:

1. `npm run info`

Then, make sure the API key and target environment variables are set:

1. `export WEBHOOK_RALLY_API_KEY=` - See above for value
2. `export WEBHOOK_TARGET_URL=` - See above for value

Then, for each additional workspace you want to enable:

1. `export WEBHOOK_RALLY_WORKSPACE_UUID=<UUID of that workspace>`
2. `npm run create-webhooks`

Security Notes

The reflector is structured as Node.js code which is deployed as an AWS Lambda, triggered by AWS API Gateway. The API Gateway is configured to be an HTTPS endpoint with an "unguessable" path. `GET` or `POST` requests to this path will trigger the Lambda code. All other requests will be rejected by the API Gateway. This path can be configued at deployment time by editing `package.json` and setting a random value to `TARGET_URL_SUFFIX`.

The data sent by the webhook from Agile Central to the Lambda over HTTPS includes full detail (without attachments) of new or modified Portfolio Items or User Stories (HierarchicalRequirements).

The AWS Lambda uses environment variables that contain a Agile Central API key, and Agile Central workspace ID to query and update Agile Central over HTTPS. The data returned from Agile Central contains information about Portfolio Items and User Stories, including:

- URL
- Name
- Children URLs
- Investment Category
- Release
- FormattedID

Data that is updated in Agile Central includes the following fields:

- Investment Category
- Release

The primary risk is that someone other than Agile Central triggers the Lambda code with a carefully crafted message which would cause the Lambda to update data in the Agile Central. This risk is migitaged by the following:

- bad actor would need to know the "unguessable" portion of the path. This is only available to someone with access to the AWS account containing the deployment OR subscription admin access to Agile Central.
- bad actor would need to know the subscription ID and valid artifact references from Agile Central. This is only available to users that already have read access to Agile Central.

The environment variables used during the deployment contain API keys for an Agile Central subscription admin. These should be treated as passwords and not stored on an deployed environment.

Test Plan

SNOW Fields

`Send Epic to Snow Indicator (DSE Only)` and `Snow Status (Don't Touch Admin Only)`

Feature

Test		Expected Result
create new without SNOW fields set	-	new has has SNOW fields blank
create new with Snow Indicator	-	new item has SNOW Indicator cleared
create new with Snow Status	-	new item has Snow status cleared
create new with Snow Indicator an Snow Status	-	new item has SNOW fields blank
copy new without SNOW fields set	-	new item has SNOW fields blank

Test		Expected Result
copy new with Snow Indicator	-	new item Snow Indicator cleared
copy new with Snow Status	-	new item Snow Status cleared
copy new with Snow Indicator an Snow Status	-	new item Snow Status and Snow Indicator cleared

Epic

Same tests as Feature but at Epic level

Investment

Same tests as Feature but at Investment level

Business Initiative

Same tests as Feature but at Business Initiative level

Release

Test		Expected Result
create story without parent, release unset	-	**TODO**
create story without parent, release set	-	**TODO**
create story with parent, release unset	-	**TODO**
create story with parent, release set	-	**TODO**
update story release, without parent	-	**TODO**
update story release, with parent with release unset	-	**TODO**
update story release, with parent with release set	-	**TODO**

Investment Category (IC)

Business Initiative

Test		Expected Result
create with IC unset	-	IC remains unset
create with IC set	-	IC remains set
change IC does not update descendents ICs	-	descendents ICs unchanged

Investment

Test		Expected Result

Test		Expected Result
create without parent, leave IC unset	-	IC is "None"
create without parent, set IC	-	IC is set value
create with parent IC unset, leave IC unset	-	IC is "None"
create with parent IC unset, set IC	-	IC is set value
create with parent IC set, leave IC unset	-	IC is "None" (ignores parent BI)
create with parent IC set, set IC	-	IC is set value (ignores parent BI)
change IC without parent	-	IC is new value
change IC with parent IC unset	-	IC is new value
change IC with parent IC set	-	IC is new valeu (ignores parent BI)
reparent with IC unset to no parent	-	IC remains unset
reparent with IC set to no parent	-	IC remains set
reparent with IC unset to parent with IC unset	-	IC remains unset
reparent with IC set to parent with IC unset	-	IC remains set
reparent with IC unset to parent with IC set	-	IC remains set (ignores parent BI)
reparent with IC set to parent with IC set	-	IC updated to parent value
change IC updates descendents	-	descendent ICs updated
reparent does not update descendents	-	descendent ICs unchanged (ignores parent BI)

Epic

Test		Expected Result
same tests as Feature	-	**TODO**
change IC updates descendents	-	descendent ICs updated
reparent updates descendents	-	descendent ICs updated to match new parent

Feature

Test		Expected Result
create without parent, leave IC unset	-	IC is "None"
create without parent, set IC	-	IC is set value
create with parent IC unset, leave IC unset	-	IC is "None"
create with parent IC unset, set IC	-	IC is set value

Test		Expected Result
create with parent IC set, leave IC unset	-	IC reset to parent IC value
create with parent IC set, set IC	-	IC updated to parent value
change IC without parent	-	IC is new value
change IC with parent IC unset	-	IC is new value
change IC with parent IC set	-	IC updated to parent value
reparent with IC unset to no parent	-	IC remains unset
reparent with IC set to no parent	-	IC remains set
reparent with IC unset to parent with IC unset	-	IC remains unset
reparent with IC set to parent with IC unset	-	IC remains set
reparent with IC unset to parent with IC set	-	IC updated to parent value
reparent with IC set to parent with IC set	-	IC updated to parent value

Limitations

1. Assumes that Portfolio Item Hierarchy is `BusinessInitiative` -> `Investment` -> `Epic` -> `Feature`
2. Assumes that an unset Portfolio Item InvestmentCategory value is `None`

Possible Enhancements

1. Lambda encrypt environment variable at rest
2. Recommended service users for webhook creation and reflector
3. Improved log messages
4. Readme clarification
5. Node install
6. Terminal / Powershell
7. Proxy setup in NPM
8. Cloud 9 as standardized environment
9. Automate get Workspace UUID
10. Troubleshooting steps, coorrective actions

Developer Information

See DEVELOPER_README for information regarding:

- Local environment setup
- Production or Test Deployment
- Unit tests and validations.

ISSUES

2023-01-16: issue was related to user stories rooted in Features, it was trying to update the field CAIBenefit in US which does not exists.