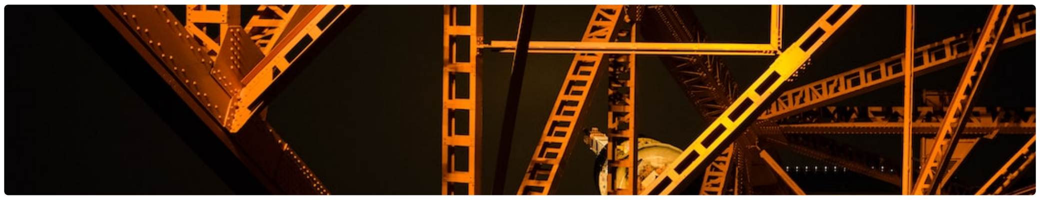
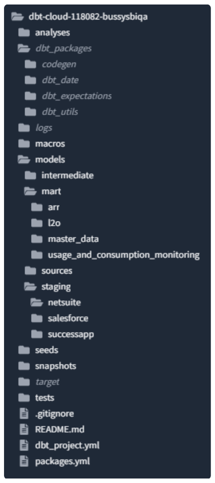


dbt (data building tool) - Dynatrace Best Practices

Project Structure

A project contains several folders. 	**Analyses**	Contains models that doesn't fit into the regular model structure and we still want to be version controlled. Any **.sql** files in this folder will be compiled but not materialized. You can find the result in **target/compiled/{project name}/analyses/{file name}.sql**	
	Macro	Contains custom macro that can be re used across all models.	
	Models	Must contains at a minimum the sub-folders below	
		sources	Must contain one YAML file per source (i.e salesforce.yml, netsuite.yml)
		staging	Must contain one sub folder per source (i.e salesforce, netsuite, successapp). Salesforce is using Fivetran dbt packages “salesforce_formula_utils”. consequently uses a “base” subfolder for the raw models (based on the macro : {{salesforce_formula_utils.sfdc_formula_view(source_table='<table_name>',source_name='salesforce', full_statement_version=true) }}. Staging models reference the base models and follow the rules defined below.
		intermediate	Contains models that are not exposed to the business either via self service (ext_ schema or Sigma) or governed analytics (PowerBI dashboard). A logical step towards creating a final data set. Typically used for: • Breaking up a very large fct_ or dim_ model into smaller pieces to reduce complexity. • Creating a reusable data set to reference in multiple downstream fct_ and dim_ models.
		mart	Final model which is robust, versatile, and ready for consumption either via self service (ext_ schema or Sigma) or governed analytics (PowerBI dashboard).
	Seeds	Contains csv files that are loaded to the Datawarehouse. The CSV files are version controlled. Can contain list of mapping of country code to country name, a list of IDs to exclude from analysis. Should never contain production data containing sensitive information such as PII or Sales / Financial data.	
	Snapshot	Contains **.sql** files with snapshot logic.	
	Tests	Contains **.sql** files with singular test logic.	

```
dbt-cloud-118082-bussysbiqa
  analyses
  dbt_packages
    codegen
    dbt_date
    dbt_expectations
    dbt_utils
  logs
  macros
  models
    intermediate
    mart
      arr
      l2o
      master_data
      usage_and_consumption_monitoring
    sources
    staging
      netsuite
      salesforce
      successapp
  seeds
  snapshots
  target
  tests
  .gitignore
  README.md
  dbt_project.yml
  packages.yml
```

Sources

YAML file should use the naming convention: **<source name>.yml**. For instance, for salesforce source data, the YAML file should be saleforce.yml. For NetSuite source data, the YAML file should be netsuite.yml.

The YAML file should contain at a minimum:

sources: - name: salesforce schema: salesforceprd	name: name for the source that will be used in {{source...}}. <we need to decide if we want to keep the schema name or create a user friendly alias. schema: if name is the same as the schema we don't to specify the schema. It is a good practice though to specify the schema name.
tables: - name: order description: Represents draft and activated orders	List all the tables that will be added to the staging folder for this specific source. Add a description that will help other developers or the business to better understand the purpose of the objects.
meta: owner: "thierry_krumeich@dynatrace.com" model_maturity: in hard frequency: Polls every 60 minutes	meta identifies: • owner: who is accountable to maintain the source pipeline for this objects • model maturity: even when developing in PRD, model maturity informs about the stage of development. • frequency: specifies how often the source object is refreshed. It will help configuring the data freshness test.
columns: - name: sfid description: unique identifier for an order	The minimum column is the primary key with the unique and not_null generic test. If a primary key doesn't exist, a surrogate key can be created using dbt_utils package.
tests: - unique tags: ['dps_ga','uniqueness '] - not_null tags: ['dps_ga','completeness']	The minimum test is to verify that the Primary Key is **unique** and **not null**. Add tag(s) for test selection.
- name: related_dynatrace_account__c description: FK to dynatrace_account sfid tests: - not_null tags: ['dps_ga','completeness '] - dbt_utils.relationships_where: to: ref('stg_sfdc_dynatrace_account') field: sfid from_condition: dps_ga__c is true to_condition: bas_account_status__c = 'Active' tags: ['dps_ga','util','completeness'] config: severity: error error_if: ">500" warn_if: ">10"	If the table includes Foreign Key(s), the minimum test is to verify the column is not null and the relationship to a table Primary Key is complete. Add tag(s) for test selection.

Staging

Staging models are models that reference the raw data you ingest directly into your data warehouse.
These are the only models that read from the raw data source. They represent a 1:1 relationship between the source and a staging model (one column of the source object is mapped to one column of the staging model).
They include basic computations (such as bytes to gigabytes), renaming, type casting and categorization, for instance using CASE WHEN statements based on a single column (i.e. CASE WHEN <column> = <value a> THEN<result a>, WHEN <column> = <value b> THEN <result b> ELSE <result c) END AS <alias>).
They don't include any joins, aggregations or filters.

Folder Structure

Contains one sub-folder per source (i.e. salesforce, netsuite, zendesk, successapp ...)

Naming convention:

Global File:

stg_<source>_<entity>.sql (stg_snow_consumption. sql). There are exceptions to the general naming convention. See detailed below for each data source.

Global Column:

The preferred method to generate staging model is to use gencode (i.e. **{{generate_base_model('salesforce', 'sbqq__quoteline__c')}}**) and to add alias to the columns.

Use snake case for alias name.
Timestamps alias are named <event>_at (i.e. createddate alias is created_at).
Date alias are named <event>_date.
Booleans alias name should be prefixed with is_ or has_ (i.e. is_active_customer and has_admin_access).
Price/revenue fields should be in decimal currency (e.g. 19.99 for $19.99; many app databases store prices as integers in cents). If non-decimal currency is used, indicate this with suffix, e.g. price_in_cents.
Avoid using reserved words as alias name.

Consistency is key! Use the same field names across models where possible (i.e. dt_account_uuid for idmaccountuuid and accountuuid).

For each data source the alias naming convention can differ. Review below data sources.

Source Specific

Salesforce (sfdc)

File Name

standard object, not ending with double underscore and c: use general naming convention.
custom object, ending with double underscore and c: drop the double underscore and c. If an objects with the same name without double underscore and c exists, remove one underscore.

Column Name

General rule: Create an alias using field label from salesforce object manager.
Exception: if a field name is commonly used by the business or already defined in OvalEdge (i.e. dt_account_uuid for IDM Account UUID or capability for sku_type), use the business name or OvalEdge Business Glossary.

Always use sfid (18 digit - case insensitive) for primary keys and alias the column sfid.
Always use sfid (18 digit - case insensitive) for foreign keys and alias the column <event>_sfid (i.e. orderid as order_sfid).

NetSuite (ns)

Zendesk (zend)

Totango (tango)

Snowflake (snow)

Objects from Snowflake shared with BusSys - BI are already curated. They should not require any additional light transformation in staging.

Materialization:

View.

Intermediate

Folder Structure

Intermediate folder will be structured with sub-folders:

- One **global** sub-folder which is not business domain dependent. For instance, int_ sfdc_product or int_sfdc_consumption_summed. are good candidate for global sub-folder because they are business domain agnostic.
- Several sub-folder representing either **business domain** or **workstream**. Each sub-folder is the responsibility of one lead. The lead is accountable for the quality of the data product delivered to our business customer. The lead will be responsible for approving any codes deployed from this sub-folder to the production schema in the data warehouse.

Naming Convention

The general rule to name an intemediate model file name is to use the format: int_<topic>_<additional_context>.sql. For instance: int_ sfdc_usage_pivoted_to_product or int_sfdc_product.
The objective is to make it easy for anybody to quickly understand what's happening in that model, even if they don't know SQL.

Materialization

Start with a view. When the view gets too long to query for end users,
Make it a table. When the table gets too long to build in your dbt Jobs,
Build it incrementally. That is, layer the data on in chunks as it comes in.

Mart

Folder Structure

Mart folder will be structured with:

- One **global** sub-folder which is not business domain dependent. For instance, dim_product or dim_dt_account or fct_usages are good candidate for global sub-folder because their are business domain agnostic.
- Several sub-folders representing either **business domain** or **workstream**. Each sub-folder is the responsibility of one lead. The lead is accountable for the quality of the data product delivered to our business customer. The lead will be responsible for approving any codes deployed from this sub-folder to the production schema in the data warehouse.

Naming Convention

The general rule to name a mart model file name in mart is to use the format: <type>_<topic>_<additional_context>.sql. For instance: dim_dt_account.sql or fct_paid_orders.sql or fct_paid_orders_daily.sql or agg_usage_region.sql

A few common types are :

dim_: Flags data which is used to describe an entity.
fct_: Flags data which is in the form of numeric facts observed during measurement events.
agg_: Flags data which is an aggregation of a lower grain fact_

The BI team might decide to use additional type. Those new type will need to be documented in this wiki page

Materialization

Make it a table. When the table gets too long to build in your dbt Jobs,
Build it incrementally. That is, layer the data on in chunks as it comes in.

Reserved name for DBT

Do not use names starting with the pefix specified below for your schema, table, view and materialized view that you might create outside of DBT:

Schema

- staging_
- intermediate_
- mart_

In DBT, the dbt_project.yml file (located at the root) stores information about the schema used during materialization. In PROD, those schema will also be materialized with a sufix _UAT if they are scheduled using the UAT jobs.

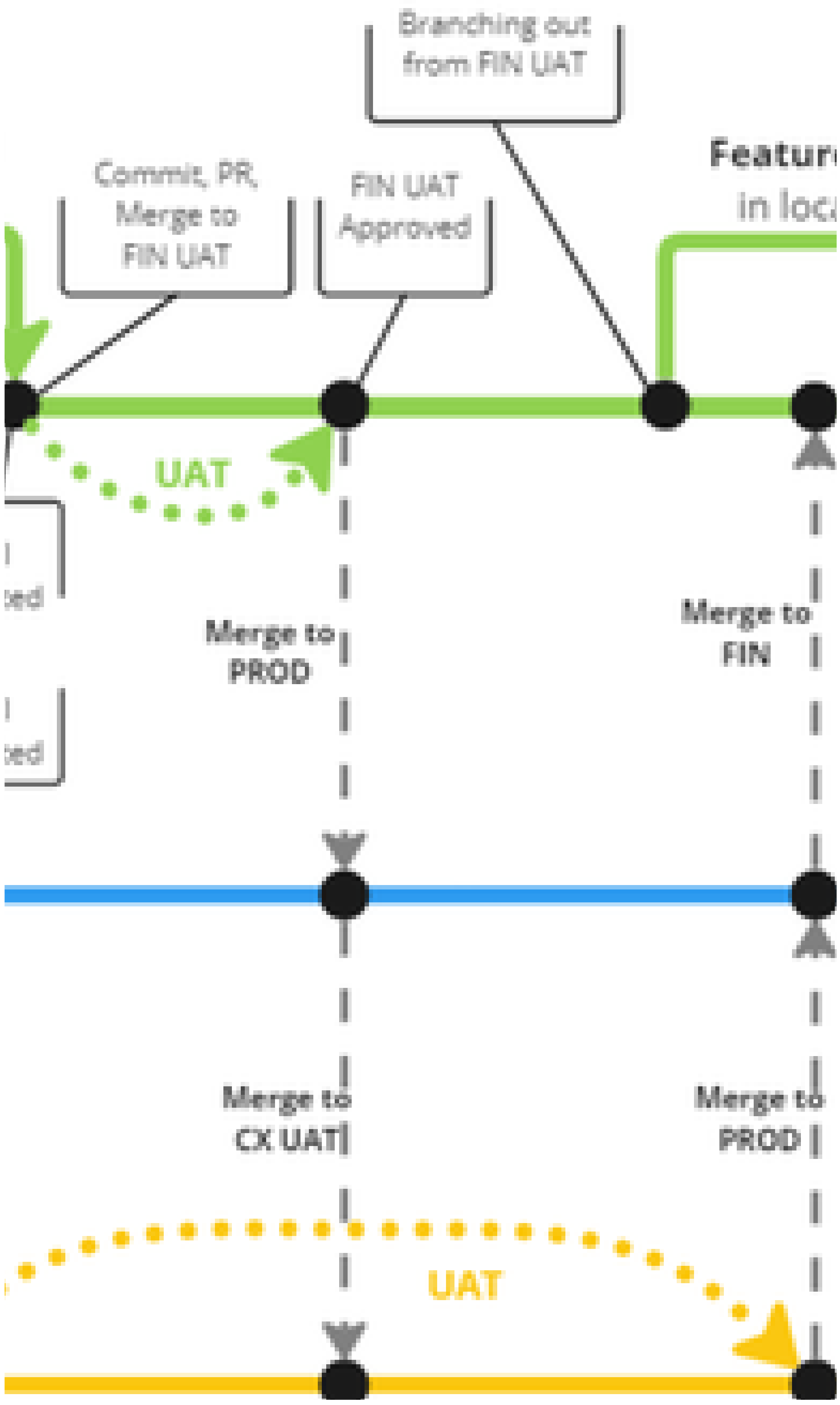
```
models:
  BusSysBI:
    # Applies to all files under models/example/
    staging:
      +materialized: view
      salesforce:
        +schema: staging_sfdc
```

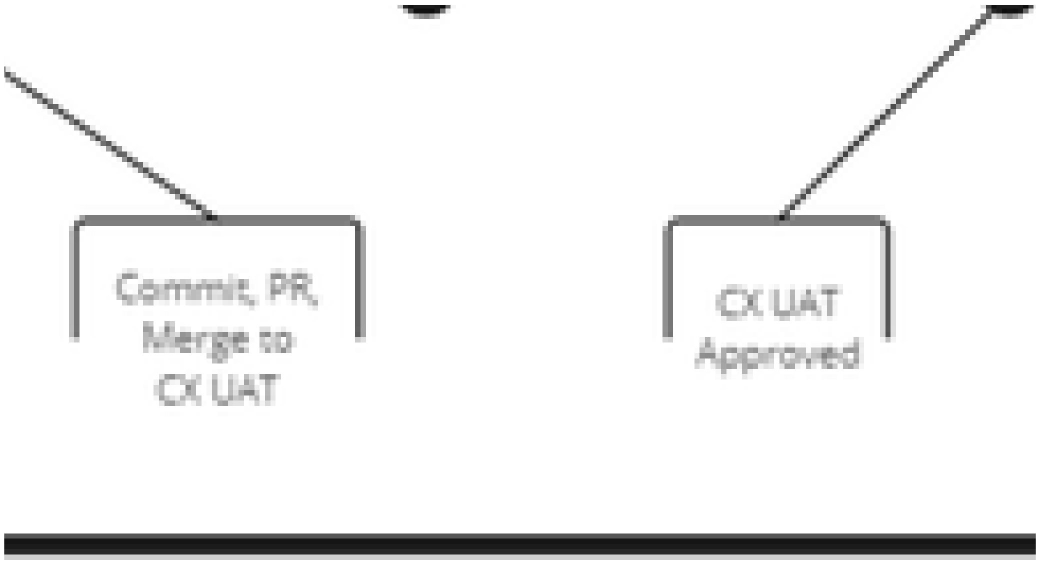
Table / View / Materialized View

- stg_
- int_

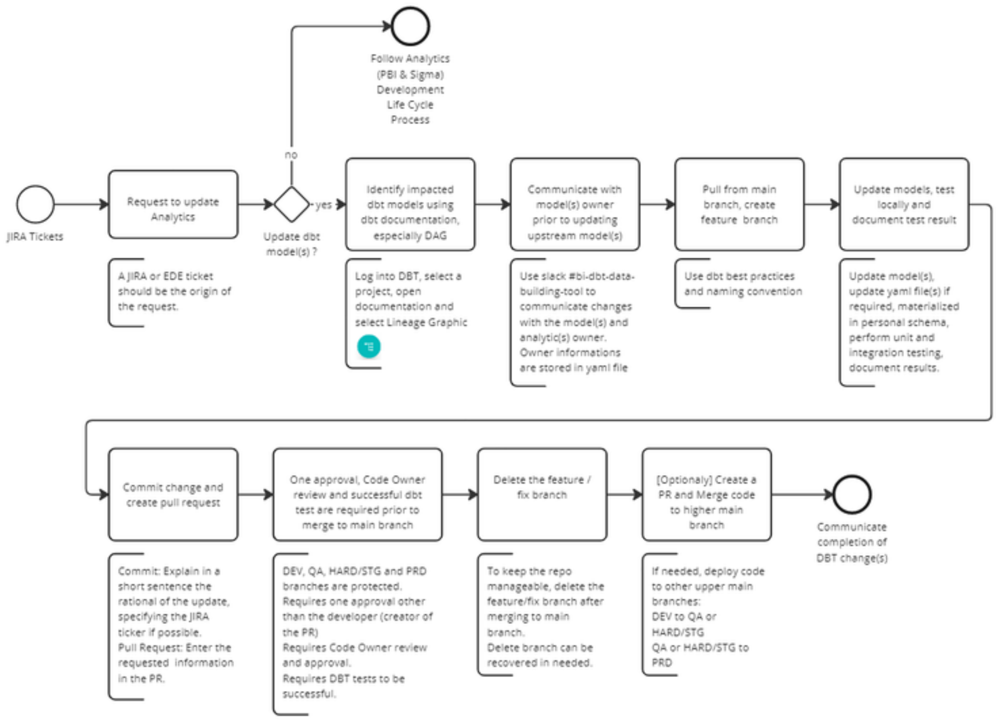
Development Life Cycle

Snowflake Branching and Release Management





GitHub Workflow - From Fix/Feature to merging in PROD Branch


Follow Analytics (PBI & Sigma) Development Life Cycle Process
no
JIRA Tickets
Request to update Analytics
A JIRA or EDE ticket should be the origin of the request.
Update dbt model(s) ?
yes
Identify impacted dbt models using dbt documentation, especially DAG
Log into DBT, select a project, open documentation and select Lineage Graphic
Communicate with model(s) owner prior to updating upstream model(s)
Use slack #bi-dbt-data-building-tool to communicate changes with the model(s) and analytic(s) owner. Owner informations are stored in yaml file
Pull from main branch, create feature branch
Use dbt best practices and naming convention
Update models, test locally and document test result
Update model(s), update yaml file(s) if required, materialized in personal schema, perform unit and integration testing, document results.
Commit change and create pull request
Commit: Explain in a short sentence the rational of the update, specifying the JIRA ticker if possible. Pull Request: Enter the requested information in the PR.
One approval, Code Owner review and successful dbt test are required prior to merge to main branch
DEV, QA, HARD/STG and PRD branches are protected. Requires one approval other than the developer (creator of the PR) Requires Code Owner review and approval. Requires DBT tests to be successful.
Delete the feature / fix branch
To keep the repo manageable, delete the feature/fix branch after merging to main branch. Delete branch can be recovered in needed.
[Optionaly] Create a PR and Merge code to higher main branch
If needed, deploy code to other upper main branches: DEV to QA or HARD/STG QA or HARD/STG to PRD
Communicate completion of DBT change(s)

Link to Miro Board.

Synchronizing Release Branches after Successful Code Release to PROD

Full Synchronization

After a successful merge to production, the following PowerShell script can be run to automatically create pull requests for all release branches, streamlining the post-merge process in software development. Each pull request will have to pass the CI job, potential conflict fixed and be approved.

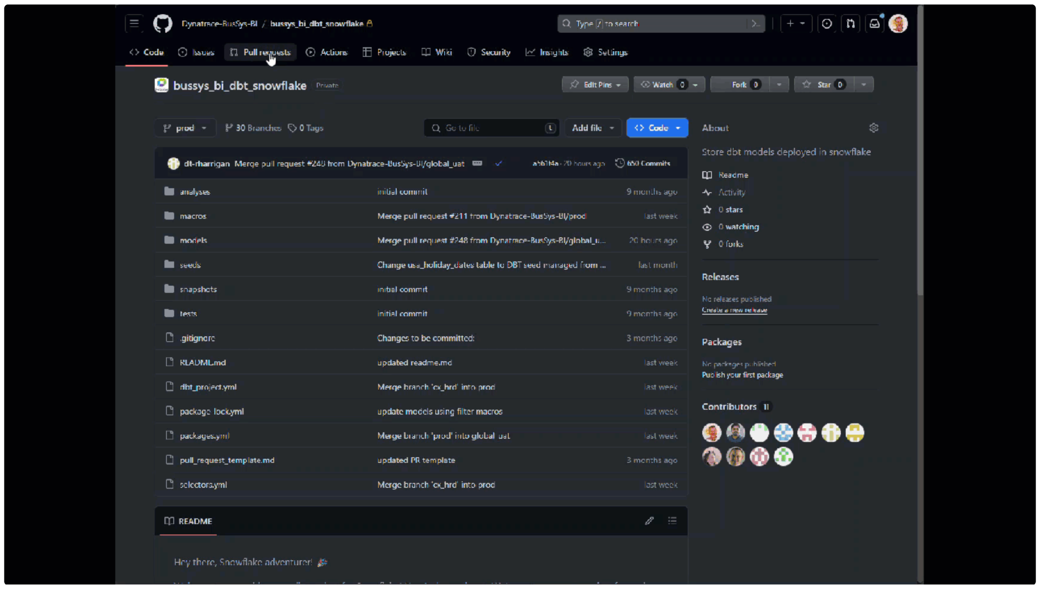
Pre-requisite:

- git CLI and GitHub CLI is locally installed.
- repository has been locally cloned.

```
# List of target branches - to update with addition of target branch
$branches = @("l2o_uat", "q2c_uat", "bussys_uat", "arr_uat", "con_uat", "fin_uat", "fpa_uat", "cx_uat", "cx_hrd"

# Iterate over each branch and create a PR from prod
foreach ($branch in $branches) {
    git checkout prod
    git pull origin prod
    git checkout $branch
    git pull origin $branch
    git checkout prod

    # Create the pull request
    gh pr create --base $branch --head prod --title "Merge from prod to $branch" --body "Automated PR to merge c
}

```

Manual Synchronization

After a successful merge to production, manually create a PR from PROD to a specific release branch using GitHub UI. The pull request will have to pass the CI job, potential conflict fixed and be approved.



Store dbt models deployed in snowflake

dt-rharrigan Merge pull request #248 from Dynatrace-BusSys-BI/global_uat a5b1f4a · 20 hours ago 650 Commits

analyses	initial commit	9 months ago
macros	Merge pull request #211 from Dynatrace-BusSys-BI/prod	last week
models	Merge pull request #248 from Dynatrace-BusSys-BI/global_u...	20 hours ago
seeds	Change usa_holiday_dates table to DBT seed managed from ...	last month
snapshots	initial commit	9 months ago
tests	initial commit	9 months ago
.gitignore	Changes to be committed:	3 months ago
README.md	updated readme.md	last week
dbt_project.yml	Merge branch 'cx_hrd' into prod	last week
package-lock.yml	update models using filter macros	last week
packages.yml	Merge branch 'prod' into global_uat	last week
pull_request_template.md	updated PR template	3 months ago
selectors.yml	Merge branch 'cx_hrd' into prod	last week

Hey there, Snowflake adventurer! 🎉

Git Branches

Git branches should:

Heroku

- be named as follows:
  - feature/<developer initial>/<envir>/name-of-feature/<JIRA Ticket #>
  - fix/<developer initial>/<envir>/name-of-fix/<JIRA Ticket #>
  - refactor/<developer initial>/<envir>/name-of-refactor

example: **feature/tk/dev/update-model-int-dt-account/AER-12345**

<envir> is the name of the environment you are developing into, which is similar of the dbt project you are working into. Values can be **dev**, **qa**, **hard** or **prod**

Snowflake

- be named as follows:
  - feature/<developer initial>/<project_envir>/name-of-feature/<JIRA Ticket #>
  - fix/<developer initial>/<project_envir>/name-of-fix/<JIRA Ticket #>
  - refactor/<developer initial>/<project_envir>/name-of-refactor

  example: **feature/tk/global_uat/update-model-int-dt-account/AER-12345**

  <project_envir> is the name of the project and environment you are developing into, which is similar of the dbt project you are working into. Values can be: **global_uat, cx_hrd, con_uat**…

Commits

Commits should:

- have a message in the imperative sense – a good way to frame this tense is to finish the sentence "this commit will ...". For example:
  - Add ARR models
  - Fix typo in sessions model description
  - Update schema to v2 schema syntax
- happen early and often! As soon as a piece of your code works, commit it! This means that if (/when), down the line, you introduce bad code, you can easily take your code back to the state it was in when it worked.

Pull requests

Pull requests should:

- tackle a functional grouping of work. While it may be tempting to (for example) build ARR models *and* add maintenance jobs in a single PR, these should be separate pieces of work.
- include a body that explains the context of the changes to the code, as well as what the code does. Useful things to include in a PR are:
  - Links to EDE or JIRA Ticket
  - Links to dbt docs that explain any new piece of functionality you have introduced
  - A screenshot of the DAG for the new models you have built
  - Explanation of any breaking changes
  - Any special instructions to merge this code, e.g. whether a full-refresh needs to be run, or any renamed models should be dropped.
- be opened with 48 hours for the reviewer to review
- be merged by its *author* when:
  - approval has been given by at least one collaborator
  - all tests have passed

Securing code using CODEOWNERS

Each DBT project can have a CODEOWNERS file located in .github folder. The CODEOWNERS file stores information about the individual responsible to review the code and is used by GitHub to auto assign reviewer.
Although security can be set at the model level, it is recommended to not be too granular to simplify maintenance.

Detailed information about how to setup CODEOWNERS file can be found here.
Note: Use email address instead of GitHub name.

Learn & Lunch Session

Action Items

Click here to access the action items document. Feel free to add any items that might be missing.

Recording

Date	Topics	Link to recording	Note
Mar 20, 2023	Introduction, team expectation and high level development from source to mart	https://dynatrace.zoom.us/rec/share/fZWVh -8LIRQyH1we34Wu8JcslOc_IOeUtGg - T283OhvuE1LnfbRCSHhFGkPKKoWg.bD XRrO9N8FPb6WgZ Passcode: %#8BNh9*	The team's expectations for the future sessions are: • Review automation test, source and target models, Unit testing with dbt, comparing data set, QA / UAT, code review before and after • Better understand best practices and naming convention. • How to leverage dbt IDE like dbeaver ? • What is the Impact on existing development life cycle (How it will change day to day development) ? • How can we better transform data ? • How to set up project, connection, job scheduling, CI/CD • How we plan to provide dbt access to business user ? • What are the other dbt capabilities ? • Can I do Exploratory Data Analysis with dbt?
Mar 21, 2023	Versioning with Git & GitHub.	https://dynatrace.zoom.us/rec/share/p6YT3 wdE9YKFNQ uX4xswq38Mj9Y00avtJnNeebsG6uLDu41 8fcAY97CpZUV 4Z6z.lHWltC3wxjODAlBE Passcode: wep?07B@	Action items are recorded in this document.
Mar 22, 2023	Building and deploying staging.	https://dynatrace.zoom.us/rec/share/hoSdK 7zgMlWUa3 7gLSFGbRK35DTwao6splKAwZSjgbEFNN Pt86OHE6TRsC1 LERTU.j39jpdboTd90CHqb Passcode: sJT^hp1C	No action item following this session.
Mar 23, 2023	Building and deploying intermediate models.	https://dynatrace.zoom.us/rec/share/JZdW CSGrf1eovyu wEGbsd3CYBQ073xYFg6iEMMluUoghTC bVvmOgrEVvB6p yopgx.9De7c5eE0aVWw1_8 Passcode: d4teYzD!	New actions items recorded in this document.

Mar 27, 2023	Automation Test.	https://dynatrace.zoom.us/rec/share/lCr8kIBuTRFgUXPQALwrFWjAmtBoiWOfi5X7aPcgpinKoLNO327fm6CDPYkdiYW.v8AVF8nEUOLA2MKN?startTime=1679932975000 Passcode: 4NN9^ny*	No action item following this session.
Mar 28, 2023	Securing repo, branches and code	https://dynatrace.zoom.us/rec/share/Z0B7UKFlq7LuonQ8YML1H0aMRurBAZYH45WzMstrzCBY0uav9B0ABzXjg-SG447.00cfthPmw35U-_4s Passcode:&*X=ik5&	New actions item recorded in this document.
Apr 3, 2023	dbt presentation to Data HQ members, especially automation testing	https://dynatrace.zoom.us/rec/share/NJ2meikwDiV-Fv8kDn_WoCiZWDteJ04IxQIvPIad14-Om8akSQNe5T9e_Mnun1Jb.hVxD-oSwHZe2sbwR Passcode: Q+%b.9Ga	
Apr 19, 2023	dbt code deployment from HARD to PROD-UAT and to PROD, and deployment release	https://dynatrace.zoom.us/rec/share/MjDmWFNkwY_xEWFKITN7S5kmwkFLQecKXT2BEXiVtqSd3swIcWq2hPfO5jvub-sm.BKkmkH9ICkH7KVRY?startTime=1681923800000 Passcode: J2f6KM+^	

Related articles

- dbt (data building tool) - Dynatrace Best Practices
- Fivetran - Extract Load Transform App
- dbt (data building tool) - First Time Login