

Managing your Analytics lifecycle in Qlik Sense Enterprise SaaS

Applying SDLC concepts to your Qlik Sense Enterprise SaaS
Tenant

v1.0

Leigh Kennedy
Master Principal Enterprise Architect - GEAR

TABLE OF CONTENTS

Introduction	3
SDLC Processes	3
Our vision for Qlik Cloud	3
Managing users in the Qlik Cloud platform	4
Your identity provider	4
Assigning system roles to identity provider groups	5
Assigning license entitlements to users	5
Assigning access to spaces and applications	5
Designing a groups framework for your SaaS SDLC	6
An SDLC environment for Qlik Application development	9
Our SDLC environment	9
SaaS and relative paths	10
Publish with replace	10
Alternative SDLC models	14
Assessing your applications	16
Data Profiling	16
Impact Analysis	17
Performance Evaluation	18
Tenants and your Software development lifecycle	19
Building context aware applications in SaaS	21
Integrating Qlik Sense enterprise SaaS with SDLC tooling	23
Building and unbuilding with Qlik CLI	23
Qlik app build – putting an app back together again	25
Encouraging re-use with qlik app build	26
Managing your Media library	28
Working with GitHub	30

Appendix A: <code>getSpaceDetails</code>	33
Appendix B: <code>appbuild.sh</code>	36
Appendix C: <code>appcreate.sh</code>	37
Appendix D: <code>distributeimage.sh</code>	38

Introduction

The majority of customers use some form of software development lifecycle in their organisations. SDLC is a process for planning, creating, testing, and deploying an information system¹.

In the context of data and analytics, a robust SDLC is important to follow, and this document aims to show techniques that can be used to incorporate SDLC processes into a Qlik Sense Enterprise SaaS environment.

SDLC Processes

While there are many different approaches to Systems development lifecycles such as agile or Waterfall, there are several common processes that an SDLC always needs to cover. This document will focus on the technical processes that need to occur within or interacting with a Qlik Sense Enterprise SaaS tenant as part of a Systems development lifecycle. The aim of this document is not to provide a strict SDLC process for customers to follow, rather it aims to provide examples of how SDLC processes could work in a Qlik Sense Enterprise SaaS environment.

Our vision for Qlik Cloud

Qlik Sense Enterprise SaaS runs on Qlik Cloud, Qlik's SaaS platform for our customer's data analytics and data integration needs. We see customers being able to manage their entire data landscape and are continually expanding our cloud offerings. We have introduced new services such as Hybrid Data Delivery and Qlik App Automation in 2021, as well as adding many new features to Qlik Sense Enterprise SaaS. While there are many other areas where Qlik Cloud can be integrated into your SDLC, this document will focus on Qlik Sense Enterprise SaaS.

Managing users in the Qlik Cloud platform

Qlik Sense Enterprise SaaS is part of the Qlik Cloud platform, and therefore several user management tasks are done at the platform level. These tasks include:

- Assignment of license entitlements
- Assignment of System roles
- Assigning and changing ownership of spaces and apps

Assigning access to spaces and the level of access to those spaces is specific to Qlik Sense Enterprise SaaS and therefore is managed within the Hub environment.

In this section we will discuss recommended techniques for managing your users in a Qlik Sense Enterprise SaaS environment.

Your identity provider

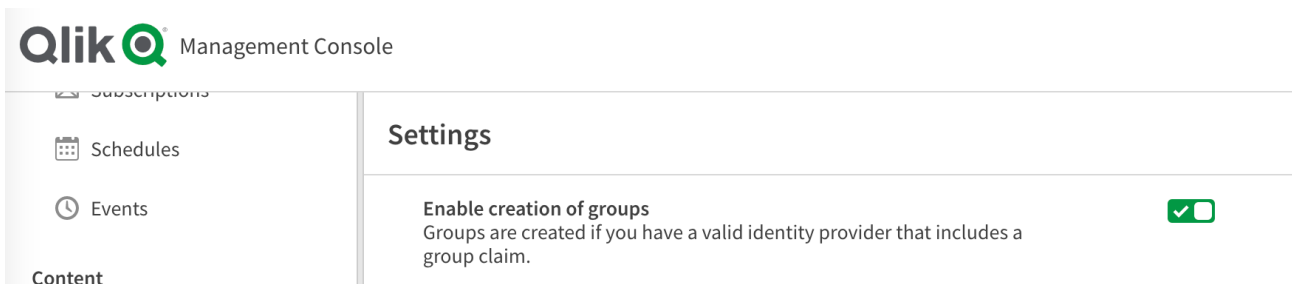
When customers set up their SaaS tenant, they have a choice of using Qlik Account based identity or using their own identity provider. Qlik provides this choice to ease the transition to SaaS for customers allowing them to get up and running Quickly.

However, for most customers using your own identity provider is recommended and provides much greater control and flexibility over your SaaS environment than using Qlik Account will. Some of the benefits of using your own identity provider include:

- Ability to enforce security policies such as password length and two factor authentication
- Ability to federate multiple identity sources (e.g. Active directory & google accounts)
- Ability to audit user logins and disable users in one place
- Ability to use groups for the purposes of assignment and authorization.

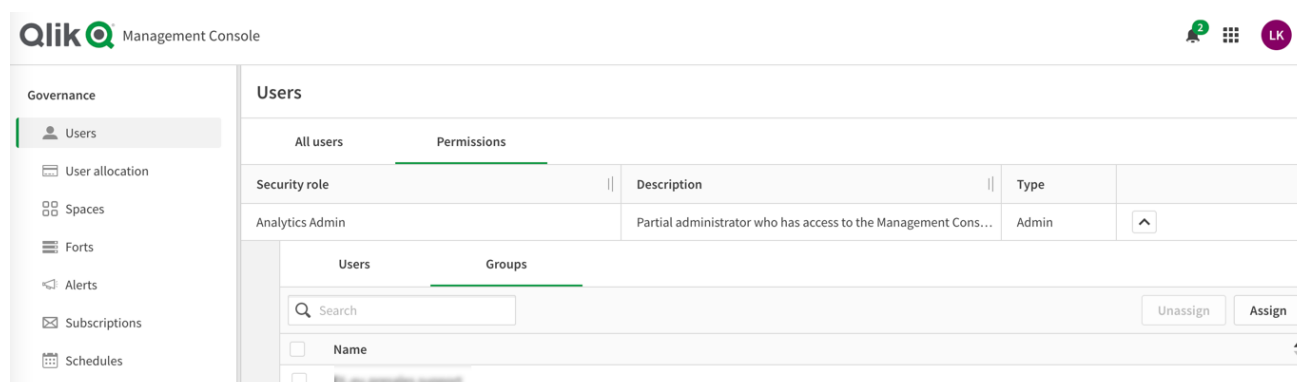
It is this last point which is of most relevance to this section; having users in groups allows us to assign permissions to groups and manage this at the organisational level, rather than having to manage users one by one in Qlik Sense Enterprise SaaS. The techniques discussed in this section are dependent on groups to be implemented.

To be able to use groups in assignments, ensure you have selected 'Enable the creation of groups' in the Management Console:



Assigning system roles to identity provider groups

When you enter the Users section of the Management Console, you are presented with a list of users and have the ability to assign roles directly to those users here. However to manage those roles using groups, select the 'Permissions' tab. This will show the roles available and by expanding this you see tabs for users and groups. By selecting the 'Groups' tab, you can assign the role to one of more groups.



Assigning license entitlements to users

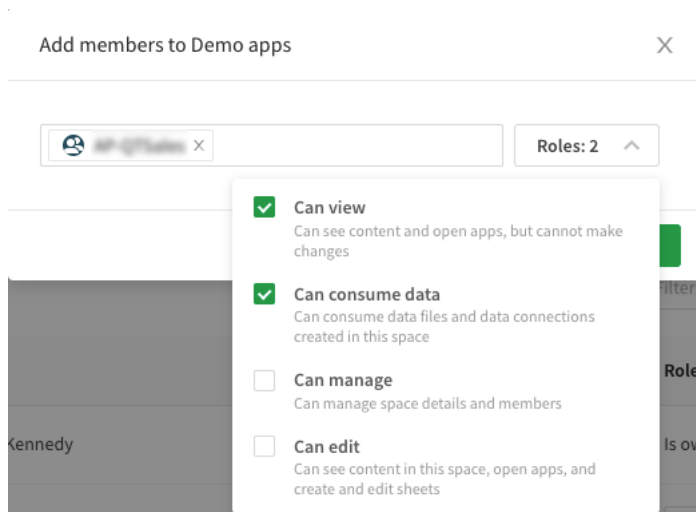
While the ability to assign license entitlements to groups is on Qlik's roadmap, this is not available today. However, this is something that customer can build themselves. The process will be different for each identity provider to get the list of users in each group but once this is obtained you can assign users by posting to the "v1/licenses/assignments/actions/add" endpoint.

An example of doing this has been shared by a colleague here:

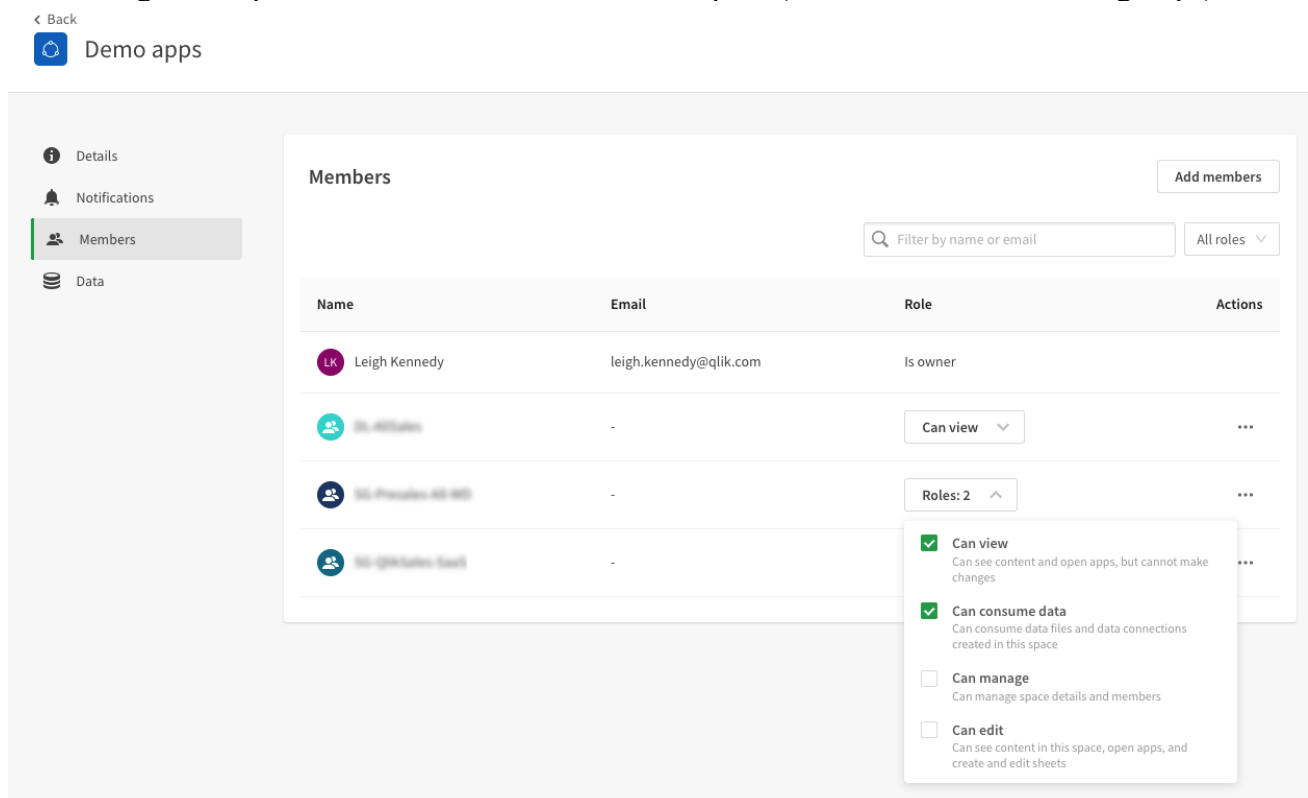
https://github.com/withdave/qlik/blob/master/snippets/qcs_create_user_assign_license.md

Assigning access to spaces and applications

When managing space access, we can assign space roles at a user or group level. Groups appear in the same list as users and the process of assigning space roles is the same.



Roles assigned to spaces then show as member of the space (blurred entries below are groups):



Designing a groups framework for your SaaS SDLC

The key to making groups integrate successfully with your SDLC is designing a group structure that fits the roles and responsibilities that your SDLC has. This will be set up in your identity provider rather than within Qlik's cloud platform, however this is a key prerequisite to making this work effectively for your organization.

So why would we want to use groups from our identity provider rather than managing this within Qlik? This is recommended for reasons such as organizational governance & centralized control, but also allows the same security controls to be used throughout an organisation's complete SDLC lifecycle, not just the part that occurs in Qlik Sense Enterprise SaaS. For example, a 'SalesDeveloper' group may grant permissions to extract data from the organisations CRM systems using that same group and the ability to save those extracts to the organisation's cloud storage location for Sales, which is then consumed by a Qlik Sense Application.

For example, if we create a groups framework around functions and business areas we might have:

Functional groups	Consumer	Developer	Admin
Sales	SalesConsumer	SalesDeveloper	SalesAdmin
Marketing	MarketingConsumer	MarketingDeveloper	MarketingAdmin
Shared	SharedConsumer	SharedDeveloper	SharedAdmin

We would then map this framework to space roles. This varies between shared and managed spaces. For Shared spaces we have:

Role assignment	Consumer	Developer	Admin
Sales	SalesConsumer = <i>'Can view'</i>	SalesDeveloper = <i>'Can edit'</i>	SalesAdmin = <i>'Can manage'</i>
Marketing	MarketingConsumer = <i>'Can view'</i>	MarketingDeveloper = <i>'Can edit'</i>	MarketingAdmin = <i>'Can manage'</i>
Shared	SharedConsumer = <i>'Can view'</i>	SharedDeveloper = <i>'Can edit'</i>	SharedAdmin = <i>'Can manage'</i>

And for managed spaces we have:

Role assignment	Consumer	Developer	Admin
Sales	SalesConsumer = <i>'Can contribute'</i>	SalesDeveloper = <i>'Can publish' & 'Can contribute'</i>	SalesAdmin = <i>'Can manage'</i>
Marketing	MarketingConsumer = <i>'Can contribute'</i>	MarketingDeveloper = <i>'Can publish' & 'Can contribute'</i>	MarketingAdmin = <i>'Can manage'</i>
Shared	SharedConsumer = <i>'Can contribute'</i>	SharedDeveloper = <i>'Can publish' & 'Can contribute'</i>	SharedAdmin = <i>'Can manage'</i>

This is just a simplified example, and it is likely some organisations will have more than the three functional roles shown here and many more business areas. But the principal is that we can design a framework to allow us to manage our Qlik Sense Enterprise SaaS environment from our identity provider.

It's worth remembering that systems roles may give users the ability to circumvent these controls (such as adding or modifying the space permissions, or changing ownership of apps and spaces). Therefore to supplement the governance provided by groups, it's recommended to integrate Qlik's event logging into your organisations auditing process.

An SDLC environment for Qlik Application development

A common scenario Qlik sees with our Enterprise customers is a multi-tier environment consisting of at a minimum Development/Test & Production. Many also have Sandbox environments for prototyping and data Science use-cases and some also contain Regression environments where bug fixes can be made in parallel with future development efforts. We will cover these concepts and how they would be used with Qlik Sense Enterprise SaaS in our hypothetical company here.

Our SDLC environment

Our environment consists of 5 tiers:

1. **Sandbox** – Developers can use their personal space as a sandbox where they can prototype, while data scientists are able to use the personal space to explore new projects.
2. **Development** – Development is a shared space used by development teams to collaboratively develop an application, or to take a prototype application duplicated from a user's sandbox and prepare it for release.
3. **Test** – This is a Managed space where application testers validate the application is ready for release to production.
4. **Production** – This is a Managed space where users can consume production applications and create self-service sheets on top of the governed data model.
5. **Regression** – This is shared space where bug fixes can be made to production in parallel with future development activities. Bug fixes made here need to be merged into the current development stream.

While in this example we use a single space for each tier, customers may have multiple spaces in each tier for different business areas or divisions.

There are two key features in Qlik Sense Enterprise SaaS that allow us to build out software development lifecycle like this:

Space Types

Qlik Sense Enterprise SaaS environments consist of 3 types of spaces:

Personal Space: Your personal space is your own private work area in the cloud hub. You cannot share apps from your personal space and other users cannot collaborate with you

Shared Space: Shared spaces are used to develop apps collaboratively and share them with other users in the space. A team might have a shared space for the private development and consumption of their own apps.

Managed Space: Managed spaces are used for providing governed access to apps with strict access control both for the app and the app data.

- Relative paths
- Publish with replace

SaaS and relative paths

When accessing data sources in Qlik Sense enterprise SaaS there are 3 possible types of locations the data connection could reference:

- The user's personal space, e.g. **[lib://SQL/MyTable]**
- A specific named space, different to where the application resides e.g. **[lib://Production:SQL/MyTable]**
- The same space that the application resides in, e.g. **[lib://:SQL/MyTable]**
(Note the ':' before the data source name!)

When we want to use relative paths we will specify a colon before the connection name, but with no space name before this. It is the ability to use relative paths that is a key feature which lets users run various SDLC stages in the same SaaS tenant. An app set up this name would automatically switch from the development to test connection when published to test, and then switch to the production connection when published to prod.

Publish with replace

The second feature that we need for this is publish with replace. This ensures that when publishing an application to production it replaces the application while ensuring any community content such as user sheets or bookmarks are preserved. This capability allows for lifecycles with multiple paths to production, such as maintaining a development and bug fix stream in parallel.

When using the publish capability, applications are generally published without data and the application is reloaded after publication to ensure the users see the appropriate data for their function. This may not always be required for testing but generally will be required for production.

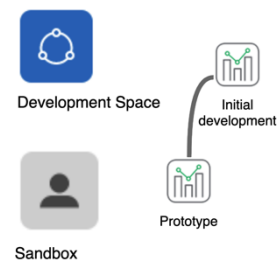
Using a SDLC in Qlik Sense Enterprise SaaS

The following legend applies to the diagrams in this section:

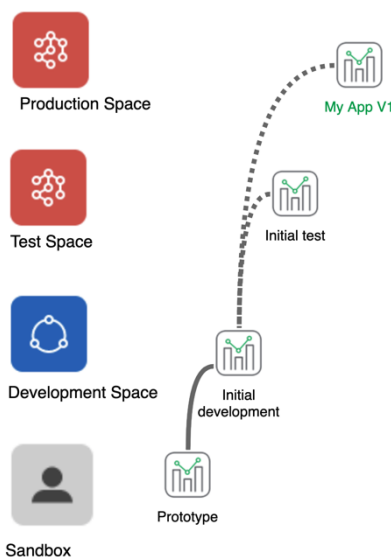
LEGEND	
Solid line	= duplicate
Dotted line	= publish
Dotted arrow	= merge changes

Let's look at some scenarios in our customer environment that we described earlier. This is by no means the only approach to an SLDC implementation with Qlik Sense Enterprise SaaS, but one that meets many of our customers needs. The goal here is to illustrate what is possible. This example can and should be adapted as is required to meet your organisations needs.

A prototype that a developer has built is duplicated into development so their team can collaborate on the application. The team makes changes as needed using relative connections to the development data sources.

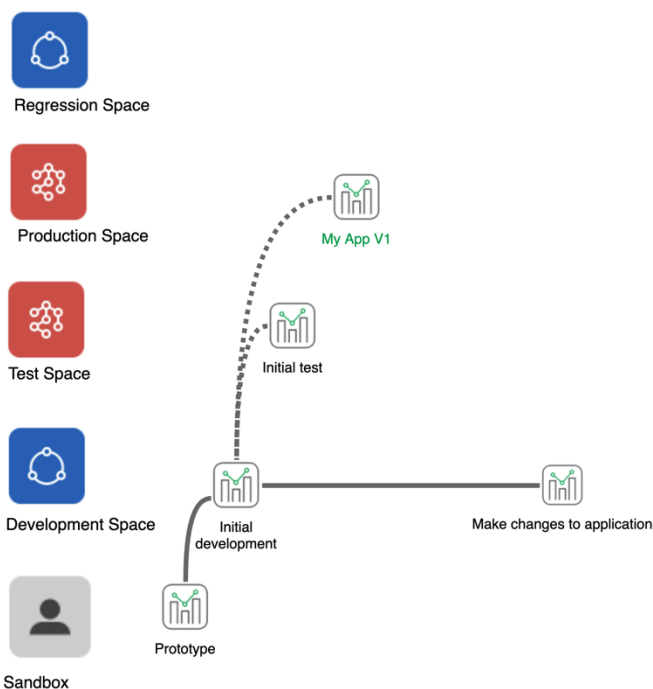


Once the development is completed and ready for testing, the application is published to the Test managed space without data. The application will then be reloaded ready for testing. Here the testers will review the application and if necessary, inform the developers of problems who will address these issues and re-publish for further testing.



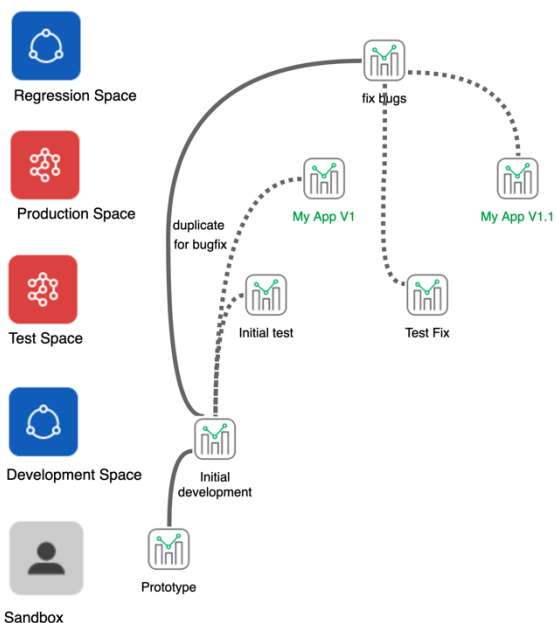
When testing is successfully completed, the application can then be published to production. Publish without data would be used to ensure production users and not exposed to test data and the application would be reloaded in production after publishing. The users would be informed the release is complete and given access to the application. The users would be able to use the application in production including creating their own personal sheets and content if they have the appropriate access levels to do so.

Next, the developers wish to work on an updated version of the application. They start by duplicating the prior version of the application. The reason for this step, rather than simply making changes to the application in place, is in case there is a need to make a bug fix in production while development is ongoing for the next version.



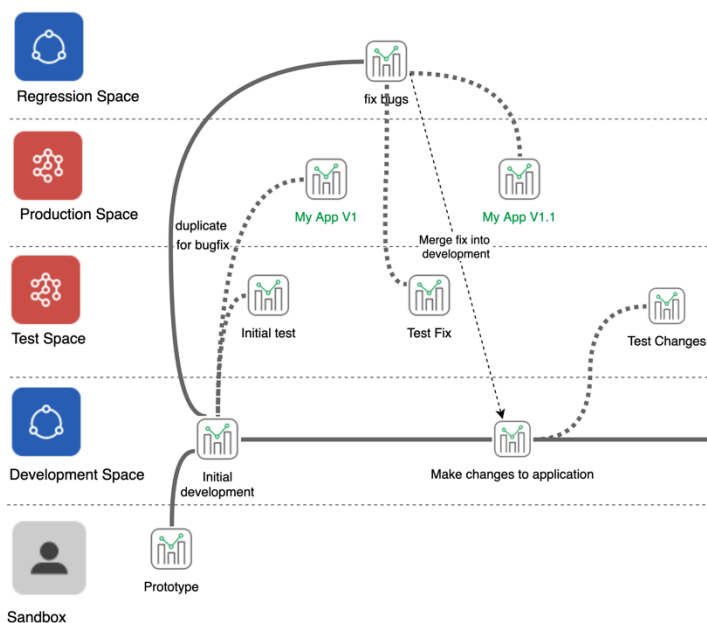
As it turns out a bug is found in production. To work on this without impacting ongoing development, the version of the application that was published to production is duplicated into the regression space.

The developers are able to address the bug and then publish the fix to the test space using the publish with replace functionality. The tester re-test the application here and once testing is completed and it's confirmed that the bug is fixed, the developers then republish the



fixed app to production using publish and replace. After reloading the application, the users are able to access the fixed app. Their personal sheets and other content are preserved.

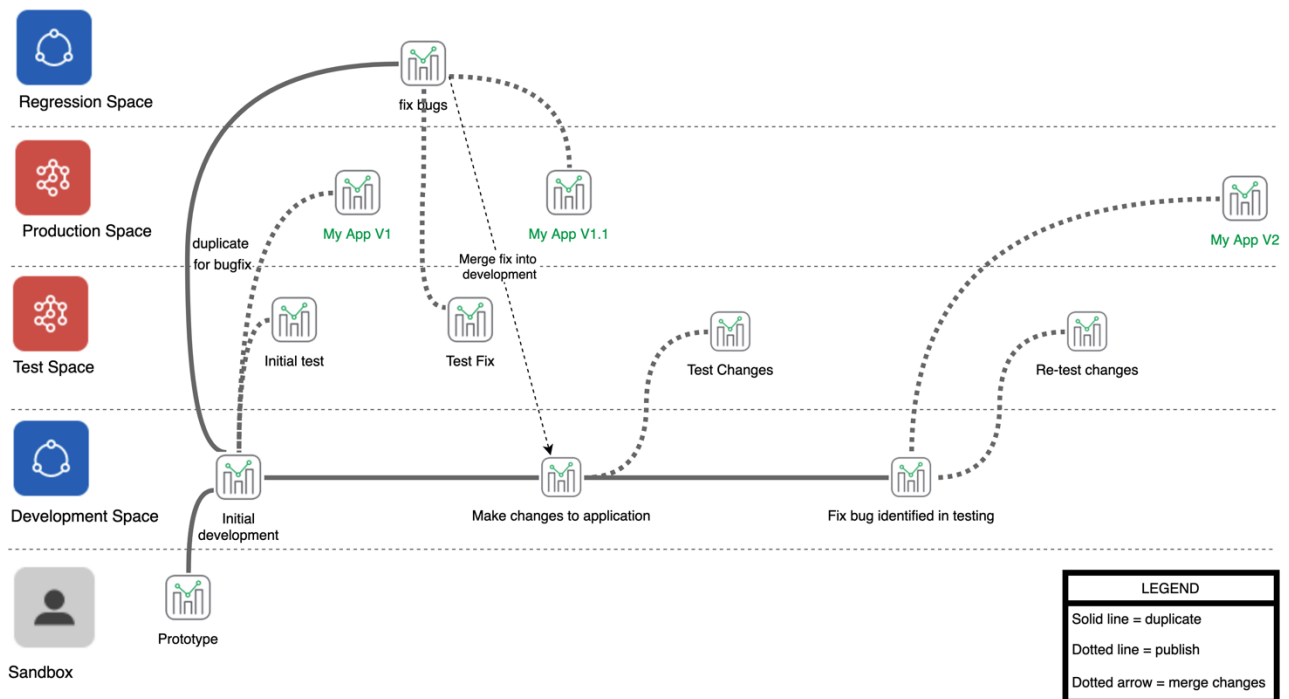
The developers are now ready to continue their development. First, they merge the changes required for the production fix into their development



application. They continue working on this new version of the application until it is ready to test. When it is ready to test they publish to the Test space using publish with replace and reload the application in test. They then hand it over to the testing team who evaluate the changes. In this case the testers find and issue with the application and inform the developers it is not ready to release to prod.

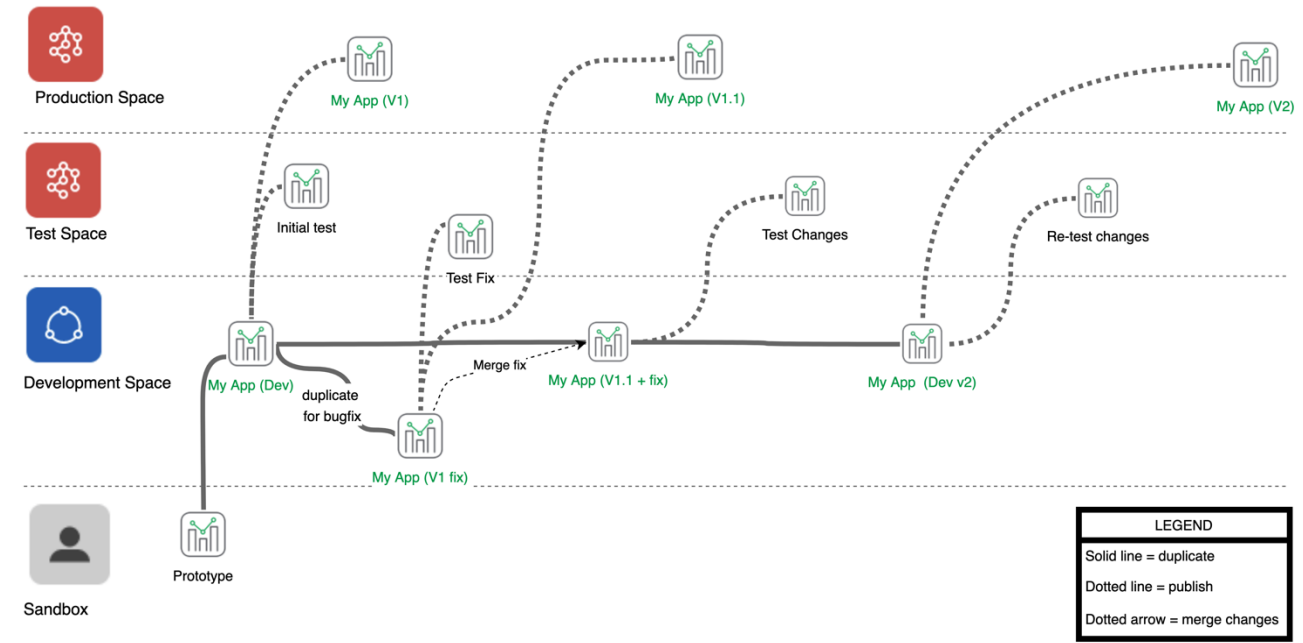
The developers address the issue raised by the testers and the publish to test again using publish with replace. After reloading the testers re-test and this time approve the changes. The developers then republish the application to production using publish with replace. After reloading, the users can now access the updated application. Their community sheets and content is preserved.

So, putting it all together this diagram shows the full lifecycle for this customer:

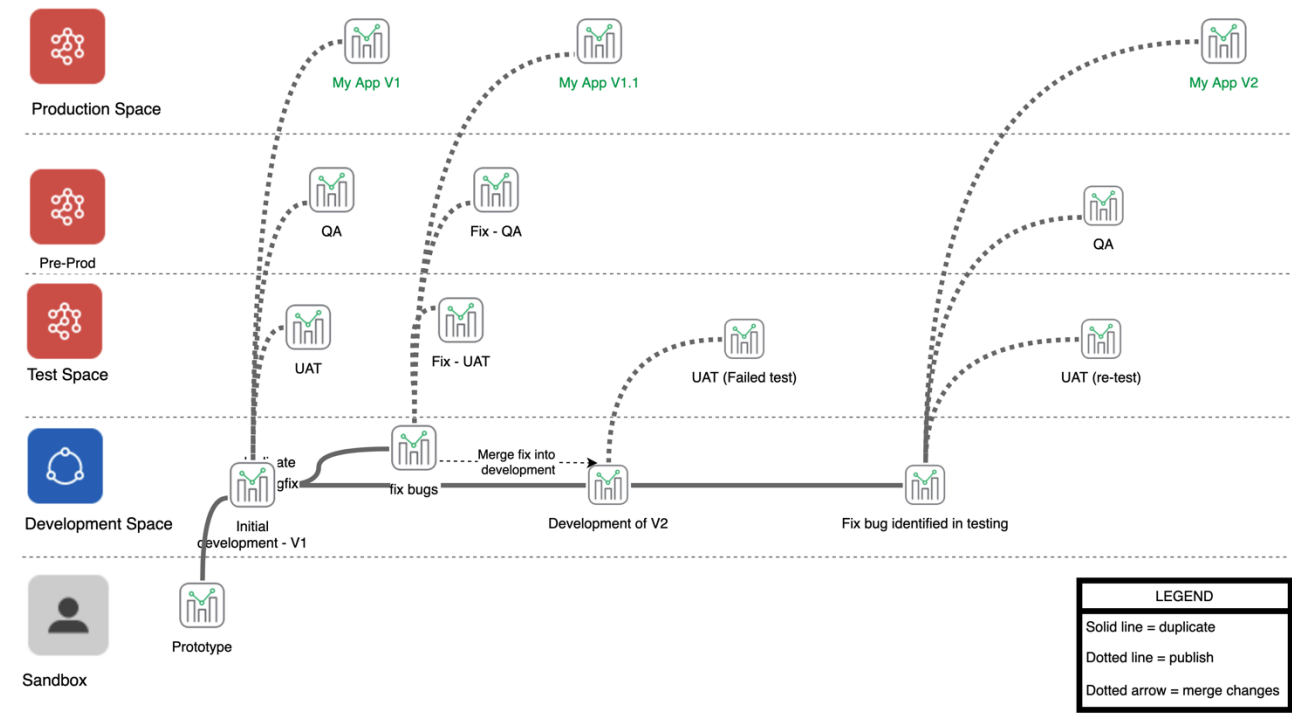


Alternative SDLC models

Many customers manage regression fixes in the same development stream as regular development by using naming conventions. As we see in this example, during the period while a fix is being implemented this needs to be coordinated so testers know what they are testing.



And a third model we see involves model we see customers using involves separating the UAT and QA testing phases into separate phases.



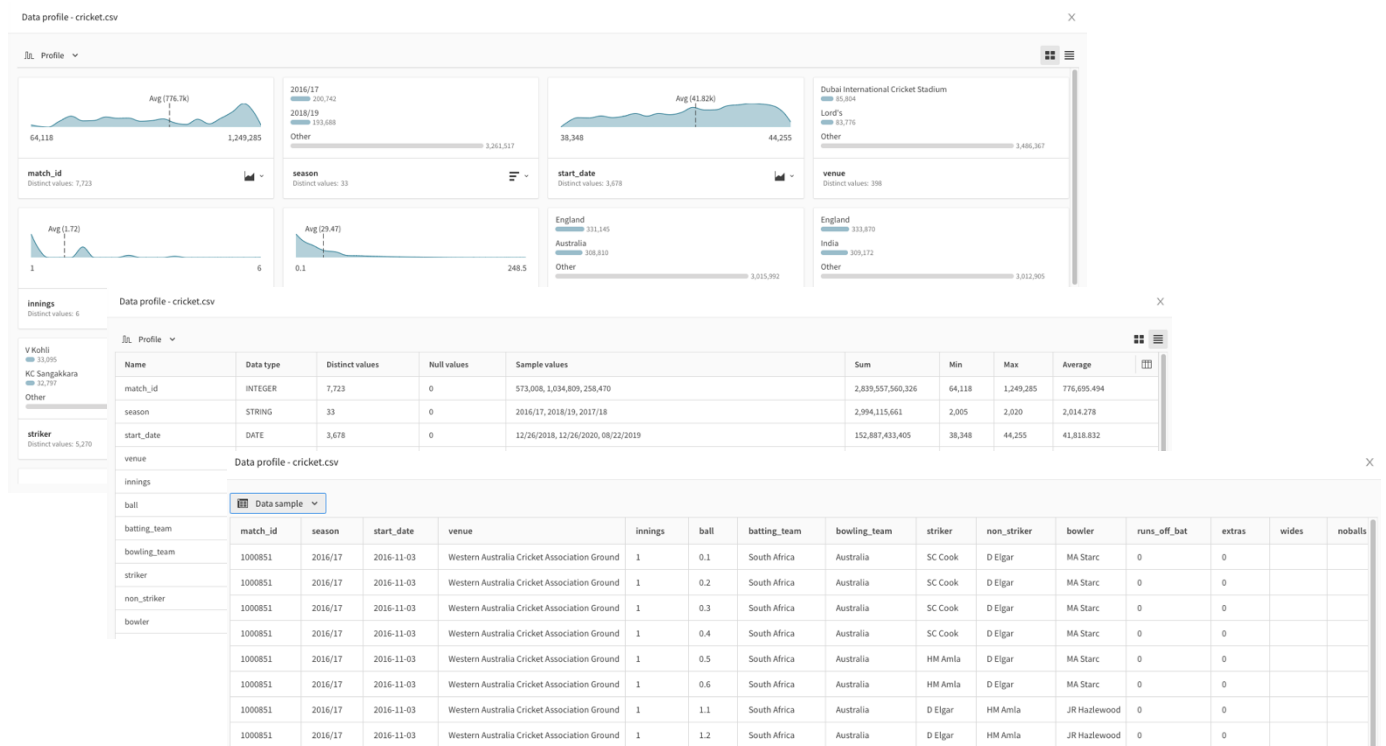
As we stated earlier these are examples and there is no one right way of doing this. We would encourage customers to adapt this to meet how their SDLC processes work in their organization.

Assessing your applications

Qlik Sense Enterprise SaaS provides a number of features to assist in developing and maintaining your Qlik Sense applications. Whether you are building a new application or maintaining an existing one, these tools will assist in ensuring your application is optimized and well understood. While they do not replace the need for formal code reviews and assessments, using these tools will enhance and accelerate your SDLC processes.

Data Profiling

Often when building applications in Qlik Sense we do not have the metadata we need to determine how best to use a particular data set. To assist with this, we have introduced data profiling into Qlik Sense Enterprise SaaS. By profiling our data we can see Sample data, whether a field contains nulls, distribution of data and for numerical data aggregation information. This information is useful in determining measures and dimensions that can be used in your applications.

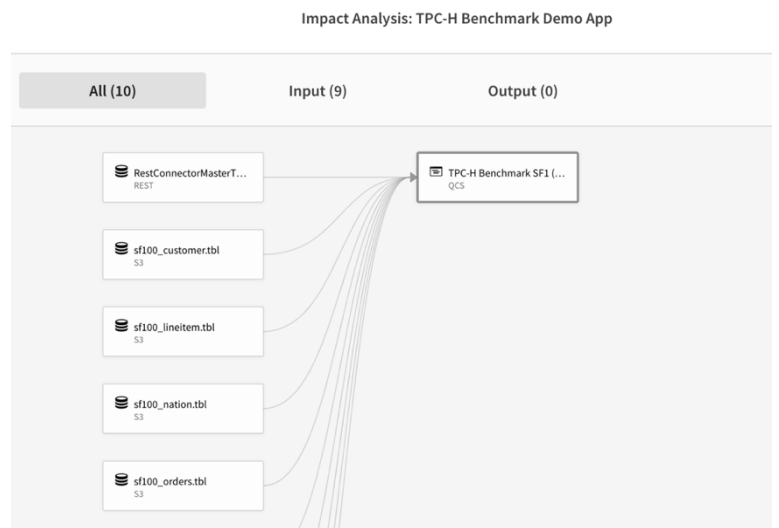


Impact Analysis

Impact analysis shows the flow of data into and out of apps and datasets in a lineage graph. This visual representation of your application assists in understanding the lineage and dependencies your application has.

For applications that generate QVDs which are then subsequently used to create another application we can see these

indirect dependencies. Thus if we are making a change to a data source we can assess the downstream implications and which other applications may require changes.



Performance Evaluation

Another feature which Qlik provides to assist in reviewing your applications is the Performance Evaluation feature. This looks at your application and highlights key areas which you can focus on to improve performance and/or reduce the resource consumption of your application. It will identify which objects are causing performance problems, so you know where to focus your efforts.

Evaluation details ⓘ ×

Overview
Details

App size 64 KB

File size 524 KB

Number of rows 717 rows

Public sheets in app 5 sheets

Public objects in app 46 objects

Overview
Details

✓ No objects exhibited problems caching ⓘ

✓ No objects exhibited single-threaded performance ⓘ

Top 5 objects with the largest uncached load time ^

Object	Time
Filter Pane	0.1 s
hjemZp	< 0.05 s
Field Cardinality	< 0.05 s
TRyPdE	< 0.05 s
Field RAM Footprint (MB)	< 0.05 s

Top 5 objects with the largest cached load time v

Top 5 tables with the largest memory allocation v

Top 5 fields with the largest memory allocation v

Help
Download log
Done

We can also compare against past performance evaluations. This is extremely useful which trying to determine the impact of a change or to troubleshoot performance problems.

Compare evaluation results ⓘ ×

Overview
Details

	Oct 18, 2021 8:55 AM	Oct 18, 2021 9:10 AM	Change
Status	Complete	Complete	-
App size	60 KB	64 KB	7 % ↑
File size	492 KB	524 KB	7 % ↑
Number of rows	677 rows	717 rows	6 % ↑
Public sheets in app	5 sheets	5 sheets	-
Public objects in app	46 objects	46 objects	-

Tenants and your Software development lifecycle

Sometimes our customers ask us about whether they need multiple tenants. Often this something they assume they will need and/or to mirror their on-premise deployment. And regularly it is to align with their software development lifecycle processes which are based on moving through environments. However, as you will have seen none of the SDLC models shown here are based on multiple tenants. This is because they are not required to meet these objectives. Qlik has designed the spaces concepts to provide the flexibility needed to implement these disciplines in a single SaaS tenant. However sometimes customers still have concerns. Here are some common concerns and whether we see a need for multiple tenants or not:

- **My development / testing could have a performance impact on my production apps:** In SaaS, Apps are assigned to an engine from a huge pool of engines based on available resources at the time they are opened. If an engine is very busy, other apps will be assigned to different engines. Our SaaS platform is handling thousands of customers and tens of thousands of applications in each region at any one time and is constantly scaling up and down to handle this.
- **I test new OS & Software patches in Development first before releasing to production:** Qlik tests all changes extensively before releasing them into our SaaS Platform. This involves extensive formal testing as well as releasing these changes to Qlik's internal users before they are released to our SaaS platform. When they are released to our SaaS platform they are monitored 24x7 and anomalies are investigated and if necessary backed out. We also are able to release changes to our SaaS platform discretely, so we are able to see the impact of individual changes. In the rare occurrences where a problem occurs, we have usually identified and rectified this before customers even become aware of it. If a customer did have separate tenants, they would be running the same versions anyway as releases are made to the platform, not the individual customers tenants.
- **I have a custom developed extension I need to test:** This is a case when a separate tenant would be beneficial. Customers who are building their own custom extensions could look at renaming the extension but this activity would likely be easier to manage with multiple tenants.
- **I have different sets of users in different locations around the world:** This is another case where multiple tenants are a valid option. Depending on where your users are they may see performance benefits from using a tenant located closer to them. The downside of this approach however is some users may need to log into multiple tenants, or alternatively you may need to duplicate some applications in multiple locations. An alternative approach to this would

be to deploy Qlik Forts in those regions to host specific apps, while keeping the majority of your apps in a single SaaS tenant.

- **I need multiple tenants for security reasons:** This is rarely necessary in our experience. The only users who have unrestricted access as tenant administrators and due to separation of duties, these users would rarely be involved in application development or business roles relating to the applications. All other users are restricted to permissions assigned at the space level, therefore access to dev/test/prod/etc. spaces would be managed the same as if there were multiple environments. Access to use data connections and data files can be granted without those users being able to modify the connections or data. And credentials can not be retrieved once entered so even users who are tenant administrators can not gain access to credentials.
- **I want to test tenant level configuration changes:** This may occur if you are planning to change your identity provider, SMTP server, or other external dependencies. It is worth speaking to Qlik about these needs as we may have roadmap items which will address these (for example multiple IDP support is planned) or be able to provide you with an alternative solution (such as a temporary tenant for one-off situations).

While we have tried to cover the most common scenarios we are asked about here there may be others you are uncertain about or you may just wish to validate whether your requirements are best suited to one or multiple tenants. In these cases Qlik's Architects are available to assist you with this.

Building context aware applications in SaaS

In a model where we have a single development, test & production space similar to what we showed earlier, it is easy to use relative connections in those spaces to ensure we are using the appropriate connection based on the SLDC stage.

However, in larger organisations, there is sometimes a need to break this down further by subject area or division, for example having *Sales_Development*, *Finance_Development*, *Shared_Development*, etc. This complicates the model in that an application in *Sales_Development* may need to access a data connection in *Shared_Development*. The only obvious way to achieve this is by hard-coding the space, however this breaks our model. What we ideally want is a way to determine which space our app is in, so we can then direct it to use *Shared_Development* in development, *Shared_Test* in test, etc.

While there is currently no direct function to do this, it turns out we can achieve this by using Qlik's APIs. APIs can be accessed from a load script using a rest data connection to our tenant. It is recommended to create this connection in a space developers can access with only the 'data consumer' role, as exposing the connections details is not necessary and could constitute a security risk.

First, we need to know what our application ID is. While not immediately obvious, the `DocumentName()` function in fact returns the ID of our application. With our application id we can query the `/api/v1/apps` endpoint to find out details about our application including the space ID for the applications space. We then in turn use `/api/v1/spaces` endpoint to get the name, type and other details on the space. We have written a set of subroutines to do this which are in **Appendix A: getSpaceDetails**.

So now we know the space we are in, we can dynamically build a connection or data-file path related to the space we are in. For example, assuming I have a set of file paths that should be used depending on whether I am in Dev, Test or Prod. I would be able to use the space information, combined with naming conventions to construct a path depending on the space I am in:

```
if '${vSpaceName}' like '*DEV*' then
  trace This is a Dev space;
  set vConn='lib://Development:DataFiles/';
elseif '${vSpaceName}' like '*TEST*' then
  trace This is a Test space;
  set vConn='lib://Test:DataFiles/';
elseif '${vSpaceName}' like '*PROD*' then
  trace This is a Prod space;
  set vConn='lib://Production:DataFiles/';
else
  trace Error: Non-standard space;
end if
```

Or let's say we have a library of subroutines we use in our apps. We may wish to load different versions depending on the space we are in. We create a function that determines the right script version based on where we are. In this case, dev uses the newest version:

```
sub getLibPath(vPath)
  if '${vSpaceName}' like '*DEV*' then
    trace This is a Dev space;
    set vPath='lib://common:DataFiles/library_1.2.QVS';
  elseif '${vSpaceName}' like '*TEST*' then
    trace This is a Test space;
    set vPath='lib://common:DataFiles/library_1.1.QVS';
  elseif '${vSpaceName}' like '*PROD*' then
    trace This is a Prod space;
    set vPath='lib://common:DataFiles/library_1.0.QVS';
  else
    trace Error: Non-standard space;
  end if
  trace $(vPath);
end sub;
```

The library has a trace statement at the beginning which prints it's version, so when we use this in our load script:

```
call getLibPath(vPath);
$(Must_Include=$(vPath));
```

We get:

```
This is a Dev space
lib://common:DataFiles/library_1.2.QVS
Loading Library Version 1.2
```

Now this is just an example and is dependent on your naming convention. For example, Production may be defined by not having a suffix at all, and you may prefer to use prefixes. It depends on what naming conventions work for your organization.

Integrating Qlik Sense enterprise SaaS with SDLC tooling

Caution: Under Construction!

This section relies on the use of experimental APIs which are highly likely to change in the near future. There are also some gaps in functionality so what is covered here is definitely not appropriate for all use-cases. Our goal is to show where we are going and our thought processes. We would advise caution before using this for business critical purposes.



Building and unbuilding with Qlik Cli

Unbuild: To dismantle or deconstruct (something previously built).

Qlik sense applications (i.e. QVF files) are binary objects. When integrating with SCM tools, binary files are a poor fit for several reasons:

- It's not possible to see the scope of the changes made
- It's not possible to compare to prior versions
- It's not possible to see which individual components that have (or have not changed)

To address this, we have added the app unbuild command to qlik cli:

```
qlik app unbuild -h
```

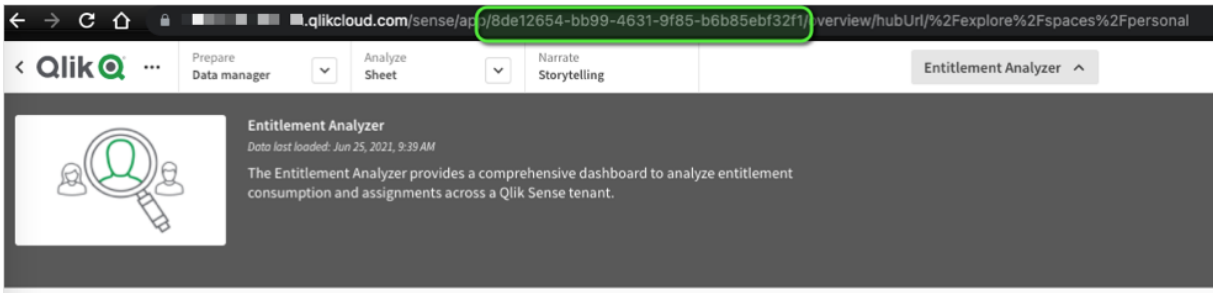
```
Extracts generic objects, dimensions, measures, variables, reload script and connections from an app in an engine into separate json and yaml files.
```

```
In addition to the resources from the app a config.yml configuration file is generated that binds them all together.
```

```
Passwords in the connection definitions can not be exported from the app and hence need to be handled manually.
```

```
Generic Object trees (e.g. Qlik Sense sheets) are exported as a full property tree which means that child objects are found inside the parent's json (the qChildren array).
```


To use the `unbuild` command on an app, we first need the app id. You can either get that from the url when the app is open, or by running `'qlik app ls'` from the cli.



Now we can unbuild our app. Qlik app unbuild takes the app id and optionally a directory name. It will generate its own name if not specified:

```
~/dev/code/QSEoCS/unbuild ➤ qlik app unbuild -a 8de12654-bb99-4631-9f85-b6b85ebf32f1 --dir entitlement-analyzer
"unbuild" command is experimental and it may change between releases
lkn@APAC-lkn ➤ ~/dev/code/QSEoCS/unbuild ➤ ls entitlement-analyzer/
app-properties.json  connections.yml  measures.json  objects          variables.json
config.yml           dimensions.json  myobjects.json script.qvs
```

We can see the various parts of the application have been written to disk. The `config.yml` provides a manifest of everything that has been extracted:

```
~/dev/code/QSEoCS/unbuild ➤ more entitlement-analyzer/config.yml
script: script.qvs
connections: connections.yml
dimensions: dimensions.json
measures: measures.json
objects: objects/*.json
variables: variables.json
app-properties: app-properties.json
entitlement-analyzer/config.yml (END)
```

It's worth reminding here that this is an experimental command and has some gaps. At the time of writing *Bookmarks*, *media* and *personal content* are excluded.

We are now able to check these files into our source code repository, review the load script against best practice, or any other actions we wish to take with the application code.

Qlik app build – putting an app back together again

While there are a lot of reasons we may wish to unbuild an application, many of those would be with the expectation we can rebuild the application later. We can do this with the 'qlik app build command'. Qlik app build does not create an app, it requires a pre-existing app. Therefore, we first create an app with the 'qlik app create' command and pass the app id for that app to 'qlik app build'.

The full syntax of 'qlik app build' is.

Shown here:

While it is possible to create application dependencies such as connections with this command, whether you want to do this really depends on your goals. If you are re-creating the application in the same

```
Usage:
  qlik app build [flags]

Examples:
  qlik app build
  qlik app build --connections ./myconnections.yml --script ./myscript.qvs

Flags:
  -a, --app string           Name or identifier of the app
  --app-properties string    Path to a json file containing the app properties
  --bookmarks string         A list of generic bookmark json paths
  --connections string       Path to a yml file containing the data connection definitions
  --dimensions string        A list of generic dimension json paths
  -h, --help                 help for build
  --limit int                Limit the number of rows to load
  --measures string          A list of generic measures json paths
  --no-reload                Do not run the reload script
  --no-save                  Do not save the app
  --objects string           A list of generic object json paths
  --script string            Path to a qvs file containing the app data reload script
  --silent                  Do not log reload output
  --variables string         A list of generic variable json paths
```

SaaS tenant, you probably won't want to do that as the connections already exist.

There are some differences between the build and unbuild commands. For example unbuild places each object in its own file. This is very useful in determining which objects have changed.

However, to build an app the command expects a json list. Therefore we need to assemble the various objects into a single file.

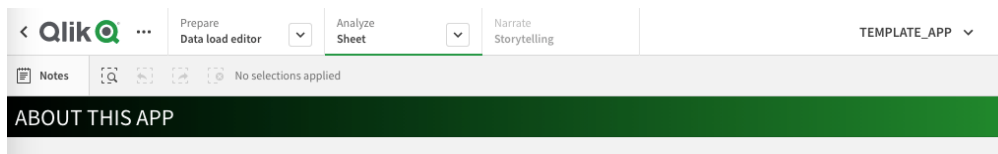
To simplify this I've created a script which builds an app based on the files created by 'qlik app unbuild'. Feel free to modify as needed. The script takes 1 parameter; the app directory (as created by qlik app unbuild) and the new app name. This script ignores connections as I'm assuming that is something we want to handle separately along with the creation of spaces (not covered in this article). It also uses the default behaviour of reloading the app after it is built. If this is not what you want, edit the script to add '--no-reload' to the build command. So to run this script against the unbuild we ran earlier (for the entitlement-analyzer), we would run:

```
./appbuild.sh entitlement-analyzer
```

Encouraging re-use with qlik app build

While we have just shown using qlik app build to recreate an existing app, it is possible to use this to create new apps also. One reason we may wish to do this is re-use. Consider that you develop a script library, default themes and standard variables you want to use in your apps. We can create a template app for this purpose:

```
itor [v] Analyze Sheet [v] Narrate Storytelling [v] TEMPLATE_APP [v]
- [?]
1 SET ThousandSep=',';
2 SET DecimalSep='.';
3 SET MoneyThousandSep=',';
4 SET MoneyDecimalSep='.';
5 SET MoneyFormat='$ ###0.00;- $ ###0.00';
6 SET TimeFormat='h:mm:ss TT';
7 SET DateFormat='M/D/YYYY';
8 SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';
9 SET FirstWeekDay=6;
10 SET BrokenWeeks=1;
11 SET ReferenceDay=0;
12 SET FirstMonthOfYear=1;
13 SET CollationLocale='en-US';
14 SET CreateSearchIndexOnReload=1;
15 SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
16 SET LongMonthNames='January;February;March;April;May;June;July;August;September;October;November;December';
17 SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
18 SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';
19 SET NumericalAbbreviation='3:k;6:M;9:G;12:T;15:P;18:E;21:Z;24:Y;-3:m;-6:μ;-9:n;-12:p;-15:f;-18:a;-21:z;-24:y';
20
21 //STANDARD INCLUDES
22 SET APP_OWNER="SAMPLE NAME";
23 SET APP_VERSION="1.0";
24 SET RELEASE_DATE="01-01-2021";
25 $(Include=lib://FunctionLibrary>DataFiles/library.qvs);
26
27 //CUSTOM INCLUDES
28
29 // INCLUDES_HERE
30
31 //END OF INCLUDES
32
```



Our Template app links to a library of subroutines we use as well as allows us to add custom includes when we build the app. We will modify our appbuild script to process those custom includes.

When our Template is ready, we extract it to a template directory, e.g:

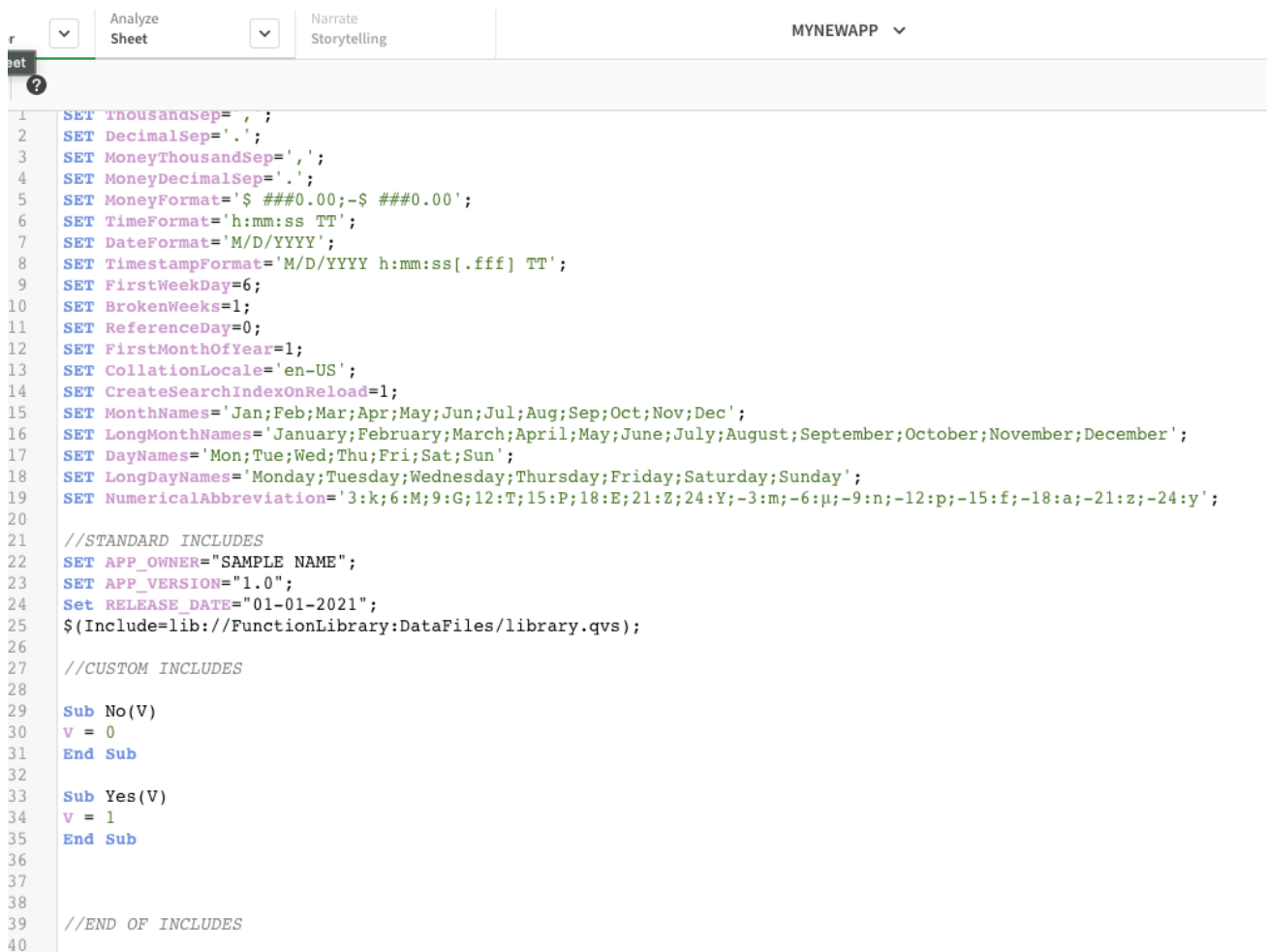
```
qlik app unbuild -app 1aa4da26-8ca3-47b9-9881-7bdb402c8696 -dir APP_TEMPLATE
```

We now have a template to use when creating new apps. Our new appcreate.sh script does the following:

1. Create A new app in our SaaS tenant
2. Replaces the line '//INCLUDES_HERE' with the contents of a file INCLUDES.QVS.
3. Uses the extracted template to build our app.

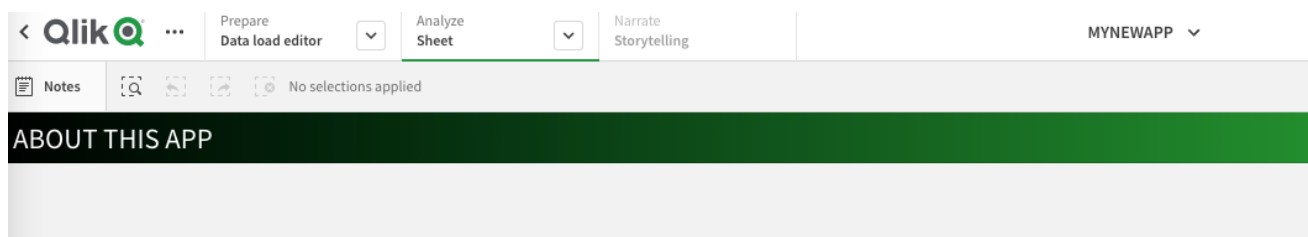
So If I create a new app with this script and open it up, We can see it has used our template as well as adding the custom includes;

```
/appcreate.sh MYNEWAPP
```



```
1 SET ThousandSep=',';
2 SET DecimalSep='.';
3 SET MoneyThousandSep=',';
4 SET MoneyDecimalSep='.';
5 SET MoneyFormat='$ ###0.00;-$ ###0.00';
6 SET TimeFormat='h:mm:ss TT';
7 SET DateFormat='M/D/YYYY';
8 SET TimestampFormat='M/D/YYYY h:mm:ss[.fff] TT';
9 SET FirstWeekDay=6;
10 SET BrokenWeeks=1;
11 SET ReferenceDay=0;
12 SET FirstMonthOfYear=1;
13 SET CollationLocale='en-US';
14 SET CreateSearchIndexOnReload=1;
15 SET MonthNames='Jan;Feb;Mar;Apr;May;Jun;Jul;Aug;Sep;Oct;Nov;Dec';
16 SET LongMonthNames='January;February;March;April;May;June;July;August;September;October;November;December';
17 SET DayNames='Mon;Tue;Wed;Thu;Fri;Sat;Sun';
18 SET LongDayNames='Monday;Tuesday;Wednesday;Thursday;Friday;Saturday;Sunday';
19 SET NumericalAbbreviation='3:k;6:M;9:G;12:T;15:P;18:E;21:Z;24:Y;-3:m;-6:μ;-9:n;-12:p;-15:f;-18:a;-21:z;-24:y';
20
21 //STANDARD INCLUDES
22 SET APP_OWNER="SAMPLE NAME";
23 SET APP_VERSION="1.0";
24 Set RELEASE_DATE="01-01-2021";
25 $(Include=lib://FunctionLibrary:DataFiles/library.qvs);
26
27 //CUSTOM INCLUDES
28
29 Sub No(V)
30 V = 0
31 End Sub
32
33 Sub Yes(V)
34 V = 1
35 End Sub
36
37
38
39 //END OF INCLUDES
40
```

And the theme and color changes from our template are also present:



Managing your Media library

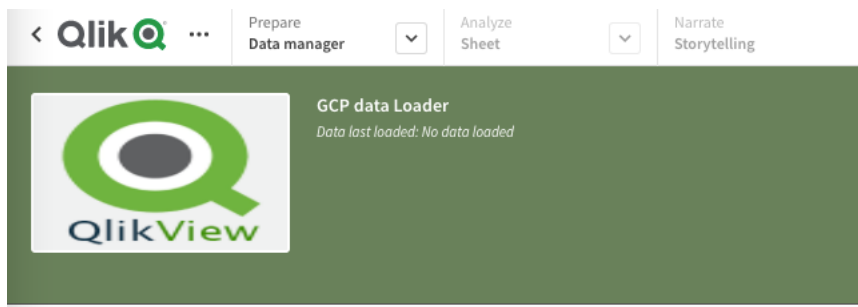
I mentioned earlier that the qlik app build and unbuild do not handle images in the media library. This can however be handled using the 'qlik app media' commands from qlik-cli.



Let's say we have a standard in our apps which display our corporate logo. We want to replace our old logo, which says QlikView, with our new logo.



For now, I'm going to update the logo in all my loader apps (i.e. apps with 'loader' in the name).



To do this I need to:

- Find the apps I want to update:

```
qlik app ls --limit 100|grep -i $APPSEARCH|sed -e's/ .*//'
```

- For each app, check if the image is present in the app and. If so, remove it:

```
IMAGES=$(qlik app media list get / --appId $APPID | jq -r '.[] | [.name]|@tsv')  
if grep -q "$IMAGE" <<< "$IMAGES"; then  
    eval "qlik app media file rm /$IMAGE --appId $APPID"
```

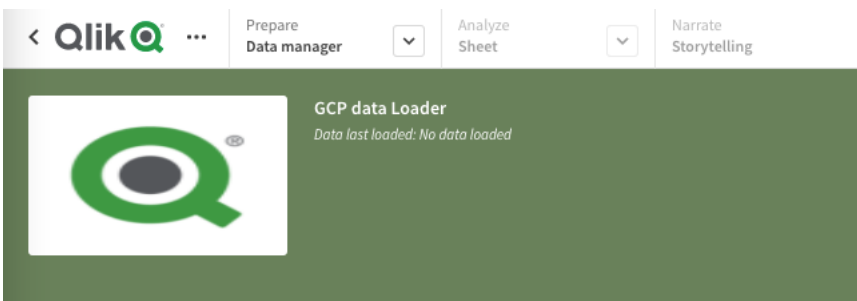
- Add the new logo image to the app:

```
qlik app media file update '/$IMAGE' --appId $APPID --file '$IMAGE'
```

See the script `distributeimage.sh` in the appendix implements this logic. It takes the pattern to match in the app name (in this case `*loader*`) and the file to distribute to the app:

```
lkn@APAC-lkn ~/dev/code/QSEoCS/unbuild$ ./distributeImage.sh loader logo.png
db589687-2ccb-419f-a01a-c9b06e1c201f
Deleting old image
Images in App GCP data Loader (db589687-2ccb-419f-a01a-c9b06e1c201f): logo.png
993daeab-d8f7-4d98-94f7-09b16169bec4
Deleting old image
Images in App Virginia data Loader (993daeab-d8f7-4d98-94f7-09b16169bec4): logo.png
d7fab019-87ff-4754-8a98-406c7a191656
Deleting old image
Images in App Azure data Loader (d7fab019-87ff-4754-8a98-406c7a191656): logo.png
16fba693-c013-4163-b12c-92a279f0013a
Deleting old image
Images in App Sydney data Loader (16fba693-c013-4163-b12c-92a279f0013a): logo.png
f788e01d-b289-40de-91a7-db72e4d818e5
Deleting old image
Images in App Ireland data Loader (f788e01d-b289-40de-91a7-db72e4d818e5): logo.png
lkn@APAC-lkn ~/dev/code/QSEoCS/unbuild$
```

We can now see the image in our apps have been updated:



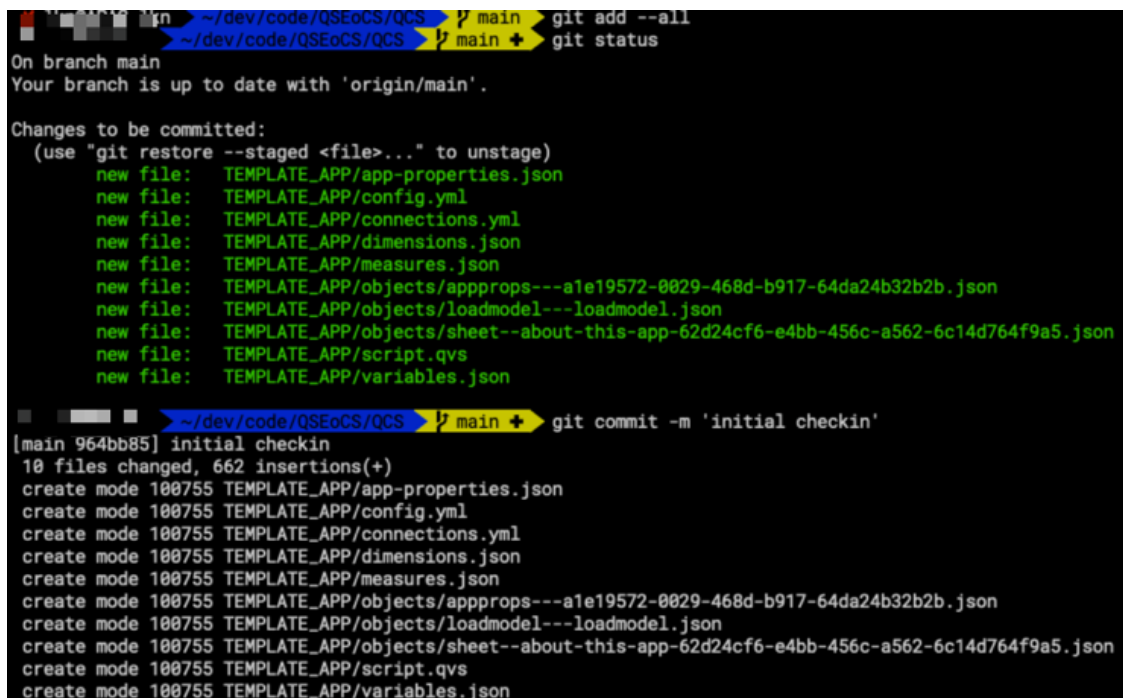
Working with GitHub

As we saw earlier the `qlik app unbuild` command deconstructs our qlik Sense app into its constituent parts. However this will recreate all the files, not just the changed files. So would this work with an SCM tool like GIT?

To test this I created a GitHub project called QCS and cloned it locally. I then ran and unbuild on my Template App we used earlier into the QCS git project:

```
qlik app unbuild -a 1aa4da26-8ca3-47b9-9881-7bdb402c8696 --dir ./QCS/TEMPLATE_APP
```

I then add this to git and commit it:



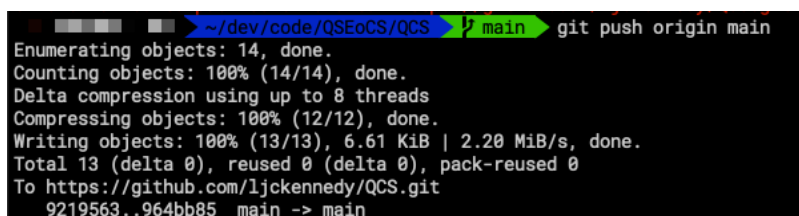
```
~/dev/code/QSEoCS/QCS main git add --all
~/dev/code/QSEoCS/QCS main git status

On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   TEMPLATE_APP/app-properties.json
    new file:   TEMPLATE_APP/config.yml
    new file:   TEMPLATE_APP/connections.yml
    new file:   TEMPLATE_APP/dimensions.json
    new file:   TEMPLATE_APP/measures.json
    new file:   TEMPLATE_APP/objects/appprops---a1e19572-0029-468d-b917-64da24b32b2b.json
    new file:   TEMPLATE_APP/objects/loadmodel---loadmodel.json
    new file:   TEMPLATE_APP/objects/sheet--about-this-app-62d24cf6-e4bb-456c-a562-6c14d764f9a5.json
    new file:   TEMPLATE_APP/script.qvs
    new file:   TEMPLATE_APP/variables.json

~/dev/code/QSEoCS/QCS main git commit -m 'initial checkin'
[main 964bb85] initial checkin
 10 files changed, 662 insertions(+)
 create mode 100755 TEMPLATE_APP/app-properties.json
 create mode 100755 TEMPLATE_APP/config.yml
 create mode 100755 TEMPLATE_APP/connections.yml
 create mode 100755 TEMPLATE_APP/dimensions.json
 create mode 100755 TEMPLATE_APP/measures.json
 create mode 100755 TEMPLATE_APP/objects/appprops---a1e19572-0029-468d-b917-64da24b32b2b.json
 create mode 100755 TEMPLATE_APP/objects/loadmodel---loadmodel.json
 create mode 100755 TEMPLATE_APP/objects/sheet--about-this-app-62d24cf6-e4bb-456c-a562-6c14d764f9a5.json
 create mode 100755 TEMPLATE_APP/script.qvs
 create mode 100755 TEMPLATE_APP/variables.json
```

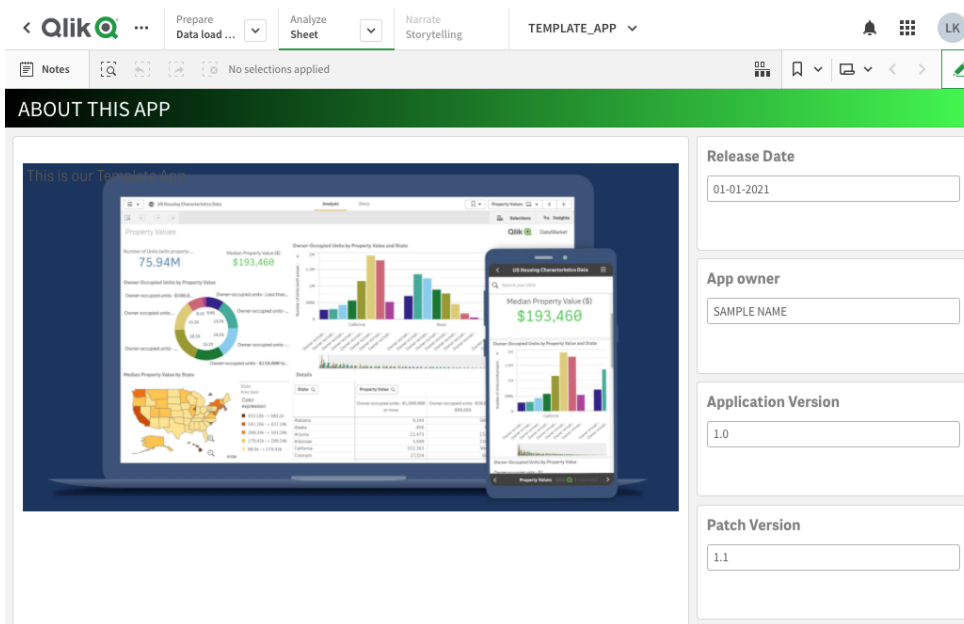
and push it to github:



```
~/dev/code/QSEoCS/QCS main git push origin main
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 8 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (13/13), 6.61 KiB | 2.20 MiB/s, done.
Total 13 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ljckennedy/QCS.git
 9219563..964bb85  main -> main
```

So what happens if I make a change?

I modified my Template app by adding an extra variable and showing those variables, as well as an image on my “About this App” sheet:



Now I'll unbuild the app again and run a git status to see what has changed:

```

~/dev/code/QSEoCS/QCS ~/main ▶ qlik app unbuild -a 1aa4da26-8ca3-47b9-9881-7bdb402c8696 --dir ./TEMPLATE_APP
"unbuild" command is experimental and it may change between releases
~/dev/code/QSEoCS/QCS ~/main ± ▶ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   TEMPLATE_APP/app-properties.json
    modified:   TEMPLATE_APP/objects/sheet--about-this-app-62d24cf6-e4bb-456c-a562-6c14d764f9a5.json
    modified:   TEMPLATE_APP/script.qvs
    modified:   TEMPLATE_APP/variables.json

no changes added to commit (use "git add" and/or "git commit -a")

```

I can see that the files I expect to have changed, did change, that is the script, the variables and the sheet. But what about the app properties? Why has that changed? Running git diff it shows me:

```

{
  "qTitle": "TEMPLATE_APP",
  - "qLastReloadTime": "2021-09-06T01:45:40.628Z",
  - "qSavedInProductVersion": "12.1085.0",
  + "qLastReloadTime": "2021-09-17T02:36:25.289Z",
  + "qSavedInProductVersion": "12.1107.0",
  "qThumbnail": {}
}

```

Ok, so I ran a reload to populate my variables so that makes sense. What is the 'qSavedInProductVersion'? I wasn't sure, so looking it up and I found it is defined as: *'Internal property reserved for app migration. The app is saved in this version of the product'*. So what has happened is in the week or so between the time I created this app and made these changes, there have been fixes and/or enhancements made to the engine. This is not likely to happen all the time, but it useful information in knowing what has changed beside my app code itself.

Now these are just a couple of examples of what we can do with qlik cli around our app lifecycle.

Some other ideas to consider include:

- Automatically comparing application code to copies stored in a git repository and updated git when the app changes
- Running code scanners against application scripts to ensure standards compliance.
- Propogating enhancements and bugfixes to shared libraries used by apps.

We at qlik see huge potential here for our customers and partners to use and expand these techniques and we at Qlik are looking to further enhancements in Qlik-cli which will make even more SDLC and DevOps use-cases possible. I encourage you to share your innovations in the Qlik Community as we would love to hear about them!

Appendix A: getSpaceDetails

```
//Configuration
Set vu_tenant_fqdn = 'mytenant.ap.qlikcloud.com';
Set vu_rest_connection = 'Development:REST_mytenant.ap.qlikcloud.com';
// End of Configuration

sub getSpaceId(vAppId, vSpaceId) //vu_tenant_fqdn, vu_rest_connection, vNextURL
    trace RUNNING:getSpaceId;
    if len('${vSpaceId}') > 0 then
        trace vSpaceId already set;
        Exit Sub;
    end if

    //set connection to use for this tenant.
    LIB Connect To '${vu_rest_connection}';

    // set the endpoint for app details
    Set baseUrl= "https://${vu_tenant_fqdn}/api/v1/apps/${vAppId}";

    RestConnectorMasterTable:
    SQL SELECT
        "__KEY_root",
        (SELECT
            "spaceId",
            "_resourcetype",
            "__KEY_attributes",
            "__FK_attributes",
            (SELECT
                "__FK_custom"
            FROM "custom" FK "__FK_custom")
        FROM "attributes" PK "__KEY_attributes" FK "__FK_attributes")
    FROM JSON (wrap on) "root" PK "__KEY_root"
    WITH CONNECTION (
        URL "${baseUrl}?${vNextURL}"
    );

    AppDetails:
    Load
        If(len(spaceId)=0,'Personal',spaceId) as SpaceId
    Resident RestConnectorMasterTable
    WHERE NOT IsNull([__FK_attributes]);

    DROP TABLE RestConnectorMasterTable;

    let vSpaceId = peek('SpaceId');
    //trace spaceId = ${vSpaceId};
    drop table AppDetails;
end sub;

sub getSpaceDetails(vSpaceName, vSpaceDescription, vSpaceType)
    trace RUNNING:getSpaceDetails;
    let vAppName = DocumentTitle();
    let vAppId = DocumentName();
    Set vNextURL = '';
    set vSpaceId = '';

    if len('${vSpaceName}') > 0 then
        trace vSpaceName already set;
        Exit Sub;
    end if
    call getSpaceId(vAppId, vSpaceId);
    if '${vSpaceId}' <> 'Personal' then

        // set the endpoint for space details
        Set baseUrl= "https://${vu_tenant_fqdn}/api/v1/spaces/${vSpaceId}";
    end if
end sub;
```

```

//LIB Connect To '${vu_rest_connection}';

RestConnectorMasterTable:
SQL SELECT
// "id",
    "type",
// "ownerId",
// "tenantId",
    "name",
    "description",
//     "createdAt",
//     "createdBy",
//     "updatedAt",
    "__KEY_root"
FROM JSON (wrap on) "root" PK "__KEY_root"
WITH CONNECTION (
    URL "${(baseUrl)}?${(vNextURL)}"
);

[space]:
LOAD
    [type] as spaceType,
    [name] as spaceName,
    [description] as spaceDescription
RESIDENT RestConnectorMasterTable
WHERE NOT IsNull([__KEY_root]);

DROP TABLE RestConnectorMasterTable;

let vSpaceType = peek('spaceType');
let vSpaceName = peek('spaceName');
let vSpaceDescription = peek('spaceDescription');
drop Table space;
else
    let vSpaceType = 'Personal';
    let vSpaceName = 'Personal';
    let vSpaceDescription = 'Personal';
end if

end sub;

sub getLibPath(vPath)
    trace 'getLibPath - ${vSpaceName}';
    if '${vSpaceName}' like '*Dev*' then
        trace This is a Dev space;
        set vPath='lib://common:DataFiles/library_1.2.QVS';
    elseif '${vSpaceName}' like '*TEST*' then
        trace This is a Test space;
        set vPath='lib://common:DataFiles/library_1.1.QVS';
    elseif '${vSpaceName}' like '*PROD*' then
        trace This is a Prod space;
        set vPath='lib://common:DataFiles/library_1.0.QVS';
    else
        trace Error: Non-standard space;
    end if
    trace ${vPath};
end sub;

sub getSpaceDatafileConn(vConn)
    trace 'getSpaceDatafileConn - ${vSpaceName}';
    if '${vSpaceName}' like '*Dev*' then
        trace This is a Dev space;
        set vConn='lib://Development:DataFiles/';
    elseif '${vSpaceName}' like '*TEST*' then
        trace This is a Test space;
        set vConn='lib://Test:DataFiles/';
    elseif '${vSpaceName}' like '*PROD*' then

```

```
    trace This is a Prod space;
    set vConn='lib://Production:DataFiles/';
else
    trace Error: Non-standard space;
end if
trace $(Conn);
end sub;
```

Example usage:

```
let vSpaceName = null();
let vSpaceDescription = null();
let vSpaceType = null();

call getSpaceDetails(vSpaceName, vSpaceDescription, vSpaceType);
trace #####;
trace spaceName = $(vSpaceName);
trace spaceDescription = $(vSpaceDescription);
trace SpaceType = $(vSpaceType);
trace #####;

let vPath = null();
call getLibPath(vPath);
trace Include: $(vPath);
$(Must_Include=$(vPath));
```

Appendix B: appbuild.sh

Usage:

```
./appbuild.sh appdir
```

Example:

```
./appbuild.sh myapp
```

Appbuild.sh:

```
#!/bin/bash
APPDIR=$1
NEWAPP=$(uuidgen)
unset OBJECTS
first=1
myobjects="./$APPDIR/myobjects.json"
echo "[" > $myobjects
for f in $(ls $APPDIR/objects); do
  if [ "$first" -eq "1" ]; then
    cat $APPDIR/objects/$f >> $myobjects
    first=0
  else
    echo ", '$'\n' >> $myobjects
    cat $APPDIR/objects/$f >> $myobjects
  fi
done
echo "]" >> $myobjects
OBJECTS="--objects $myobjects"

while read line; do
  if [[ ${line} != *"objects"* ]] && [[ ${line} != *"connections"* ]]; then
    conf=$(echo $line|sed -e "s/: / .\/$APPDIR\/\/")
    OBJECTS="$OBJECTS --$conf\"'$'\n'"
  fi
done < $APPDIR/config.yml
#OBJECTS="$OBJECTS\"'$'\n'"$myobjects\"'$'\n'"
#echo "qlik app build $OBJECTS -a $appid"

appid=$(qlik app create --attributes-name $NEWAPP --quiet)
eval "qlik app build $OBJECTS -a $appid"
exit;
```

Appendix C: appcreate.sh

Usage:

```
./ appcreate.sh appname
```

Example:

```
./appcreate.sh MYNEWAPP
```

appcreate.sh:

```
#!/bin/bash

APPDIR=APP_TEMPLATE
NEWAPP=$1

mv APP_TEMPLATE/script.qvs APP_TEMPLATE/oldscript.qvs
cat APP_TEMPLATE/oldscript.qvs| sed -e '/INCLUDES_HERE/ r INCLUDE.QVS' -e
'/INCLUDES_HERE/d' > APP_TEMPLATE/script.qvs

unset OBJECTS
first=1
myobjects="./$APPDIR/myobjects.json"
echo "[" > $myobjects
for f in $(ls $APPDIR/objects); do
  if [ "$first" -eq "1" ]; then
    cat $APPDIR/objects/$f >> $myobjects
    first=0
  else
    echo ","$'\n' >> $myobjects
    cat $APPDIR/objects/$f >> $myobjects
  fi
done
echo "]" >> $myobjects
OBJECTS="--objects $myobjects"

while read line; do
  if [[ ${line} != *"objects"* ]] && [[ ${line} != *"connections"* ]] ;then
    conf=$(echo $line|sed -e "s:/ ./\$/APPDIR\$/")
    OBJECTS="$OBJECTS --$conf\"$'\n'"
  fi
done < $APPDIR/config.yml

appid=$(qlik app create --attributes-name $NEWAPP --quiet)
eval "qlik app build $OBJECTS -a $appid"
eval "qlik app update $appid --attributes-name $NEWAPP"

rm APP_TEMPLATE/script.qvs
mv APP_TEMPLATE/oldscript.qvs APP_TEMPLATE/script.qvs
exit;
```

Appendix D: distributeimage.sh

```
#!/bin/bash

APPSEARCH=$1

IMAGE=$2
APPLIST=$(qlik app ls --limit 100|grep -i $APPSEARCH|sed -e's/ .*//')

#IFS=$'\n'
for APPID in $APPLIST; do
  echo $APPID
  IMAGES=$(qlik app media list get / --appId $APPID | jq -r '.[.] | [.name]@tsv')
  if grep -q "$IMAGE" <<< "$IMAGES"; then
    echo "Deleting old image"
    eval "qlik app media file rm /$IMAGE --appId $APPID"
    sleep 1s
  fi
  eval "qlik app media file update '/$IMAGE' --appId $APPID --file '$IMAGE'"
  echo "Images in App $(qlik app get $APPID |jq -r '.attributes.name') ($APPID): $(qlik
app media list get / --appId $APPID --show recursive | jq -r '.[.] | [.name]@tsv')"
done
```



About Qlik

Qlik's vision is a data-literate world, where everyone can use data and analytics to improve decision-making and solve their most challenging problems. Qlik provides an end-to-end, real-time data integration and analytics cloud platform to close the gaps between data, insights and action. By transforming data into active intelligence, businesses can drive better decisions, improve revenue and profitability, and optimize customer relationships. Qlik does business in more than 100 countries and serves over 50,000 customers around the world.

qlik.com

Qlik  LEAD WITH DATA™

© 2020 QlikTech International AB. All rights reserved. All company and/or product names may be trade names, trademarks and/or registered trademarks of the respective owners with which they are associated.